UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

INSTITUTO DE MATEMÁTICA

PROGRAMA DE PÓS GRADUAÇÃO

MATEMÁTICA EM REDE NACIONAL

MESTRADO PROFISSIONAL

SILVIO ROGÉRIO ALVES ESQUINCA

ALGORITMOS: RESOLUÇÃO DE PROBLEMAS BÁSICOS

CAMPO GRANDE

OUTUBRO DE 2014

UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL

INSTITUTO DE MATEMÁTICA

PROGRAMA DE PÓS GRADUAÇÃO

MATEMÁTICA EM REDE NACIONAL

MESTRADO PROFISSIONAL

SILVIO ROGÉRIO ALVES ESQUINCA

ALGORITMOS: RESOLUÇÃO DE PROBLEMAS BÁSICOS

Orientadora: Prof.ª Dr.ª Elisabete Sousa Freitas

Dissertação apresentada ao Programa de Pós-Graduação em Matemática em Rede Nacional do Instituto de Matemática IM/UFMS, como parte dos requisitos para obtenção do título de Mestre.

CAMPO GRANDE

OUTUBRO DE 2014

ALGORITMOS: RESOLUÇÃO DE PROBLEMAS BÁSICOS

SILVIO ROGÉRIO ALVES ESQUINCA

Dissertação submetida ao Programa de Pós-Graduação em Matemática em Rede Nacional, Instituto de Matemática, da Universidade Federal de Mato Grosso do Sul, como parte dos requisitos para obtenção do título de Mestre.

Aprovado pela Banca Examinadora:

Prof.ª Dr.ª Elisabete Souza Freitas - UFMS
Prof.ª Dr.ª Janete de Paula Ferrareze Silva- UFMS
Prof. Dr. Rui Seimetz - UNB

CAMPO GRANDE

OUTUBRO DE 2014

Dedico este trabalho aos meus pais, os quais, mesmo com suas limitações financeiras, fizeram o que estava em seu alcance, para que eu estudasse e tentasse ter um futuro melhor e mais dígno. Agradeço por terem me mostrado que o caminho da verdade, por mais árduo que seja, ainda é o melhor a ser seguido, mesmo havendo muitos obstáculos na busca da realização dos sonhos pessoais. Que Deus e nosso senhor Jesus Cristo os abençoe sempre.

Epígrafe

Mas os que esperam no senhor renovarão as suas forças, subirão com asas como águias, correrão e não se cansarão, caminharão e não se fatigarão.

(Isaías 40:31)

AGRADECIMENTOS

Primeiramente, agradeço a Deus e ao nosso Senhor Jesus Cristo, por terem me concedido a graça de ingressar e de prosseguir até o fim neste mestrado.

Agradeço ainda a minha esposa Josiane Colombo Pedrini Esquinca, por ajudar-me durante este curso, para que eu pudesse chegar até o fim dele.

Aos meus pais, por acreditarem em mim e por suas orações, as quais, com certeza, surtiram efeito, no sentido de eu ser aprovado nas disciplinas do curso e no exame de qualificação.

A minha professora e orientadora, Elisabete Souza Freitas, que, com respeito, determinação, dedicação e amparo, em todos os momentos da construção deste trabalho, dedicou-se, sem medir esforços, em orientar-me, para que este trabalho fosse concluído com êxito.

Ao professor Claudemir Aniz, por ter administrado este curso, conforme os princícpios que regem a administração pública, garantido seriedade e respeito em todas as etapas deste mestrado.

A todos os professores que fizeram parte deste curso, por terem me transmitido conhecimentos preciosos e importantes para o desempenho da função de professor de matemática.

Ao programa Profmat, por dar-me a oportunidade de adquir o título de mestre, almejado há muito tempo, o qual, com certeza, ampliará as oportunidades de trabalho, além de proporcionar-me uma maior bagagem de conhecimento.

A Capes, pelo incentivo e financiamento do curso.

Aos meus professores de graduação, por terem dado-me o conhecimento que garantiu minha profissão de professor e que me ajudou chegar até o Profmat. Em particular, agradeço a Antônio Carlos Tamarozzi, por ter acreditado em mim, orientando-me do início ao fim da graduação, sem medir esforços, incentivando-me a prosseguir nos estudos.

Aos colegas de turma de mestrado, pelo companheirismo e apoio nos momentos difíceis, que levou à troca de experiências.

Aos colegas de trabalho, lotados na Escola Estadual José Ferreira Barbosa, Mariomar Rezende Diniz Junior, Suely Oliveira de Assis, Eder Régis e Johnny Mattos, por todo o apoio prestado para que eu pudesse concluir este curso.

Aos colegas de trabalho, lotados na Escola Municipal Professor Plínio Mendes dos Santos, Osvaldo Alves Pinto, Rosângela Cristina Ferreira Lino, Marinês, por todo o apoio prestado para que eu pudesse concluir este curso.

Resumo

O objetivo deste trabalho é apresentar uma introdução ao estudo de algoritmo. Através de exemplos, o custo de algoritmos será analisado de forma simplificada. Serão discutidos o algoritmo da divisão, o algoritmo de Strassen, para a multiplicação de matrizes, e alguns problemas clássicos da computação.
Palavras-chave: Algoritmo. Algoritmo da divisão. Algoritmo de Strassen.

Abstract

The objective of this work is to present an introduction to the study of algorithm. Through examples, cost algorithms will be discussed in simplified form. Discussed the division algorithm, Strassen's algorithm for matrix multiplication, and some classical problems of computing.
Keywords: Algorithm. Division algorithm. Strassen algorithm.

Sumário

Capítulo 1

Introdução

O objetivo principal desse trabalho é apresentar uma motivação para o estudo da Matemática, através de uma iniciação ao estudo de algoritmos.

Já no ensino básico, aprende-se algoritmos para solucionar certos problemas. Por exemplo, cálculo do quociente e do resto da divisão de um inteiro por outro (algoritmo da divisão) ou cálculo do máximo divisor comum de dois inteiros (algoritmo euclidiano). Estes dois algoritmos já aparecem descritos nos *Elementos de Euclides*, escrito por volta de 300 a.C.

Um algoritmo representa os passos necessários para realizar uma tarefa. Sua execução pode ser feita por computadores, capazes de efetuar enormes cálculos, ou por pessoas.

Atualmente existem nas escolas laboratórios de computação e alunos e professores são estimulados a usar programas ou softwares para auxiliar no processo de ensino-aprendizagem. Um programa de computador é essencialmente um algoritmo que diz ao computador os passos específicos e em que ordem eles devem ser executados, como por exemplo, os passos a serem tomados para calcular o máximo divisor comum de dois inteiros. Outro exemplo, os passos para calcular as médias finais dos alunos de uma escola.

Antes de executar um programa em um computador, o programa deve ser codificado em uma linguagem de programação da escolha do programador. E, antes de codificar o programa, o programador deve ter escolhido (criado, descoberto) um algoritmo para a resolução do problema. Portanto, a criação de um algoritmo precede a própria escrita do programa na linguagem de programação. Antes de criar um algoritmo para resolver um problema, ou escolher um algoritmo já pronto, não há como codificá-lo numa linguagem de programação, muito menos como executá-lo em um computador.

Os algoritmos podem realizar a mesma tarefa, cada um gastando mais ou menos tempo, espaço ou esforço do que o outro. Para qualquer processo computacional, o algoritmo precisa estar bem definido, a corretividade do algoritmo precisa ser estudada matematicamente, a quantidade assintótica de tempo e espaço (custo) necessários para a sua execução deve ser analisada.

Uma introdução sobre algoritmos é apresentada no capítulo 2, tratando do conceito, representação, custo de execução e eficiência de algoritmos.

No capítulo 3, são discutidos, além de outros exemplos, o algoritmo da divisão, de dois números naturais e o algoritmo de Strassen, para a multiplicação de matrizes.

No capítulo 4, são apresentados alguns problemas clássicos da computação.

Capítulo 2

Algoritmos

Informalmente, um algoritmo é uma sequência precisa (sem dúvidas sobre o que deve ser feito) de instruções que, se lida e executada por qualquer pessoa ou por um computador, produz o resultado esperado, isto é, a solução de um problema. Esta sequência de instruções é uma descrição da sequência dos passos necessários, os quais devem ser executados, para manipular informações, ou dados, para se chegar na resposta do problema.

Mais formalmente, um algoritmo é uma sequência finita $A = (i_1,\ i_2,\ i_3, ...,\ i_n)$ de instruções $i_k,\ k \in \{1,\ 2, ...,\ n\}$, tais que:

1. i_k não apresenta ambiguidade, $\forall k \in \{1,\ 2, ...,\ n\}$;
2. depois da execução de i_k, não haverá ambiguidade sobre $i_j,\ j \in \{k+1,\ k+2, ...,\ n\}$;
3. a instrução de parar é sempre alcançada depois da execução de um número finito de instruções; e
4. há produção de resultado correto para o problema a ser solucionado.

Exemplo 1. Abaixo estão as instruções inscritas em uma placa de metal anexada à frente de um videogame:

Passo 1 (i_1)**:** Insira vinte e cinco centavos na fenda para moedas ao lado da máquina.

Passo 2 (i_2)**:** Pressione o botão verde no topo da máquina quando estiver pronto para iniciar.

Veja que existem duas instruções, i_1 e i_2, bem definidas, sem ambiguidades. Há clareza sobre o próximo passo a ser executado (i_2). A instrução de parar foi omitida, pois fica claro que cada instrução é executada apenas uma vez para cada jogo. Isto permite dizer, conforme a definição, que tais passos representam um algoritmo para iniciar um jogo, sendo este indicado por $A(i_1,\ i_2)$.

Exemplo 2. Suponha que uma terceira instrução, i_3, seja adicionada no exemplo 1, conforme o que segue abaixo:

Passo 3 (i_3)**:** Quando cada jogo for finalizado, escreva suas iniciais e pressione o botão vermelho no topo da máquina para gravar seus pontos obtidos para a posteridade.

O passo 3 cumpre o papel de uma instrução de parada do jogo fornecido pela máquina, para não durar para sempre, estando de acordo com a terceira propriedade da definição formal de algoritmo, e mostra que você não poderá efetuar um segundo jogo com o dinheiro pago inicialmente para o primeiro jogo.

Exemplo 3. Suponha ainda que seja inserida a seguinte instrução, i_4, no exemplo 1:

Passo 4 (i_4)**:** Para cada 10.000 pontos que você acumular, ganhará um jogo grátis. Quando o jogo em execução finalizar, se você tem direito a um jogo grátis, vá para o passo 2 (i_2).

O passo 4 contraria a definição de algoritmo, pois seria possível obter a sequência infinita $A = (i_1,\ i_2,\ i_3,\ i_4,\ i_2,\ i_3,\ i_4,\ i_2,\ i_3,\ i_4, ...)$. Portanto, $A = (i_1,\ i_2,\ i_3,\ i_4)$ não representa um algoritmo. O passo 4 é considerado como *loop* (laço condicional), por permitir a execução de um passo mais do que uma vez, toda vez que a afirmação dele for verdadeira. Os exemplos 1 e 2 são chamados de algoritmos sequenciais, pois não permitem o retorno para a execução de uma instrução anterior, mais de uma vez.

Para transformar a inclusão e permitir que $A = (i_1,\ i_2,\ i_3,\ i_4)$ seja um algoritmo, pode-se reescrevê-lo conforme o que segue abaixo.

Exemplo 4. Suponha que a instrução i_4 tenha sido modificada, conforme o que segue:

Passo 4 (i_4)**:** Para cada 10.000 pontos que acumular, você ganhará um jogo de graça, sendo o número máximo de jogos gratuitos ganhos igual a 10 para cada jogo pago. Quando o jogo em execução finalizar, se você tem direito a um jogo grátis, vá para o passo 2.

Veja agora que o novo passo 4 (*loop*) será executado uma quantidade finita de vezes. Agora, nestas condições, $A = (i_1,\ i_2,\ i_3,\ i_4)$ é um algoritmo.

■

A sequência de instruções não será um algoritmo correto se faltarem instruções necessárias para a produção da solução do problema.

Suponha que x e y sejam duas variáveis assumindo os respectivos valores 5 e 2. É uma tarefa comum trocar os valores das variáveis, como se estivesse arranjando os números 2 e 5 para as posições x e y, fixadas. Neste caso, após a troca, tem-se que $x = 2$ e $y = 5$.

Exemplo 5. Os passos abaixo representam uma forma para se trocar os valores assumidos pelas variáveis x e y.

♦ Passo 1: Faça x assumir o valor de y.
♦ Passo 2: Faça y assumir o valor de x.
♦ Passo 3: Pare.

Será que o algoritmo deste exemplo está correto? Veja como ele funcionará para a suposição dada acima.

Instrução	*Valor assumido para x*	*Valor assumido para y*
Antes de i_1	5	2
Depois de i_1	2	2
Depois de i_2	2	2

Por que depois do passo 2 a variável y não assumiu o valor 5? Acontece que depois do passo 1 o valor 5 foi esquecido. Mas é preciso obter $y = 5$, depois do passo 2. Porém, como fazer isso se o velho valor de x, 5, foi regravado para o valor 2? Uma solução para tal problema é apresentada no próximo exemplo.

Exemplo 6. Segue abaixo uma sequência de instruções que realizará a troca dos valores conforme desejado.

- Entrada: x e y.
- Saída: x e y com valores trocados.
- Passo 1: Atribua o valor de y a uma variável auxiliar chamada *y_ velho.*
- Passo 2: Faça y assumir o valor de x ($y := x$).
- Passo 3: Faça x assumir o valor de *y_ velho.*
- Passo 4: Pare.

■

O resultado obtido é mostrado logo abaixo.

Passos	*Valor assumido para x*	*Valor assumido para y*
Passo 1	5	2
Passo 2	5	5
Passo 3	2	5
Passo 4	Pare.	

Para solucionar um problema, um algoritmo precisa de uma entrada, representada por uma ou mais variáveis, relacionadas com o problema. Após a execução de todas as instruções, o algoritmo produz uma saída que é a solução para o problema.

Exemplo 7. (*Conjectura de Lothar Collatz*) Considere o seguinte procedimento:

- Entrada: O número positivo z.
- Saída: O número 1.
- Passo 1: Entre com um número inteiro positivo z.
- Passo 2: Se z for par, troque o valor de z por $\frac{z}{2}$.
- Passo 3: Se $z = 1$, então a saída será z e pare.
- Passo 4: Se z for ímpar, troque o valor de z por $3z + 1$.
- Passo 5: Vá para i_2.

Este é um problema proposto pelo alemão *Lothar Collatz* para o qual não se sabe ainda se a sequência de números produzida findará em 1, para todo número inteiro não negativo. Ele só será um algoritmo, se houver parada com produção de resultado final $z = 1$.

Exemplo 8. Executando a conjectura de Collatz para $z = 1$, $z = 20$ e $z = 7$, tem-se:

Sequência obtida para $z = 1$: (1).

Sequência obtida para $z = 20$: (20, 10, 5, 16, 8, 4, 2, 1).

Sequência obtida para $z = 7$: (7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1).

■

Exemplo 9. A sequência $(1, 1, 2, 3, 5, 8, 13, 21, ...)$ é composta pelos números de Fibonacci, obedecendo à recorrência $a_n = a_{n-1} + a_{n-2}$, para $a_1 = a_2 = 1$ e $n \geqslant 3$, ou seja, para se obter o n-ésimo termo desta sequência, recorre-se aos dois termos anteriores a a_n, somando-os para dar o valor de a_n. Segue abaixo um algoritmo que retorna os n primeiros números da sequência de Fibonacci, para $n \geqslant 3$.

- Entrada: O número n, sendo $n \geqslant 3$.
- Saída: Os n primeiros números da sequência de Fibonacci.
- Passo 1: Considere $a_1 = a_2 = 1$.
- Passo 2: Para $i := 3$ até n, faça $a_i := a_{i-1} + a_{i-2}$.
- Passo 3: Para $i := 1$ até n, retorne a_i.
- Passo 4: Pare.

Exemplo 10. Uma sequência finita de números inteiros $(a_1, a_2, ..., a_n)$ será crescente se $a_1 \leqslant a_2 \leqslant a_3 \leqslant ... \leqslant a_n$. O algoritmo a seguir lê os números de uma sequência finita e diz se ela é crescente ou não.

- Entrada: A quantidade n de elementos da sequência e os seus elementos: $a_1, a_2, ..., a_n$.
- Saída: Apenas uma das duas frases: "Esta sequência não é crescente" ou "Esta sequência é crescente".
- Passo 1: Para $i := 1$ até $n - 1$, se $a_i > a_{i+1}$ escreva: "Esta sequência não é crescente". Caso contrário, escreva: "Esta sequência é crescente".
- Passo 2: Pare.

Para compreender melhor como são executadas as instruções do algoritmo, pode-se efetuar o chamado teste de mesa, que consiste na execução mental desse algoritmo, o qual também é chamado de código, sendo os passos anotados, em sequência de execução, no papel. Nesse teste, a pessoa será o processador e ficará responsável por executar cada linha do código. O teste permite verificar se o código está realmente executando a função para a qual foi criado, e se existe algum erro de lógica nos passos apresentados.

O teste surgiu em uma época na qual programar um computador era extremamente complicado. Apenas aceitar que um código estava correto era um desperdício de tempo e, portanto, era necessário ter certeza de que aquele programa funcionaria. Munidos, então, de papel e lápis, os programadores faziam simulações de execução e corrigiam erros encontrados antes mesmo de executar o código na máquina real. Essa prática se perdeu com o tempo, com o advento da programação interativa, na qual o programador tem acesso à máquina e pode testar alterações.

A vantagem desse tipo de teste é que ele ajuda a desenvolver o raciocínio lógico do programador, que, depois de um certo tempo de prática, consegue identificar problemas apenas olhando para o código impresso, sem necessitar executá-lo.

Exemplo 11. Dada uma lista de números reais: $a_1, a_2, ..., a_n$, a média aritmética dos números desta lista é dada por $MA = \dfrac{a_1 + a_2 + ... + a_n}{n}$. Veja abaixo, à esquerda, o algoritmo que calcula a média aritmética de tal lista.

- Entrada: n e os valores de $a_1,\ a_2, ...,\ a_n$. - Saída: A média aritmética dos n números. - Passo 1: Considere $x = 0$. - Passo 2: Para cada valor de i, variando de 1 até n, faça x assumir o valor do antigo x somado com a_i ($x := x + a_i$). - Passo 3: Divida x por n e considere o resultado obtido como MA $\left(MA = \frac{x}{n}\right)$. - Passo 4: Retorne MA e pare.	Veja os passos executados logo abaixo: - Passo 1: $x := 0$. - Passo 2: $i := 1$ e $x := 0 + a_1 = a_1$. $i := 2$ e $x := \underbrace{a_1}_{x\ velho} + a_2$. $i := 3$ e $x := \underbrace{a_1 + a_2}_{x\ velho} + a_3$ $\vdots$ $i := n$ e $x :=$ $= \underbrace{a_1 + a_2 + ... + a_{n-1}}_{x\ velho} + a_n$ - Passo 3: $MA := \frac{x}{n}$. - Passo 4: Retorne MA.

Exemplo 12. Para calcular a distância entre dois pontos do plano: $A(x_1,\ y_1)$ e $B(x_2,\ y_2)$, utiliza-se a fórmula $D_{A,B} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, na qual $D_{A,B}$ indica a distância entre os pontos A e B. As instruções abaixo representam um algoritmo para calcular essa distância:

♦ Entrada: Considere os valores: $x_1,\ x_2,\ y_1$ e y_2 (entrada).
♦ Saída: A distância entre os dois pontos do plano que possuem essas coordenadas.
♦ Passo 1: $a := x_1 - x_2$
♦ Passo 2: $b := y_1 - y_2$
♦ Passo 3: $a := a * a$ (novo valor de a)
♦ Passo 4: $b := b * b$ (novo valor de b)
♦ Passo 5: $m := a + b$
♦ Passo 6: $D_{A,B} := sqrt(m)$
♦ Passo 7: Escreva o valor de $D_{A,B}$ (saída).
♦ Passo 8: Pare.

Observação 1. $sqrt(m)$ significa $\sqrt{m}$.

2.1 Algoritmos para expansão na base decimal e binária

O sistema decimal posicional é utilizado universalmente pelas pessoas para representar os números inteiros. Chama-se decimal por ser representado pelos dez algarismos: 0, 1, 2, 3, 4, 5, 6, 7, 8 e 9, chamados de algarismos indo-arábicos, pois foram desenvolvidos pelo povo indiano e divulgados pelo povo árabe.

Existem outros sistemas de numeração em uso, como o sistema binário ou em bases potências de 2, que é utilizado em computação. Alguns computadores, para executar um algoritmo, convertem as palavras deste algoritmo para código binário. Veja a seguir a tabela com alguns códigos binários e suas representações:

Binário	***Glifo***
0010 1011	+
0010 1111	/
0011 0000	0
0011 0001	1
0011 0010	2
0011 0011	3

Binário	***Glifo***
0011 0100	4
0011 0101	5
0011 0110	6
0011 0111	7
0011 1000	8
0011 1001	9

O sistema também é chamado posicional, pois cada algarismo, além do seu valor intrínseco, possui um peso que lhe é atribuído em função da posição que ele ocupa no número. No caso do sistema decimal, esse peso, é uma potência de dez e ele varia do seguinte modo:

O algarismo da extrema direita tem peso 10^0 (posição 0); o seguinte, sempre da direita para a esquerda, tem peso 10^1 (posição 1); o seguinte tem peso 10^2 (posição 2); o seguinte tem peso 10^3 (posição 3), etc.

Exemplo 13. Considere o número 12019 na base 10. Assim,

$$12019 = 1 \cdot 10^4 + 2 \cdot 10^3 + 0 \cdot 10^2 + 1 \cdot 10^1 + 9 \cdot 10^0 = 1 \cdot 10^4 + 2 \cdot 10^3 + 1 \cdot 10 + 9$$

Os sistemas de numeração posicionais baseiam-se no teorema a seguir, que é uma aplicação da divisão euclidiana.

Teorema 1. Dados $a,\ b \in \mathbb{N} \cup \{0\}$, com $b > 1$, existem números naturais $c_0,\ c_1, ..., c_n$, menores do que b, univocamente determinados, tais que $a = c_0 + c_1 b + c_2 b^2 + ... + c_n b^n$.

Demonstração:

O teorema será demonstrado por meio da segunda forma do Princípio de Indução Matemática, sobre a. Se $a = 0$, ou se $a = 1$, basta tomar $n = 0$ e $c_0 = a$, e vale a unicidade.

Supondo o resultado válido para todo natural menor do que $a \geqslant 1$, será provado sua validade também para a. Pela divisão euclidiana, existem q e r, únicos, tais que

$$a = bq + r, \text{ com } 0 \leqslant r < b.$$

Tem-se:

$$b > 1 \Rightarrow ab > a \Rightarrow a < ab \leqslant ab + r \Rightarrow bq + r < ab + r \Rightarrow q < a.$$

Como $q < a$, pela hipótese de indução, segue que existem números naturais n' e $d_0, d_1, ..., d_{n'}$, com $d_j < b$, para todo j , univocamente determinados, tais que

$$q = d_0 + d_1 b + \cdots + d_{n'} b^{n'}.$$

Levando em conta as igualdades acima destacadas, tem-se que

$$a = bq + r = b(d_0 + d_1 b + \cdots + d_{n'} b^{n'}) + r.$$

O resultado segue tomando $c_0 = r,\ n = n' + 1$ e $c_j = d_{j-1}$ para $j = 1, ..., n$, os quais são univocamente determinados.

■

Corolário 1. Todo número natural se escreve de modo único como soma de potências distintas de 2.

A representação dada no teorema acima é chamada de expansão relativa à base b. Quando $b = 10$, essa expansão é chamada expansão decimal, e quando $b = 2$, ela recebe o nome de expansão binária.

A demonstração do Teorema também fornece um algoritmo para determinar a expansão de um número qualquer relativamente à base b.

Trata-se de aplicar, sucessivamente, a divisão euclidiana, como segue:

$$a = bq_0 + r_0, \quad r_0 < b,$$

$$q_0 = bq_1 + r_1, \quad r_1 < b,$$

$$q_1 = bq_2 + r_2, \quad r_2 < b,$$

e assim por diante. Como $a > q_0 > q_1 > \cdots$, deve-se, em um certo ponto, ter $q_{n-1} < b$ e, portanto, de

$$q_{n-1} = bq_n + r_n,$$

decorre que $q_n = 0$, o que implica $0 = q_n = q_{n+1} = q_{n+2} = \cdots$, e portanto

$$0 = r_{n+1} = r_{n+2} = \cdots.$$

Tem-se então que

$$a = r_0 + r_1 b + \cdots + r_n b^n.$$

A expansão numa dada base b fornece um método para representar os números naturais. Para tanto, escolha um conjunto S tal que $S = \{1, 2, 3, ..., (b-1)\}$. Um número natural a na base b se escreve da forma $x_n x_{n-1} ... x_1 x_0$, com $x_0, ..., x_n \in S$, e n variando, dependendo de a, representando o número $x_0 + x_1 b + \cdots + x_n b^n$.

No sistema decimal, isto é, de base $b = 10$, usa-se $S = \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9\}$.

Exemplo 14. A expansão decimal do número 23456 é dada por:

$$2 \cdot 10^4 + 3 \cdot 10^3 + 4 \cdot 10^2 + 5 \cdot 10^1 + 6 \cdot 10^0.$$

Exemplo 15. No sistema de base $b = 2$, tem-se que $S = \{0,\ 1\}$ e todo número natural é representado por uma sequência de 0 e 1. Por exemplo, o número 10, na base 2, representa o número 2 (na base 10). Tem-se também que

$$100 = 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 2^2,\ 101 = 1 + 2^2,\ 111 = 1 + 2 + 2^2,\ 1011 = 1 + 2 + 2^3.$$

■

A sequência de instruções abaixo é um algoritmo para converter número binário para a forma decimal equivalente.

Binário para Decimal

- Entrada: Número binário $s = s_j s_{j-1} ... s_1 s_0$.
- Saída: Número decimal m equivalente à s.
- Passo 1: Comece considerando $j = 0$, representando a posição do primeiro algarismo de s, da direita para a esquerda.
- Passo 2: Considere também $m = 0$. Ele representará o número decimal equivalente aos algarismos obtidos, da direita para a esquerda, até a posição j.
- Passo 3: Se a posição j não possuir algarismo então pare e retorne o valor de m como resposta. Caso contrário, continue.
- Passo 4: Se o algarismo da posição j for 1 então o número decimal m será dado por $\underbrace{m}_{\text{Novo}} = \underbrace{m}_{\text{Velho}} + 2^j$.
- Passo 5: Incremente j, ou seja, aumente uma unidade no j anterior, para trabalhar com mais algarismos de s.
- Passo 6: Volte a executar o passo 3.

Exemplo 16. Convertendo o número binário $s = 110$ para a forma decimal equivalente, seguindo os passos dados logo acima, tem-se:

- Passo 1: $j := 0$.
- Passo 2: $m := 0$.
- Passo 3: Existe algarismo na posição $j = 0$, o zero. Então continue.
- Passo 4: O algarismo da posição $j = 0$ não é 1 ($s_0 \neq 1$).
- Passo 5: $j := 1$.
- Passo 6: Volte a executar o passo 3.
- Passo 3: Existe algarismo na posição $j = 1$, $s_1 = 1$. Então continue.
- Passo 4: O algarismo da posição $j = 1$ é 1. Então m passa a valer $m + 2^j = 0 + 2^1 = 2$, ou seja, $m := 2$.
- Passo 5: $j := 2$.
- Passo 6: Volte a executar o passo 3.
- Passo 3: Existe algarismo na posição $j = 2$, $s_2 = 1$. Então continue.
- Passo 4: O algarismo da posição $j = 2$ é 1. Então $m := 6$.
- Passo 5: $j := 3$.
- Passo 6: Volte a executar o passo 3.
- Passo 3: Não existe s_3. Então pare e a resposta procurada é $m = 6$, na forma decimal.

■

Pode-se representar tais passos da seguinte forma simplificada, chamada de pseudocódigo.

Algoritmo Binário para Decimal

–> Entrada: O número binário $s = s_j s_{j-1} ... s_1 s_0$, diferente de zero.
i_1: $j := 0$.
i_2: $m := 0$
i_3: Se não existir s_j então pare e retorne m como resposta.
i_4: Se $s_j = 1$ então $m := m + 2^j$.
i_5: $j := j + 1$.
i_6: Vá para i_3.

Para converter um número decimal para a forma binária equivalente, utilizando o que foi comentado anteriormente, obtém-se o seguinte esquema na tabela abaixo:

Conversão de Número Decimal para Binário - Algoritmo

Número a ser dividido por 2	*Quociente obtido*	*Resto obtido*
m	q_0	r_0
q_0	q_1	r_1
q_1	q_2	r_2
q_2	q_3	r_3
$\vdots$	$\vdots$	$\vdots$
q_{n-2}	q_{n-1}	r_{n-1}
q_{n-1}	0	r_n

Assim, os algarismos que formarão o número binário serão todos os restos obtidos na terceira coluna da tabela acima, ou seja, tal número binário será $r_n r_{n-1} ... r_3 r_2 r_1 r_0$, nessa ordem de posição.

Exemplo 17. Segue abaixo a conversão do número decimal 210 para a forma binária equivalente, conforme a tabela acima.

Número a ser dividido por 2	*Quociente obtido*	*Resto obtido*
210	105	0
105	52	1↑
52	26	0↑
26	13	0↑
13	6	1↑
6	3	0↑
3	1	1↑
1	0	1↑

Conforme a tabela deste exemplo, o número binário obtido é: 11010010.

A tabela dada antes deste exemplo permite a construção de uma sequência finita de instruções ou passos, bem definidos, que representam um algoritmo para converter número decimal para a forma binária equivalente, conforme o que segue abaixo.

Decimal para Binário

- ♦ Entrada: O número decimal m.
- ♦ Saída: O número binário $s = s_j s_{j-1} ... s_1 s_0$.
- ♦ Passo 1: Considere $j = 0$, o qual representa a posição do primeiro algarismo do resultado, da direita para a esquerda.
- ♦ Passo 2: Divida m por 2, obtendo quociente q_j e resto r_j (0 ou 1).
- ♦ Passo 3: Insira r_j na posição j de s, ou seja, faça $s_j := r_j$.
- ♦ Passo 4: Se $q_j = 0$ então pare e retorne s como resposta. Caso contrário, continue.
- ♦ Passo 5: Faça m assumir valor q_j, ou seja, $m := q_j$.
- ♦ Passo 6: Incremente o valor de j, somando a ele uma unidade.
- ♦ Passo 7: Volte a executar o passo 2.

Exemplo 18. Segue abaixo a conversão do número decimal $m = 21$ para a forma binária equivalente, aplicando os passos do algoritmo logo acima.

- ♦ Passo 1: Considere $j = 0$.
- ♦ Passo 2: $m = 21$ dividido por 2 fornece quociente $q_0 = 10$ e resto $r_0 = 1$.
- ♦ Passo 3: s_0 assume valor $r_0 = 1$, ou seja, $s_0 := 1$.
- ♦ Passo 4: $q_0 \neq 0$. Então continue.
- ♦ Passo 5: m assume o valor de q_0, isto é, $m := 10$.
- ♦ Passo 6: j é incrementado de modo que $j := 1$.
- ♦ Passo 7: Volte a executar o passo 2.
- ♦ Passo 2: $m = 10$ dividido por 2 fornece quociente $q_1 = 5$ e resto $r_1 = 0$.
- ♦ Passo 3: s_1 assume valor $r_1 = 0$, ou seja, $s_1 := 0$.
- ♦ Passo 4: $q_1 \neq 0$. Então continue.
- ♦ Passo 5: m assume o valor de q_1, isto é, $m := 5$.
- ♦ Passo 6: $j := 2$.
- ♦ Passo 7: Volte a executar o passo 2.
- ♦ Passo 2: $m = 5$ dividido por 2 fornece quociente $q_2 = 2$ e resto $r_2 = 1$.
- ♦ Passo 3: $s_2 := r_2 = 1$.
- ♦ Passo 4: $q_2 \neq 0$. Então continue.
- ♦ Passo 5: $m := q_2 = 2$.
- ♦ Passo 6: $j := 3$.
- ♦ Passo 7: Volte a executar o passo 2.
- ♦ Passo 2: $m = 2$ dividido por 2 fornece quociente $q_3 = 1$ e resto $r_3 = 0$.
- ♦ Passo 3: $s_3 := r_3 = 0$.
- ♦ Passo 4: $q_3 \neq 0$. Então continue.
- ♦ Passo 5: $m := q_3 = 1$.
- ♦ Passo 6: $j := 4$.
- ♦ Passo 7: Volte a executar o passo 2.
- ♦ Passo 2: $m = 1$ dividido por 2 fornece quociente $q_4 = 0$ e resto $r_4 = 1$.
- ♦ Passo 3: $s_4 := r_4 = 1$.
- ♦ Passo 4: $q_4 = 0$. Então a saída é $s = s_4 s_3 s_2 s_1 s_0 = 10101$.

■

Tem-se abaixo o pseudocódigo para esse procedimento.

Procedimento Binário para Decimal

i_1: $j := 0$
i_2: Divida m por 2, para obter o quociente q_j e o resto r_j (0 ou 1).
i_3 : Coloque r_j na posição j de $s = s_j s_{j-1}...s_1 s_0$.
i_4: Se $q_j = 0$, então pare e retorne s como resposta. Caso contrário, continue.
i_5: $m := q_j$.
i_6: $j := j + 1$.
i_7: Vá para i_2.

2.2 Algoritmo para determinação dos subconjuntos de um conjunto

Proposição 1. Dado um conjunto A, com n elementos, então o número de subconjuntos de A é 2^n.

Demonstração:

Por Indução Finita sobre n, tem-se:

$P(n)$: Se A possui n elementos, então o número de subconjuntos de A é 2^n. Dessa forma, para $n = 1$, tem-se:

$A = \{a_1\}$ de modo que $\emptyset$ e $\{a_1\}$ são os subconjuntos de A. Como $2^n = 2^1 = 2$, $P(1)$ é verdadeira.

Supondo $P(n)$ verdadeira para algum número natural $n \geqslant 1$, será provado que $P(n+1)$ também é verdadeira, ou seja, que se A tem $n+1$ elementos então A possuirá 2^{n+1} subconjuntos. Tem-se que:

$$A = \{a_1,\ a_2, ..., a_n,\ a_{n+1}\} = \underbrace{\{a_1,\ a_2, ..., a_n\}}_{= B} \cup \{a_{n+1}\}.$$

Como o número de elementos de B é igual a n, segue, por hipótese de indução, que B possui 2^n subconjuntos: $B_1 = \emptyset$, B_2,..., B_{2^n}, os quais também são subconjuntos de A, já que $B_i \subset B \subset A,\ \forall i \in \{1, 2, ..., 2^n\}$. Além disso, $a_{n+1} \notin B_i,\ \forall i \in \{1, 2, ..., 2^n\}$. Para construir os subconjuntos restantes de A, basta inserir o elemento a_{n+1} em cada subconjunto B_i, obtendo-se 2^n novos subconjuntos. Assim, o total de subconjuntos de A é $2^n + 2^n = 2^n.(1 + 1) = 2^n.\ 2 = 2^{n+1}$. Portanto $P(n+1)$ é verdadeira.

A demonstração da proposição 1 fornece uma sequência finita de passos, bem definidos, para produzir os subconjuntos de A, escrita na forma de um pseudocódigo.

Algoritmo Subconjuntos

i_1: **Entrada**: n e $a_1,\ a_2, ...,\ a_n$.
i_2: $B(1) = \emptyset$
i_3: $k := 1$
i_4: **Para** $j := 1$ **até** n, **faça**
i_5: **Para** $i := 1$ **até** 2^{j-1}, **faça**
i_6 : $B(k+1) := B(i) \cup \{a(j)\}$
i_7 : $k := k + 1$
FimPara
FimPara
i_8 : **Pare.**

Exemplo 19. Fazendo o teste de mesa aplicando o *Algoritmo Subconjuntos* em $A = \{a_1,\ a_2\}$, para listar os subconjuntos de A, tem-se:

Instrução Executada	k	j	i	*Subconjunto Obtido*
$\vdots$				
i_2				$B(1) = \emptyset$
i_3	1			
i_4	1	1		
i_5	1	1	1	
i_6	1	1	1	$B(2) = B(1) \cup \{a(1)\} = \{a(1)\}$
i_7	2	1	1	
i_4	2	2	1	
i_5	2	2	1	
i_6	2	2	1	$B(3) = B(1) \cup \{a(2)\} = \{a(2)\}$
i_7	3	2	1	
i_5	3	2	2	
i_6	3	2	2	$B(4) = B(2) \cup \{a(2)\} = \{a(1),\ a(2)\}$
i_8	Pare			

2.3 Algoritmo para o cálculo do valor de uma potência

Considerando que as operações aritméticas básicas de um computador são: adição, subtração, multiplicação e divisão, segue abaixo um algoritmo para se obter o valor de uma potência de base x.

Algoritmo Exponenciação

i_1: Entre com os valores de $x \neq 0$ e de n (n uma variável inteira não negativa).
i_2: $i\ := 1$
i_3: $x(i-1)\ := 1$
i_4: **Enquanto** $i \leqslant n$, **faça**
i_5: $x(i)\ := x(i-1)\ *\ x$
i_6: $i\ := i + 1$
i_7: **Se** $n = 0$ **então retorne** $x^n = 1$.
i_8: **Senão retorne:** $x^n = x(i-1)$.
i_9: **Pare.**

Esse algoritmo considera $x \neq 0$ e $n \in \mathbb{N} \cup \{0\}$, iniciando com $i = 1$ e $x(0) = 1$. Depois, para cada valor assumido por i, de 1 até n, tem-se:

$$x(1) = \underbrace{1}_{x(0)} \cdot x \longrightarrow x(2) = \underbrace{x}_{x(1)} \cdot x \longrightarrow x(3) = \underbrace{x^2}_{x(2)} \cdot x$$

$$x(4) = \underbrace{x^3}_{x(3)} \cdot x \longrightarrow \cdots \longrightarrow x(n) = \underbrace{x^{n-1}}_{x(n-1)} \cdot x,$$

sendo $x(n)$ retornado como solução.

Exemplo 20. Aplicando o algoritmo para $x = 5$ e $n = 3$, obtém-se:

i_1:	$x = 5$ e $n = 3$	i_6:	$i = 3$
i_2:	$i = 1$	i_4:	Teste verdadeiro
i_3:	$x(0) = 1$	i_5:	$x(3) = x(2) \cdot 5 = 25 \cdot 5 = 125$
i_4:	Teste verdadeiro	i_6:	$i = 4$
i_5:	$x(1) = x(0) \cdot 5 = 1 \cdot 5 = 5$	i_4:	Teste falso
i_6:	$i = 2$	i_7:	Teste falso
i_4:	Teste verdadeiro	i_8:	$5^3 = 125$
i_5:	$x(2) = x(1) \cdot 5 = 5 \cdot 5 = 25$	i_9:	Pare.

Definição 1. Chama-se piso de um número real x, e denota-se o piso de x por $\lfloor x \rfloor$ o maior inteiro menor do que ou igual a x.

Exemplo 21. Conforme a definição 1, tem-se:

$$\lfloor 7 \rfloor = 7;\ \lfloor 4,23 \rfloor = 4 \text{ e } \lfloor -5,23 \rfloor = -6.$$

Definição 2. Dado um número real x, o menor inteiro maior do que ou igual a x será representado por $\lceil x \rceil$.

Exemplo 22. De acordo com a definição acima, tem-se:

$\lceil 7 \rceil = 7;\ \lceil 4,23 \rceil = 5$ e $\lceil -5,23 \rceil = -5$.

■

Será visto a seguir um outro algoritmo, denominado exponenciação rápida, para o cálculo de potências. Ele é baseado em sucessivas divisões de expoente pelo número 2, conforme as igualdades abaixo, para o cálculo de x^n.

$$\begin{aligned}
x^n &= x^{2q_0+r_0} \\
&= (x \cdot x)^{q_0} \cdot x^{r_0} \\
&= (x^2)^{2q_1+r_1} \cdot x^{r_0} \\
&= (x^2 \cdot x^2)^{q_1} \cdot \underline{(x^2)^{r_1}} \cdot \underline{x^{r_0}} \\
&= (x^4)^{2q_2+r_2} \cdot \underline{(x^2)^{r_1}} \cdot x^{r_0} \\
&= (x^4 \cdot x^4)^{q_2} \cdot \underline{(x^4)^{r_2}} \cdot \underline{(x^2)^{r_1} \cdot x^{r_0}} \\
&\vdots \\
&= \left(x^{2^{k-2}}\right)^{2 \cdot 1 + r_{k-2}} \cdot \left(x^{2^{k-3}}\right)^{r_{k-3}} \cdot \left(x^{2^{k-4}}\right)^{r_{k-4}} \cdot \ldots \cdot (x^2)^{r_1} \cdot x^{r_0} \\
&= \left(x^{2^{k-2}} \cdot x^{2^{k-2}}\right)^1 \cdot \underline{\left(x^{2^{k-2}}\right)^{r_{k-2}}} \cdot \underline{\left(x^{k-3}\right)^{r_{k-3}} \cdot \ldots \cdot (x^2)^{r_1} \cdot x^{r_0}} \\
&= \left(\underline{x^{2^{k-1}}}\right)^{2 \cdot 0 + 1} \cdot \underline{\left(x^{2^{k-2}}\right)^{r_{k-2}} \cdot \left(x^{2^{k-3}}\right)^{r_{k-3}} \cdot \ldots \cdot (x^2)^{r_1} \cdot x^{r_0}}
\end{aligned}$$

Mas, esse procedimento faz com que n seja escrito na forma binária, ou seja, $n = r_{k-1}r_{k-2}...r_1r_0$, sendo $r_{k-1} = 1$, $r_j \in \{0,1\}$, $\forall j \in \{0,1,2,...,(k-2)\}$ e $k-1$ o número de restos produzidos na divisão por 2.

Assim, tem-se que:

$$2^{k-1} \leqslant \underbrace{2^{k-1} + r_{k-2}2^{k-2} + ... + r_1 2 + r_0}_{=n} \leqslant$$

$$\leqslant \underbrace{2^{k-1} \ + \ 2^{k-2} \ + \ ... \ + \ 2 \ + \ 1}_{Soma\ dos\ k\ termos\ da\ PG\ de\ razão\ 2} = 1 \cdot \tfrac{2^k-1}{2-1} = 2^k - 1 < 2^k.$$

Portanto, $2^{k-1} \leqslant n < 2^k$ e

$$2^{k-1} \leqslant n < 2^k \Leftrightarrow log_2 2^{k-1} \leqslant log_2 n < log_2 2^k \Leftrightarrow$$

$$k - 1 \leqslant log_2 n < k \Leftrightarrow k - 1 = \lfloor log_2 n \rfloor \Leftrightarrow$$

$$k = 1 + \lfloor log_2 n \rfloor,$$

sendo $\lfloor log_2 n \rfloor$ o maior inteiro menor do que ou igual a $log_2 n$.

Para esse algoritmo, tem-se:

- Total de divisões $= k$.
- Total de multiplicações $= (k-1) + (k-1) = 2k - 2$.
- Total de operações $= 3k - 2$.

Há portanto $3k - 2 = 3\,(1 + \lfloor log_2 n \rfloor) - 2 \leqslant 3\,(1 + log_2 n) - 2 = 1 + 3\ log_2 n = g(n)$ multiplicações e divisões, no total.

Algoritmo Exponenciação Rápida

i_1: Entre com os valores de x e n, considerando $resp\ := 1$.
i_2: Divida n por 2 para obter o quociente q e o resto r.
i_3: Se $r = 1$, considere $resp\ := resp * x$.
i_4: Se $q = 0$, então pare.
i_5: $n\ := q$.
i_6: $x\ := x * x$.
i_7: Vá para i_2.

Exemplo 23. A seguir é executado o teste de mesa para o cálculo de x^{25}.

Instrução Executada	*Variáveis*				
	x	n	$resp$	q	r
i_1	x	25	1	-	-
i_2	x	25	1	12	1
i_3	x	25	x	12	1
i_5	x	12	x	12	1
i_6	x^2	12	x	12	1

Instrução Executada	*Variáveis*				
	x	n	$resp$	q	r
i_2	x^2	12	x	6	0
i_5	x^2	6	x	6	0
i_6	x^4	6	x	6	0
i_2	x^4	6	x	3	0
i_5	x^4	3	x	3	0
i_6	x^8	3	x	3	0
i_2	x^8	3	x	1	1
i_3	x^8	3	x^9	1	1
i_5	x^8	1	x^9	1	1
i_6	x^{16}	1	x^9	1	1
i_2	x^{16}	1	x^9	0	1
i_3	x^{16}	1	x^{25}	0	1
i_4	Pare				

O algoritmo exponenciação requer n multiplicações para computar x^n. Indica-se por $f(n)$ (função de n) esta quantia. Neste caso, $f(n) = n$. Tal algoritmo é portanto chamado de linear.

O algoritmo exponenciação rápida requer não mais do que $1 + 3\ log_2 n$ multiplicações e divisões para computar x^n. Indica-se essa quantia pela função $g(n) = 1 + 3log_2 n$. Este algoritmo é chamado de logarítmico.

Exemplo 24. No caso $n = 4$, tem-se $f(4) = 4$ e $g(4) = 1 + 3 \times log_2 4 = 1 + 3 \times log_2 2^2 = 1 + 3 \times 2 = 7$. Agora para $n = 2^{11} = 2048$, tem-se: $f(n) = 2048$ e $g(n) = 1 + 3 \cdot log_2 2^{11} = 1 + 3 \times 11 = 34$.

O Lema a seguir fornece uma propriedade necessária para a demostração da próxima proposição, a qual compara os algoritmos exponenciação e exponenciação rápida.

Proposição 2. Para todo natural $n \geqslant 12$, tem-se que $1 + 3 \cdot log_2 n < n$.

Demonstração:

Basta provar, usando o Princípio de Indução, que para $n \geqslant 12$ tem-se $n^3 < 2^{n-1}$. De fato,

$$1 + 3 \cdot log_2 n < n \Leftrightarrow log_2 n < \frac{n-1}{3} \Leftrightarrow n < 2^{\frac{n-1}{3}} \Leftrightarrow n^3 < 2^{n-1}.$$

Para $n = 12$, obtém-se $n^3 = 12^3 = 1728$ e $2^{n-1} = 2^{11} = 2048$, portanto a desigualdade é verdadeira.

Suponha a desigualdade ser verdadeira para um n, isto é, $n^3 < 2^{n-1}$, na qual $n \geqslant 12$ (hipótese de indução). Será provado que também é verdadeira para $n + 1$, ou seja, que $(n + 1)^3 < 2^n$.

Usando a hipótese de indução, tem-se que:

$$(n + 1)^3 = n^3 + 3n^2 + 3n + 1 < 2^{n-1} + 3n^2 + 3n + 1.$$

Além disso, tem-se que:

$$2^{n-1} + 3n^2 + 3n + 1 < 2^{n-1} + 3n^2 + 3n^2 + 3n^2 = 2^{n-1} + 9n^2 \ .$$

Como $n \geqslant 12$, segue que $9n^2 < n^3 < 2^{n-1}$, logo

$$(n+1)^3 < 2^{n-1} + 9n^2 < 2^{n-1} + n^3 < 2^{n-1} + 2^{n-1} = 2^n.$$

Portanto, $(n+1)^3 < 2^n$.

■

A comparação de dois ou mais algoritmos que solucionarão o mesmo problema, é fundamental para a escolha do mais eficiente.

A eficiência de um algoritmo pode ser medida, por exemplo, em termos do tempo de execução ou do espaço (ou memória) usado. O custo de um algoritmo, dado por algum desses tipos de medição, é também chamado de complexidade.

Existe complexidade de tempo, de uso de memória, de comunicação, do número de processadores, entre outras. A complexidade de um algoritmo consiste na quantidade de "trabalho" necessário para a sua execução. Este estudo (Complexidade de Algoritmos) é uma área ampla da teoria dos algoritmos.

A contagem de passos na execução de um algoritmo fornece uma caracterização simplificada da eficiência de um algoritmo. Neste trabalho, deseja-se apenas dar uma pequena ideia sobre custo de um algoritmo.

Em geral, considera-se que cada execução da linha i possui um custo $c_i > 0$, sendo c_i a quantidade de passos executados implicitamente, pelo computador. Por exemplo, o custo da adição $11 + 14$, escritos no sistema decimal, não será apenas duas adições, $1+4$ (unidades) e $1+1$ (dezenas) pois, em termos computacionais trabalha-se com a expansão binária. Assim, 11 é interpretado pelo computador como 1011 e o 14 como 1110 e será realizada a soma desses números binários. Então o custo, número total de adições, é dado pela expressão

$$\begin{array}{cccccc} & 1 & 1 & 1 & & \\ & 0 & 1 & 0 & 1 & 1 \\ + & 0 & 1 & 1 & 1 & 0 \\ \hline & 1 & 1 & 0 & 0 & 1 \end{array},$$

na qual ocorre mais adições do que no caso anterior.

Fazendo a contagem levando em consideração os custos de cada linha do pseudo-código, obtém-se a função $f : \begin{array}{ccc} \mathbb{N} & \longrightarrow & \mathbb{R}^+ \\ n & \longmapsto & f(n) \end{array}$, na qual $f(n)$ é o custo da execução do algoritmo, para a entrada n. A Proposição 2 mostra que o algoritmo exponenciação rápida é mais eficiente do que o algoritmo exponenciação.

Se a função de custo for um polinômio, por exemplo, $f(n) = an^2 + bn + c$, sendo a, b e c constantes positivas, dadas em função de c_i, então $f(n) = an^2\left(1 + \frac{b}{an} + \frac{c}{an^2}\right)$, de modo que, para $n \mapsto \infty$, tem-se $\left(1 + \frac{b}{an} + \frac{c}{an^2}\right)$ tendendo a 1. Isso quer dizer que, a partir de n suficientemente grande, a função f cresce com n^2, podendo $f(n) = an^2$ ser considerada como

a função custo, pois os termos inferiores, de menor crescimento na função, e as constantes multiplicativas contribuem pouco na comparação, podendo ser descartados.

Quando se observa o tamanho de entradas grandes o suficiente, para tornar relevante apenas a ordem de crescimento do tempo de execução, está sendo estudada a eficiência assintótica do algoritmo. Assim, deseja-se saber como o tempo de execução de um algoritmo aumenta com o tamanho da entrada aumentando indefinidamente (sem limitação). Em geral, um algoritmo que é assintóticamente mais eficiente será a melhor escolha para todas as entradas, exceto talvez para as muito pequenas.

2.4 Notação Assintótica O

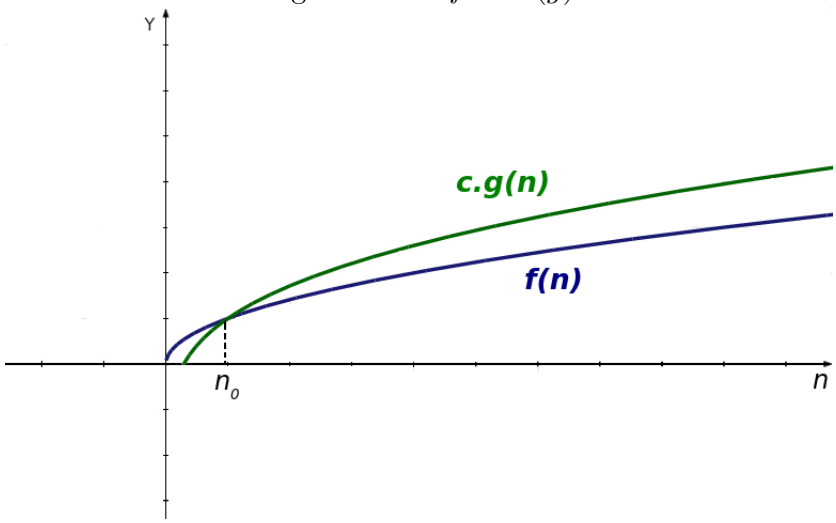
Definição 3. Dadas as funções $f : \mathbb{N} \underset{n \longmapsto f(n)}{\longrightarrow} \mathbb{R}^+$ e $g : \mathbb{N} \underset{n \longmapsto g(n)}{\longrightarrow} \mathbb{R}^+$, diz-se que $f(n)$ é $O(g(n))$ se existem uma constante positiva c e $n_0 \in \mathbb{N}$ tais que $0 < f(n) \leqslant c.g(n)$, para todo $n \geqslant n_0$. Neste caso, usa-se a notação $f = O(g)$.

Em outras palavras, dada uma função $g(n)$, indicando por $O(g(n))$ o conjunto das funções

$$\{f(n) \mid \text{existem constantes } c > 0 \text{ e } n_0 \in \mathbb{N} \text{ tais que } 0 \leqslant f(n) \leqslant c \cdot g(n),\ \forall n \geqslant n_0\},$$

$f = O(g)$ significa que $f(n) \in O(g(n))$.

Figura 2.4.1: $f = O(g)$



Tal notação é utilizada para comparar algoritmos e é considerada como uma cota superior para o número de passos requeridos, significando que o custo do algoritmo não ultrapassa $O(g(n))$, para entradas suficientemente grandes. Mas a cota superior de um algoritmo pode mudar se alguém descobrir um outro algoritmo melhor, como aconteceu

no caso do cálculo do valor de x^n, $x \neq 0$. Antes, com o algoritmo exponenciação, o custo não ultrapassava a cota superior $O(n)$, agora com o exponenciação rápida este custo não ultrapassa a cota superior $O(log_2 n)$. Ou seja, a cota superior do problema do cálculo de x^n passou a ser $O(log_2 n)$.

A cota superior para o custo de um problema é análoga ao record mundial de uma modalidade de atletismo. Ele é estabelecido pelo melhor atleta (algoritmo) do momento. Assim como o record mundial, a cota superior pode ser melhorada por um algoritmo (atleta) mais veloz, [19].

Exemplo 25. Seja $f(n) = a_d n^d + a_{d-1} n^{d-1} + ... + a_1 n + a_0$, $d \in \mathbb{N}$, a_i uma constante real, $\forall i \in \{0, 1, 2, ..., d\}$. Então, $f(n) = O(n^d)$.

Tem-se:

$$f(n) = a_d n^d + a_{n-1} n^{d-1} + ... + a_1 n + a_0 \leqslant$$

$$\leqslant |a_d| n^d + |a_{n-1}| n^{d-1} + ... + |a_1| n + |a_0| \leqslant$$

$$\leqslant |a_d| n^d + |a_{n-1}| n^d + ... + |a_1| n^d + |a_0| n^d =$$

$$= (\underbrace{|a_d| + |a_{d-1}| + ... + |a_0|}_{>0}) n^d = cn^d$$

Portanto, $f(n) = O(n^d)$.

Observação 2. $|a_d|$ significa módulo ou valor absoluto de a_d.

Assim, $O(n^2 + 2n - 1) = O(n^2)$ e $O(5n^4 - 3n^2 + n + 2) = O(n^4)$.

Lema 1. Se r for um número inteiro maior do que 5, então $2^r > (r+1)^2$.

Demonstração:

Tal lema será provado por indução finita sobre r.

Observe que o lema será falso para $r = 1,\ 2,\ 3,\ 4,\ 5$, na tabela abaixo.

r	2^r	$(r+1)^2$
1	2	4
2	4	9
3	8	16

r	2^r	$(r+1)^2$
4	16	25
5	32	36
6	64	49

Tem-se que $P(6)$ é verdadeira.

Supondo que $P(r)$ seja verdadeira para algum $r \geqslant 6$, será provado que $P(r+1)$ também é verdadeira, ou seja, que $2^{r+1} > (r+2)^2$. Tem-se que

$$\begin{aligned} 2^{r+1} &= 2 \,.\, 2^r \\ &> 2(r+1)^2 \qquad \text{(hipótese de indução)} \\ &= 2r^2 + 4r + 2 \\ &= r^2 + 4r + 4 + (r^2 - 2) \\ &= (r+2)^2 + (r^2 - 2) \qquad (1) \end{aligned}$$

Como $r \geqslant 6$, obtém-se $r^2 - 2 > 0$ (2).
De (1) e (2), segue que:

$$2^{r+1} > (r+2)^2 + (r^2 - 2) > (r+2)^2.$$

Logo, $2^{r+1} > (r+2)^2$, isto é, $P(r+1)$ é verdadeira.

Exemplo 26. $log_2 n = O(\sqrt{n})$.

De fato, suponha que k seja o maior inteiro tal que $2^k \leqslant n$, ou seja, $k = \lfloor log_2 n \rfloor$. Tem-se:

$$2^{k+1} > n \geqslant 2^k \Leftrightarrow 2^{k+1} > n \text{ e } n \geqslant 2^k \Leftrightarrow$$

$$k + 1 > log_2 n \text{ e } n \geqslant 2^k. \qquad (1)$$

Seja $n \geqslant 64 = 2^6$, tem-se, do Lema 1 e de (1), que:

$$k + 1 > log_2 n \text{ e } n \geqslant 2^k > (k+1)^2,$$

$$k + 1 > log_2 n \text{ e } \sqrt{n} > k + 1,$$

$$\sqrt{n} > k + 1 > log_2 n,$$

$$\sqrt{n} > log_2 n.$$

2.4.1 Propriedades da Notação O

(i) A notação O é reflexiva, ou seja, se $f(n)$ é a função de custo então $f(n) = O(f(n))$.

(ii) A notação O é transitiva, ou seja, se $f = O(g)$ e $g = O(h)$ então $f = O(h)$.

(iii) Se $f = O(g)$ então $a.f = O(g),\ \forall a > 0$.

(iv) Se $f = O(g_1)$ e $h = O(g_2)$ então $f + h = O(g_1) + O(g_2) = O\left(max\{g_1,\ g_2\}\right)$ e $f.h = O(g_1) \cdot O(g_2) = O(g_1.g_2)$.

De fato,

(i) $f(n) \leqslant 1 \cdot f(n),\ \forall n \geqslant 1$.

(ii) Sendo $f = O(g)$ e $g = O(h)$ então existirão $c_1,\ c_2 > 0$ e $n_1,\ n_2 \in \mathbb{N}$ tais que:

$$f(n) \leqslant c_1 g(n),\ \forall n \geqslant n_1 \text{ e } g(n) \leqslant c_2 h(n),\ \forall n \geqslant n_2$$

Tomando $n_0 = max\{n_1,\ n_2\}$, tem-se:

$$f(n) \leqslant c_1 g(n) \leqslant c_1 c_2 h(n),\ \forall n \geqslant n_0.$$

(iii) Existem $c > 0$ e $n_0 \in \mathbb{N}$ tais que

$$f(n) \leqslant cg(n),\ \forall n \geqslant n_0.$$

Como $a > 0$ então $af(n) \leqslant (ac)g(n),\ \forall n \geqslant n_0$.

(iv) Seja $g = max\{g_1,\ g_2\}$. Então $g(n) \geqslant g_1(n)$ e $g(n) \geqslant g_2(n)$, para toda entrada n.

Por hipótese, existem $c_1,\ c_2 > 0$ e $n_1,\ n_2 \in \mathbb{N}$ tais que $f \leqslant c_1 \cdot g_1$, para todo $n \geqslant n_1$, e $f \leqslant c_2 \cdot g_2$, para todo $n \geqslant n_2$. Tomando $n_0 = max\{n_1,\ n_2\}$, tem-se

$$f + g \leqslant c_1 \cdot g_1 + c_2 \cdot g_2 \leqslant c_1 \cdot g + c_2 \cdot g = (c_1 + c_2)g, \text{ para todo } n \geqslant n_0.$$

Retomando a hipótese, tem-se:

$$0 \leqslant f(n) \leqslant c_1 g_1(n) \text{ e } 0 \leqslant h(n) \leqslant c_2 g_2(n) \Rightarrow f(n).h(n) \leqslant c_1 c_2 g_1(n).g_2(n) \Rightarrow$$

$$f.h(n) \leqslant c_1 c_2 (g_1.g_2)(n),\ (c_1 c_2 > 0).$$

Portanto, $f.h = O(g_1.g_2)$.

■

Observe que a notação O não é simétrica. De fato, $n = O(n^2)$ e $n^2 \neq O(n)$.

Exemplo 27. Suponha que $f(n) = log_2 n\ .\ \sqrt{n^3}$. Então, $f(n) = (log_2 n) \cdot (n \cdot \sqrt{n})$. Como $log_2 n = O(\sqrt{n})$, tem-se:

$$f(n) = O(\sqrt{n}) \cdot O(n \cdot \sqrt{n}) = O(\sqrt{n} \cdot n \cdot \sqrt{n}) = O(n^2).$$

Proposição 3. Para todo $p \in \mathbb{N}$ tem-se que:

$$1^p + 2^p + 3^p + ... + n^p = \sum_{k=1}^{n} k^p \text{ é um polinômio de grau } p+1 \text{ em } n.$$

Demonstração:

Por indução finita sobre p, tem-se:

Para $p = 1$, tem-se $1 + 2 + ... + n = \frac{n(n+1)}{2}$, que é um polinômio de grau 2 em n.

Suponha a afirmação verdadeira para todo p, $1 \leqslant p \leqslant s$. Será provado que é válida também para $s + 1$.

Usando o Binômio de Newton, tem-se que:

$$(k+1)^{s+2} = k^{s+2} + \begin{pmatrix} s+2 \\ 1 \end{pmatrix} k^{s+1} + \begin{pmatrix} s+2 \\ 2 \end{pmatrix} k^s + ... + \begin{pmatrix} s+2 \\ s+1 \end{pmatrix} k + 1.$$

Fazendo k variar de 1 até n e somando as equações obtidas, obtém-se:

$$\sum_{k=1}^{n} (k+1)^{s+2} = \sum_{k=1}^{n} k^{s+2} + \sum_{k=1}^{n} \begin{pmatrix} s+2 \\ 1 \end{pmatrix} k^{s+1} + \begin{pmatrix} s+2 \\ 2 \end{pmatrix} \sum_{k=1}^{n} k^s + ... + \begin{pmatrix} s+2 \\ s+1 \end{pmatrix} \sum_{k=1}^{n} k + n.$$

Usando a hipótese de indução obtém-se:

$$\sum_{k=1}^{n}(k+1)^{s+2} = \sum_{k=1}^{n}k^{s+2} + \sum_{k=1}^{n}(s+2)k^{s+1} + F(n),$$

sendo $F(n)$ um polinômio de grau $s+1$ em n.

Como $\sum_{k=1}^{n}(k+1)^{s+2} = \sum_{k=2}^{n+1}k^{s+2}$, obtém-se:

$$\sum_{k=2}^{n+1}k^{s+2} = \sum_{k=1}^{n}k^{s+2} + \sum_{k=1}^{n}(s+2)k^{s+1} + F(n) \text{ e simplificando,}$$

$$(n+1)^{s+2} = 1 + (s+2)\sum_{k=1}^{n}k^{s+1} + F(n), \text{ implicando em}$$

$$\sum_{k=1}^{n}k^{s+1} = \frac{(n+1)^{s+2} - F(n) - 1}{s+2},$$

que é um polinômio de grau $s+2$ em n.

Corolário 2. Se F é um polinômio de grau p, então $\sum_{k=1}^{n}F(k)$ é um polinômio de grau $p+1$, em n.

Demonstração:

Seja $F(x) = a_0 + a_1x + ... + a_px^p$, com $a_p \neq 0$. Tem-se que:

$$\sum_{k=1}^{n}F(k) = \sum_{k=1}^{n}a_0 + a_1k + ... + a_pk^p = \sum_{k=1}^{n}a_0 + a_1\sum_{k=1}^{n}k + ... + a_p\sum_{k=1}^{n}k^p,$$

que é um polinômio de grau $p+1$ em n.

Capítulo 3

Algoritmos na Matemática Básica

Agora que já se sabe o que é um algoritmo e como obter o seu custo, serão estudados alguns algoritmos da matemática básica e, para alguns deles, será efetuada a análise de custo.

3.1 Divisão Euclidiana

Teorema 2. (*Divisão Euclidiana*)
Se $a,\ b \in \mathbb{N}$, com $b > 0$, então existem dois únicos naturais q e r tais que

$$a = b \cdot q + r, \text{ com } 0 \leqslant r < b.$$

Demonstração:
Existência:
Se $a < b$, existem $q = 0$ e $r = a$ nas condições exigidas, assim pode-se assumir que $a \geqslant b > 0$.

Considere o conjunto $S = \{a - by \mid y \in \mathbb{N}\} \cap \{\mathbb{N} \cup \{0\}\}$.

Como $a - b \geqslant 0$, tem-se que $a - b \in S$, logo o conjunto S é não vazio. Como S é limitado inferiormente, tem-se que S possui um menor elemento $r = a - bq \geqslant 0$. Será mostrado que $r < b$.

Suponha por absurdo que $r \geqslant b$. Segue que $r = s + b$ com $s \in \mathbb{N} \cup \{0\}$, logo $s = r - b = a - bq - b = a - (q + 1)b \in S$. Como r é o menor elemento de S, chega-se a uma contradição, já que $r = s + b \Rightarrow s < r$ e $s \in S$.

Unicidade:
Suponha $a = bq + r = bq' + r'$, sendo $0 \leqslant r$ e $r' < b$. Sem perda de generalidade, suponha $r \leqslant r'$. Daí tem-se que $0 \leqslant b(q - q') = r' - r < b$, o que só é possível se $q = q'$ e consequentemente $r = r'$.

Nas condições do Teorema os números q e r são chamados, respectivamente, de quociente e resto da divisão de a por b.

O tipo de demonstração do Teorema da Divisão Euclidiana é chamada de demonstração algorítmica porque a prova da existência fornece uma receita para o cálculo do resto e do quociente da divisão de a por b.

Exemplo 28. Determinação de q e r da divisão de 17 por 3, usando o procedimento da demonstração do Teorema.

Tem-se que $S = \{17-3\cdot1,\ 17-3\cdot2,\ 17-3\cdot3,\ 17-3\cdot4,\ 17-3\cdot5\} = \{14, 11, 8, 5, 2\}$, portanto $r = 2$ e $q = 5$.

Pode-se assim escrever um algoritmo para calcular o quociente e o resto da divisão de a por b.

Algoritmo da Divisão

- Entrada: Números naturais a e b.
- Saída: Números naturais q e r tais que $a = bq + r$, com $0 \leqslant r < b$.
- Passo 1: Comece fazendo $Q = 0$ e $R = a$.
- Passo 2: Se $R < b$ retorne a frase "O quociente é Q e o resto é R" e pare. Caso contrário, vá para o passo 3.
- Passo 3: Se $R \geqslant b$ faça $\underbrace{R}_{\text{Novo } R} = \underbrace{R}_{\text{Velho } R} - b$, incremente Q de 1 e volte para o passo 2.

Ao final da execução do algoritmo, Q vai ser o quociente e R o valor do resto.

Para determinar estes valores deve-se executar os passos 2 e 3 várias vezes. Chama-se isso de laço (*loop*). Ao final de cada laço, as variáveis Q e R terão valores diferentes.

A mudança de valor é efetuada no passo 3. A instrução: "subtraia b de R" significa: à variável R será atribuido um novo valor que corresponde ao valor que tinha no final do passo anterior, menos b. De modo análogo, incremente Q de 1 significa: à variável Q será atribuido o valor que tinha no passo anterior, mais 1.

Por exemplo, se $a > b$, então passado uma vez pelo passo 3, obtém-se $Q = 1$ e $R = a-b$. Se $a-b \geqslant b$ então aplicando novamente o passo 3 tem-se $Q = 2$ e $R = a-2b$. E assim por diante.

Observe que, aplicando o passo 3 várias vezes será gerada a seguinte sequência de valores para R:

$$a,\ a-b,\ a-2b,\ a-3b,\ a-4b\cdots$$

Trata-se de uma sequência decrescente de números naturais. Como existe apenas um número finito de naturais entre a e 0, a sequência vai chegar a um valor menor do que b. Neste momento, o algoritmo para de modo a ter os valores procurados. De fato, se r e q são obtidos pelo algoritmo então tem-se $r = a-bq$ e $r < b$. Assim, $a = bq + r$ e $r < b$, faltando provar que $r \geqslant 0$. Como o laço parou quando $r = a-bq$, no passo anterior tem-se $a-(q-1)b \geqslant b$. Subtraindo b desta última desigualdade obtém-se $a-qb \geqslant 0$. Portanto o algoritmo está correto.

É fácil perceber que o algoritmo apresentado é lento. O número de laços executados é igual ao quociente. Por exemplo, ao ser dividido b^{k+1} por b, o passo 3 será executado b^k vezes e o crescimento será exponencial.

Observação 3. Certamente ao se dividir $a = 5712$ por $b = 75$, usando lápis e papel, usa-se outro método: em primeiro lugar escolhe-se o menor número formado pelos algarismos 5712, começando pela esquerda, que é maior do que ou igual a 75, no caso 571. Em seguida, é dividido 571 por 75, encontrando o quociente 7 e o resto 46 e depois é dividido 462 por 75,

obtendo o quociente 6 e resto 12. O resultado final será $q = 76$ e $r = 12$. Ao dividir 571 por 75, uma pergunta é feita: Qual o maior número que pode ser multiplicado por 75 e ainda assim obter um produto menor do que ou igual a 571? Se os números são pequenos, o valor correto é obtido rapidamente. Se os números são grandes, a quantidade de possibidades pode tornar impossível resolver o problema.

O processo da divisão é feito por passos, cada um consistindo na divisão de um número $u = u_0u_1...u_n$ por outro número $v = v_1v_2...v_n$, tal que $\frac{u}{v} < 10$. Depois de cada passo usa-se a quantidade $10r + (a\ outra\ parte\ do\ dividendo)$ como sendo o novo u.

$$\begin{array}{cccc|cc} 5 & 7 & 1 & 2 & 7 & 5 \\ \cline{5-6} & 4 & 6 & 2 & 7 & 6 \\ & & 1 & 2 & & \end{array}$$

A seguir será estudado um algoritmo que resolverá este problema. Trabalha-se usando o sistema decimal, mas poderia ser feito de modo inteiramente análogo num sistema qualquer de base $b > 1$.

Considere dois números escritos na base decimal, u, com $n+1$ posições, e v, com n algarismos, tais que $u < 10v$, sendo

$$u = u_0u_1...u_n = u_0 \cdot 10^n + u_1 \cdot 10^{n-1} + \cdots + u_n \text{ e}$$

$$v = v_1v_2 \cdots v_n = v_1 \cdot 10^{n-1} + v_2 \cdot 10^{n-2} + \cdots + v_n.$$

A condição $u < 10v$ significa que a parte inteira de $\frac{u}{v}$, representada por $\left\lfloor \frac{u}{v} \right\rfloor$ é menor do que 10, ou ainda que

$$u_0u_1...u_{n-1} < v_1v_2...v_n.$$

Quando se dividiu 5712 por 75, a primeira etapa foi a divisão de 571 por 75. Neste caso $u = 571$, $v = 75$ e $\left\lfloor \frac{571}{75} \right\rfloor < 10$.

Se $u = v \cdot q + r$, com $0 \leqslant r < v$, então deseja-se determinar $q = \left\lfloor \frac{u}{v} \right\rfloor$, o quociente da divisão euclidiana de u por v. Uma aproximação do valor de q será determinada pelos algarismos u_0 e u_1, de u, e do algarismo v_1, de v.

Proposição 4. Sejam $u = u_0u_1...u_n$ e $v = v_1v_2...v_n$ tais que $u < 10v$. Se q é o quociente da divisão de u por v e $\widetilde{q} := min\left(\left\lfloor \frac{u_0 \cdot 10 + u_1}{v_1} \right\rfloor,\ 9\right)$, então $\widetilde{q} \geqslant q$.

Demonstração:

Como $q \leqslant 9$, a proposição é verdadeira quando $\widetilde{q} = 9$.

Suponha que $\tilde{q} = min\left(\left\lfloor \frac{u_0 \cdot 10 + u_1}{v_1} \right\rfloor,\ 9\right)$. Segue que $\tilde{q} > \frac{u_0 \cdot 10 + u_1}{v_1} - 1$, e daí

$$\widetilde{q} \cdot v_1 > u_0 \cdot 10 + u_1 - v_1.$$

Como os algarismos são números inteiros, conclui-se que

$$\widetilde{q} \cdot v_1 \geqslant u_0 \cdot 10 + u_1 - v_1 + 1.$$

Calculando $u - \widetilde{q}v$ e usando a desigualdade anterior, tem-se que
$u - \widetilde{q}v = u_0 \cdot 10^n + u_1 \cdot 10^{n-1} + ... + u_n - \widetilde{q}(v_1 \cdot 10^{n-1} + v_2 \cdot 10^{n-2} + ... + v_n)$
$u - \widetilde{q}v \leqslant u_0 \cdot 10^n + u_1 \cdot 10^{n-1} + ... + u_n - \widetilde{q}(v_1 \cdot 10^{n-1})$
$u - \widetilde{q}v \leqslant u_0 \cdot 10^n + u_1 \cdot 10^{n-1} + ... + u_n - (u_0 \cdot 10 + u_1 - v_1 + 1)10^{n-1}$,
sendo $u_0 \cdot 10^n + u_1 \cdot 10^{n-1} + ... + u_n - (u_0 \cdot 10 + u_1 - v_1 + 1)10^{n-1} = u_2 \cdot 10^{n-2} + ... + u_n - 10^{n-1} + v_1 \cdot 10^{n-1}$.

Portanto, $u - \widetilde{q}v \leqslant u_2 \cdot 10^{n-2} + ... + u_n - 10^{n-1} + v_1 \cdot 10^{n-1} < v_1 \cdot 10^{n-1} \leqslant v$.
Obteve-se $u - \widetilde{q}v < v$. Portanto, $\widetilde{q} \geqslant q$.

Proposição 5. Sejam $u = u_0u_1...u_n$ e $v = v_1v_2...v_n$ tais que $u < 10v$, q o quociente da divisão de u por v e $\widetilde{q} := min\left(\left\lfloor \frac{u_0 \cdot 10 + u_1}{v_1} \right\rfloor,\ 9\right)$. Se $v_1 \geqslant 5$ então $q = \widetilde{q},\ \widetilde{q} - 1$ ou $\widetilde{q} - 2$.

Demostração:

Suponha $q \leqslant \widetilde{q} - 3$.
Tem-se então que $\widetilde{q} \leqslant \frac{u_0 \cdot 10 + u_1}{v_1} = \frac{u_0 \cdot 10^n + u_1 \cdot 10^{n-1}}{v_1 \cdot 10^{n-1}} \leqslant \frac{u}{v_1 \cdot 10^{n-1}}$.
O caso $v = 10^{n-1}$ implica $u = u_0 \cdot 10^n + u_1 \cdot 10^{n-1} + ... + u_n \geqslant 10^n = 10 \cdot v$, o que não é possível.

Segue que $v - 10^{n-1} = v_1 \cdot 10^{n-1} + v_2 \cdot 10^{n-2} + ... + v_n - 10^{n-1} < v_1 \cdot 10^{n-1}$ e assim

$$\widetilde{q} \leqslant \frac{u}{v_1 \cdot 10^{n-1}} < \frac{u}{v - 10^{n-1}}.$$

Tem-se que $q + 1 = \left\lfloor \frac{u}{v} \right\rfloor + 1 > \frac{u}{v}$ e $\widetilde{q} - 3 \geqslant q > \frac{u}{v} - 1$.
Dessa forma,

$$3 \leqslant \widetilde{q} - q < \widetilde{q} + 1 - \frac{u}{v} < \frac{u}{v - 10^{n-1}} + 1 - \frac{u}{v} =$$

$$= \frac{u \cdot v}{v \cdot (v - 10^{n-1})} + 1 - \frac{u}{v} = \frac{u}{v}\left(\frac{10^{n-1}}{v - 10^{n-1}}\right) + 1.$$

Daí, observando que $\frac{v - 10^{n-1}}{10^{n-1}} \geqslant v_1 - 1$, tem-se:

$$\frac{u}{v} \geqslant 2 \cdot \left(\frac{v - 10^{n-1}}{10^{n-1}}\right) \geqslant 2(v_1 - 1).$$

Finalmente,

$$6 \geqslant \widetilde{q} - 3 \geqslant q = \left\lfloor \frac{u}{v} \right\rfloor \geqslant 2(v_1 - 1) = 2v_1 - 2$$

e assim obtém-se $v_1 \leqslant 4$.

Conclui-se portanto que se $v_1 \geqslant 5$ então $q \geqslant \widetilde{q} - 2$.

Observação 4. A condição $v_1 \geqslant 5$ não será uma restrição para se obter o quociente, porque se forem multiplicados u e v por um mesmo inteiro positivo, o quociente permanece o mesmo:

$$u = vq + r,\ 0 \leqslant r < v \Leftrightarrow ku = (kv) \cdot q + kr,\ 0 \leqslant kr < kv.$$

Um modo de se obter v_1, suficientemente grande, é fazendo a multiplicação de u e v por $\left\lfloor \frac{10}{v_1+1} \right\rfloor$.

De fato, como $1 \leqslant v_1 < 10$ será mostrado que $v_1 \cdot \left\lfloor \frac{10}{v_1+1} \right\rfloor \geqslant 5$.

Tem-se que $v_1 \cdot \left\lfloor \frac{10}{v_1+1} \right\rfloor < (v_1+1) \cdot \left\lfloor \frac{10}{v_1+1} \right\rfloor$, com $(v_1+1) \cdot \left\lfloor \frac{10}{v_1+1} \right\rfloor \leqslant (v_1+1) \cdot \frac{10}{v_1+1} = 10$.

Se $v_1 \geqslant 5$ então $v_1 \left\lfloor \frac{10}{v_1+1} \right\rfloor \geqslant v_1 \geqslant 5$.

Suponha agora que $1 \leqslant v_1 < 5$.

Tem-se que $v_1 \left\lfloor \frac{10}{v_1+1} \right\rfloor > v_1 \left(\frac{10}{v_1+1} - 1 \right)$. Observando que $v_1 \left(\frac{10}{v_1+1} - 1 \right) - 4 = \frac{-v_1^2+5v_1-4}{v_1+1} \geqslant 0$, para $1 \leqslant v_1 \leqslant 4$, conclui-se que $v_1 \left\lfloor \frac{10}{v_1+1} \right\rfloor > v_1 \left(\frac{10}{v_1+1} - 1 \right) \geqslant 4$.

Portanto, $v_1 \left\lfloor \frac{10}{v_1+1} \right\rfloor \geqslant 5$.

■

Com base nos resultados apresentados acima, segue abaixo um outro algoritmo da divisão.

Algoritmo da Divisão

♦ Entrada: Números naturais $u = u_1u_2...u_{m+n}$ e $v = v_1v_2...v_n$, com $n > 1$.
♦ Saída: Números naturais q e r tais que $u = vq + r$, com $0 \leqslant r < v$.
♦ Passo 1 (normalização): Comece fazendo $d = \left\lfloor \frac{10}{v_1+1} \right\rfloor$ e troque u e v por du e dv, respectivamente. Os novos números serão representados por $u = u_0u_1u_2...u_{m+n}$ e $v = v_1v_2...v_n$.

Exemplo 29. $u = 51247$ e $v = 32$, sendo $v_1 = 3$ e $d = 2$, a partir daí $u = 2 \times 51247 = 102494$ e $v = 64$.

Exemplo 30. $u = 51247$ e $v = 61$, sendo $v_1 = 6$ e portanto $d = 1$, a partir daí $u = 1 \times 51247 = 51247$ e $v = 1 \times 61 = 61$.

♦ Passo 2: Inicie com $j = 0$.

Observação 5. Este laço fará a divisão de $u_ju_{j+1}...u_{j+n}$ por $v_1v_2...v_n$, obtendo um quociente q_j. Começando por $j = 0$, fará a divisão de $u_0u_1...u_n$ por $v_1v_2...v_n$, obtendo quociente q_0.

♦ Passo 3: Se $u_j = v_1$ tome $\widetilde{q} = 9$, caso contrário, $\widetilde{q} = \left\lfloor \frac{u_j 10 + u_{j+1}}{v_1} \right\rfloor$. Em seguida, faça o teste: Se $v_2 \cdot \widetilde{q} > (u_j \cdot 10 + u_{j+1} - \widetilde{q} \cdot v_1) \cdot 10 + u_{j+2}$, diminua 1 de $\widetilde{q}$, obtendo $\widetilde{q} - 1$ como o novo $\widetilde{q}$ e repita o teste

Observação 6. O teste do passo 3 é baseado no seguinte fato:

Se $v_2 \cdot \widetilde{q} > (u_0 \cdot 10 + u_1 - \widetilde{q} \cdot v_1) \cdot 10 + u_2$, então $q < \widetilde{q}$.

De fato, na demostração da Proposição 4, repetindo os mesmos passos e trocando $-\widetilde{q}(v_1 \cdot 10^{n-1})$ por $-\widetilde{q}(v_2 \cdot 10^{n-2})$ obtém-se $u - qv < 0$. Portanto, $\widetilde{q} > q$.

Exemplo 31. $u = 631$ e $v = 67$, $\widetilde{q} = 9$ e depois do teste $\widetilde{q} = 9$.

Exemplo 32. $u = 631$ e $v = 54$, $\widetilde{q} = \left\lfloor \frac{63}{5} \right\rfloor = 12$ e depois do teste $\widetilde{q} = 11$.

Exemplo 33. $u = 631$ e $v = 72$, $\widetilde{q} = \left\lfloor \frac{63}{7} \right\rfloor = 9$ e depois do teste $\widetilde{q} = 9$.

♦ Passo 4: Troque $u_j u_{j+1}...u_{j+n}$ por $(u_j u_{j+1}...u_{j+n}) - \widetilde{q}(v_1 v_2...v_n)$.
♦ Passo 5: Faça $q_j = \widetilde{q}$. Se o resultado do passo 4 for positivo vá para o passo 7, caso contrário vá para o passo 6.
♦ Passo 6: Diminua 1 de $\widetilde{q}$ e adicione $0v_1v_2...v_n$ à $u_j u_{j+1}...u_{j+n}$.
♦ Passo 7: Acrescente 1 à j. Se $j \leqslant n$ volte para o passo 3.
♦ Passo 8: $q = q_0 q_1 ... q_n$ é o quociente desejado e o resto r é obtido dividindo $u_{m+1}...u_{m+n}$ por d.

Exemplo 34. Efetuando o teste de mesa para $u = 67121$ e $v = 46$, tem-se:

E	n	t	r	a	d	a
u_1	u_2	u_3	u_4	u_5	|	$\underline{v_1}$	$\underline{v_2}$
6	7	1	2	1		4	6

$\longrightarrow$ $\underbrace{\begin{cases} Normalização \\ d = \left\lfloor \frac{10}{v_1+1} \right\rfloor = 2 \\ u = d \cdot u = 134242 \\ v = d \cdot v = 92 \end{cases}}_{Passo\ 1}$

Passo 4

	$u\underbrace{0=j}_{Passo\ 2}$	u_1	u_2	u_3	u_4	u_5
	1	3	4	2	4	2
-		9	2			
	u_0	u_1	u_2			
$\underbrace{0 <}_{Passo\ 5}$	0	4	2			

v_1	v_2
9	2

1	4	5	9
$\tilde{q} = \underbrace{\left\lfloor \frac{13}{9} \right\rfloor}_{Passo\ 3} = \underbrace{1 = q_0}_{Passo\ 5}$	$\tilde{q} = \underbrace{\left\lfloor \frac{42}{9} \right\rfloor}_{Passo\ 3} = \underbrace{4 = q_1}_{Passo\ 5}$	$\tilde{q} = \underbrace{\left\lfloor \frac{54}{9} \right\rfloor}_{Passo\ 3} = 6$	$\tilde{q} = \underbrace{\left\lfloor \frac{84}{9} \right\rfloor}_{Passo\ 3} = \underbrace{9 = q_1}_{Passo\ 5}$

$\widetilde{q} = 5 = \underbrace{q_2}_{Passo\ 5}$

Passo 4

	$u\underbrace{1=j}_{Passo\ 7}$	u_2	u_3
	4	2	2
-	3	6	8
	u_1	u_2	u_3
$\underbrace{0 <}_{Passo\ 5}$	0	5	4

Teste (P_3)

$v_2 \cdot \widetilde{q} > (u_j \cdot 10 + u_{j+1} - \widetilde{q} \cdot v_1) \cdot 10 + u_{j+2}$
$2 \cdot 1 \not> (13 - 1 \cdot 9) \cdot 10 + 4 = 44$
$2 \cdot 4 \not> (42 - 4 \cdot 9) \cdot 10 + 2 = 62$

Passo 4			
	$\underbrace{u_2 = j}_{Passo\ 7}$	u_3	u_4
	5	4	4
-	4	6	0
	u_2	u_3	u_4
$\underbrace{0 <}_{Passo\ 5}$	0	8	4

$2 \cdot 6 > (54 - 6 \cdot 9) \cdot 10 + 4 = 4$

$2 \cdot 5 \not> (54 - 5 \cdot 9) \cdot 10 + 4 = 94$

$2 \cdot 9 \not> (84 - 9 \cdot 9) \cdot 10 + 2 = 32$

Passo 4			
	$\underbrace{u_3 = j}_{Passo\ 7}$	u_4	u_5
	8	4	2
-	8	2	8
	u_3	u_4	u_5
$\underbrace{0 <}_{Passo\ 5}$	0	1	4

$$\underbrace{j = 4 > m}_{Passo\ 7}$$

$$p_8 \begin{cases} q = q_0 q_1 q_2 q_3 = 1459 \\ r = \frac{u_{m+1} \ldots u_{m+n}}{d} = \frac{u_4 u_5}{2} = \frac{14}{2} = 7 \end{cases}$$

3.2 Máximo Divisor Comum de Dois Números Naturais

Lema 2. Se $a = bq + r$, então $(a,\ b) = (b,\ r)$ (Lema de Euclides).

Observação 7. O máximo divisor comum de dois números naturais a e de b é denotado por $(a,\ b)$. O leitor interessado em mais detalhes sobre este lema poderá ver [6] e [15].

O *Algoritmo de Euclides* consiste na aplicação reiterada do lema 2, no qual q e r são respectivamente o quociente e o resto na divisão euclidiana de a por b, para se obter $(a,\ b)$, supondo, sem perda de generalidade, que $b \leqslant a$. A tabela abaixo faz essa aplicação, mostrando que tal algoritmo é uma sequência finita de instruções, bem definidas, para produzir corretamente o valor de $(a,\ b)$, o máximo divisor comum de a e b.

Algoritmo de Euclides

Divisão Euclidiana	*Sequência de restos obtida*	*Lema de Euclides*
$a = bq_0 + r_0$	$0 \leqslant r_0 < b$	$(a,\ b) = (b,\ r_0)$
$b = r_0 q_1 + r_1$	$0 \leqslant r_1 < r_0 < b$	$(a,\ b) = (b,\ r_0) = (r_0,\ r_1)$
$r_0 = r_1 q_2 + r_2$	$0 \leqslant r_2 < r_1 < r_0 < b$	$(a,\ b) = (b,\ r_0) = (r_0,\ r_1) = (r_1,\ r_2)$
$\vdots$	$\vdots$	$\vdots$
$r_{n-1} = r_n q_{n+1} + \underbrace{r_{n+1}}_{=0}$	$0 \leqslant r_n < \ldots < r_2 < r_1 < r_0 < b$	$(a,\ b) = (b,\ r_0) = (r_0,\ r_1) = (r_1,\ r_2) = \ldots = (r_n,\ \underbrace{r_{n+1}}_{=0}) = r_n$

Como os restos obtidos na segunda coluna da tabela acima formam uma sequência de números naturais, estritamente decrescente, existirá $n \in \mathbb{N}$ tal que $r_{n+1} = 0$ e assim $mdc(r_n,\ r_{n+1}) = mdc(r_n,\ 0) = r_n$.

As divisões sucessivas do algoritmo euclidiano costumam ser representadas do seguinte modo:

	q_0	q_1	q_2	$\cdots$	q_{n-2}	q_{n-1}	q_n
a	b	r_1	r_2	$\cdots$	r_{n-2}	r_{n-1}	r_n
r_1	r_2	r_3	r_4	$\cdots$	r_n	0	

Exemplo 35. Calculando o $mdc(1001,\ 109)$, usando o algoritmo de Euclides, tem-se:

	9	5	2	4	2
1001	109	20	9	2	1
20	9	2	1	0	

Portanto, $(1001,\ 109) = 1$.

■

Algoritmo de Euclides (MDC)

1 Entrada: a e b.
2 **Se** $a = 1$ **então**
3 **Retorne:** "$(a,\ b) = 1$."
4 **Pare.**
5 **SenãoSe** $b = 1$ **então**
6 **Retorne:** "$(a,\ b) = 1$."
7 **Pare.**
8 **FimSe**
9 **Se** $a \leqslant b$ **então**
10 $x\ := a$
11 **Senão**
12 $x\ := b$
13 **FimSe**
14 $m\ := a$
15 $n\ := b$
16 $q\ := \left\lfloor \frac{a}{b} \right\rfloor$
17 $r\ := a - b \cdot q$
18 $a\ := b$
19 $b\ := r$
20 **Enquanto** $r \neq 0$ **faça**
21 $x\ := r$
22 $q\ := \left\lfloor \frac{a}{b} \right\rfloor$
23 $r\ := a - b \cdot q$
24 $a\ := b$
25 $b\ := r$
26 **FimEnquanto**
27 **Retorne:** "$(m,\ n) = x$."

No algoritmo de Euclides, a quantidade de restos é também a quantidade de divisões efetuadas. O pior caso desse algoritmo ocorrerá para um número máximo de divisões efetuadas, o que equivale a dizer que o pior caso ocorrerá para o maior número de restos obtidos. Mas, para se obter o maior número de restos possível, deve-se obter a sequência $r_0 > r_1 > r_2 > r_3 > ... > r_{n-2} > r_{n-1} > r_n = 0$, para $n + 1$ divisões efetuadas, na qual $r_i = r_{i+1} + 1,\ \forall i \in \{1, 2, ..., (n-1)\}$. Isso implica dizer que $r_{n-1} = 1$ e $q_i = \left\lfloor \frac{r_i}{r_{i+1}} \right\rfloor = 1$ sempre. Assim,

$$\begin{array}{rcl} r_{n-1} & = & 1 \\ r_{n-2} & = & r_{n-1} + 1 \\ r_{n-3} & = & r_{n-2} + 1 \\ r_{n-4} & = & r_{n-3} + 1 \\ \vdots & \vdots & \vdots \\ r_2 & = & r_3 + 1 \end{array} \qquad \begin{array}{rcl} r_1 & = & r_2 + 1 \\ r_0 & = & r_1 + 1 \\ a & = & b + r_0 \end{array}$$

Somando membro a membro as equações acima, obtém-se: $a = n+b \Leftrightarrow n = a-b$. Mas, pela divisão euclidiana de a por b, sob hipótese de ter $q = 1$, tem-se: $\underbrace{a-b}_{n} = r_0 < b \Rightarrow$ $a - b < b$. Assim, $n < b \Rightarrow n + 1 \leqslant b$. Pode-se então dizer que o número total de divisões não ultrapassa o valor de b. Isso quer dizer que o algoritmo de Euclides é linear.

■

O algoritmo de Euclides pode ser estendido de forma a calcular, além do máximo divisor comum d, de dois inteiros a e b, também inteiros x e y tais que

$$ax + by = d.$$

A ideia é expressar cada resto r_i, obtido no cálculo do $(a,\ b)$, em função de a e b. Por exemplo, para $a = bq_0 + r_0$, obtém-se $r_0 = a \cdot (1) + b \cdot (-q_0)$. Para $b = r_0q_1 + r_1$, obtém-se

$$r_1 = b - r_0q_1 = b - (a - b \cdot q_0)q_1 = a(-q_1) + b(1 + q_0q_1)$$

e assim por diante, até chegar à expressão de d como combinação linear de a e b.

Escreve-se todas as divisões efetuadas no cálculo do $mdc(a, b)$ e ao lado de cada equação é colocada a expressão dos restos, sendo x_i e y_i os inteiros procurados.

$$\begin{array}{rcrcrcr}
a & = & bq_0 + r_0 & \text{e} & r_0 & = & ax_0 + by_0 \\
b & = & r_0q_1 + r_1 & \text{e} & r_1 & = & ax_1 + by_1 \\
r_0 & = & r_1q_2 + r_2 & \text{e} & r_2 & = & ax_2 + by_2 \\
r_1 & = & r_2q_3 + r_3 & \text{e} & r_3 & = & ax_3 + by_3 \\
 & \vdots & & \vdots & & \vdots & \\
r_{n-3} & = & r_{n-2}q_{n-1} + r_{n-1} & \text{e} & r_{n-1} & = & ax_{n-1} + by_{n-1} \\
r_{n-2} & = & r_{n-1}q_n + r_n & \text{e} & r_n & = & ax_n + by_n
\end{array}$$

Agora as informações são colocadas numa tabela, na qual as duas últimas colunas são as incógnitas a determinar, observando que $a = 1 \cdot a + 0 \cdot b$, $b = 0 \cdot a + 1 \cdot b$ e que $x_0 = 1$, $y_0 = -q_0$, $x_1 = -q_1$ e $y_1 = 1 + q_0q_1$.

Restos	*Quocientes*	x	y
a	*	1	0
b	*	0	1
r_0	q_0	x_0	y_0
r_1	q_1	x_1	y_1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
r_{n-1}	q_{n-1}	x_{n-1}	y_{n-1}
$r_n = d$	q_n	x_n	y_n

As quatro primeiras linhas da tabela acima são conhecidas, assim é preciso preencher a tabela e no final surgirão os valores procurados x_n e y_n.

Tomando três linhas quaisquer da tabela, supondo que as duas primeiras sejam conhecidas, para ser determinada a terceira, tem-se:

Restos	*Quocientes*	x	y
r_{j-2}	q_{j-2}	x_{j-2}	y_{j-2}
r_{j-1}	q_{j-1}	x_{j-1}	y_{j-1}
r_j	q_j	x_j	y_j

na qual se sabe que $r_j = r_{j-2} - r_{j-1}q_j$, $r_{j-2} = x_{j-2}a + y_{j-2}b$ e $r_{j-1} = x_{j-1}a + y_{j-1}b$. Segue daí que

$$r_j = x_{j-2}a + y_{j-2}b - (x_{j-1}a + y_{j-1}b)q_j, \text{ obtendo}$$

$$r_j = (x_{j-2} - x_{j-1}q_j)a + (y_{j-2} - y_{j-1}q_j)b.$$

Portanto, a partir das duas primeiras linhas sabe-se preencher a tabela toda.

Exemplo 36. Considerando $a = 1234$ e $b = 54$, serão determinados x e y tais que $mdc(a, b) = ax + by$.

Restos	*Quocientes*	x	y
1234	*	1	0
54	*	0	1
46	22	1	-22
8	1	-1	23
6	5	6	-137
2	1	-7	160

Portanto, $mdc(1234,\ 54) = 2 = 1234 \cdot (-7) + 54 \cdot (160)$.

■

Segue abaixo o algoritmo estendido de Euclides.

Algoritmo Euclidiano Estendido

Entrada: a e b, não simultaneamente nulos, $a \geqslant b \geqslant 0$.
Saída: $mdc(a,\ b) = d$, x e y tais que $ax + by = d$.
i_1: Se $b = 0$ então,
  i_2: d assume o valor de a.
  i_3: x assume valor 1.
  i_4: y assume valor 0.
  i_5: Retorne $d,\ x$ e y.
  i_6 : Pare.
FimSe
i_7: r_1 assume o valor de a.
i_8: r_2 assume o valor de b.
i_9: x_1 assume valor 1.
i_{10}: x_2 assume valor 0.
i_{11}: y_1 assume valor 0.
i_{12}: y_2 assume valor 1.
i_{13}: Se $r_2 > 0$ então
  i_{14}: q assume o valor de $\left\lfloor \frac{r_1}{r_2} \right\rfloor$.
  i_{15}: r assume o valor de r_1.
  i_{16}: x assume o valor de x_1.
  i_{17}: y assume o valor de y_1.
  i_{18}: r_1 assume o valor de r_2.
  i_{19}: x_1 assume o valor de x_2.
  i_{20}: y_1 assume o valor de y_2.
  i_{21}: r_2 assume o valor de $r - r_2q$.
  i_{22}: x_2 assume o valor de $x - x_2q$.
  i_{23}: y_2 assume o valor de $y - y_2q$.
  i_{24}: Vá para o passo 13.
FimSe
i_{25}: d assume o valor de r_1.
i_{26}: x assume o valor de x_1.
i_{27}: y assume o valor de y_1.
i_{28}: Retorne d, x e y.
i_{29}: Pare.

Por efetuar a mesma quantidade de divisões produzidas pelo algoritmo de Euclides, esse algoritmo estendido também será linear.

3.3 Divisão de Polinômios

3.3.1 Dispositivo Prático de Briot-Ruffini

Nesta seção, trabalha-se com divisão de polinômios, sendo o divisor na forma $g(x) = x - a$. Lembrando que dado um polinômio não nulo $g(x) = b_0 + b_1x + ... + b_mx^m$, com $b_m \neq 0$, denomina-se m o grau do polinômio e denota-se $m = \delta g(x)$.

O algoritmo da divisão euclidiana pode ser estendido para a divisão de polinômios.

Teorema 3. Dados dois polinômios com coeficientes reais $f(x) = a_0 + a_1x + ... + a_nx^n$ e $g(x) = b_0 + b_1x + ... + b_mx^m$, sendo $g(x) \neq 0$, com $b_m \neq 0$, existem dois polinômios $q(x)$ e $r(x)$, unicamente determinados, tais que

$$f(x) = q(x)g(x) + r(x),$$

no qual $r(x) = 0$ ou $\delta r(x) < \delta g(x)$, sendo $\delta p(x)$ o grau do polinômio $p(x)$.

Demonstração:

Existência:

Se $f(x) = 0$ basta tomar $q(x) = r(x) = 0$. Suponha então $f(x) \neq 0$ e assim pode-se assumir $\delta f(x) = n$.

Se $n < m$, basta tomar $q(x) = 0$ e $r(x) = f(x)$ e assim pode-se assumir $n \geqslant m$.

Será demonstrada a afirmação por indução completa (segunda forma) sobre n, o grau de $f(x)$, sendo $n \geqslant m$.

Se $n = 0$ então $m = 0$, $f(x) = a_0 \neq 0$, $g(x) = b_0 \neq 0$ e tem-se que $f(x) = a_0b_0^{-1}g(x)$, bastando tomar $q(x) = a_0b_0^{-1}$ e $r(x) = 0$.

Considere o polinômio $f_1(x)$ definido por $f(x) = a_nb_m^{-1}x^{n-m}g(x) + f_1(x)$. Observe que $\delta f_1(x) < \delta f(x)$.

Tem-se que $f_1(x) = f(x) - a_nb_m^{-1}x^{n-m}g(x)$ e, pela hipótese de indução, existem $q_1(x)$, $r_1(x)$ tais que

$$f_1(x) = q_1(x)g(x) + r_1(x),$$

no qual $r_1(x) = 0$ ou $\delta r_1(x) < \delta g(x)$.

Daí, segue que

$$f(x) = (q_1(x) + a_nb_m^{-1}x^{n-m})g(x) + r_1(x),$$

tal que $r_1(x) = 0$ ou $\delta r_1(x) < \delta g(x)$. Portanto, tomando $q(x) = (q_1(x) + a_nb_m^{-1}x^{n-m})$ e $r(x) = r_1(x)$ a afirmação será válida para n.

Unicidade:

Suponha que $f(x) = q_1(x)g(x) + r_1(x) = q_2(x)g(x) + r_2(x)$, nas condições exigidas.

Dai, $(q_1(x)-q_2(x))g(x) = r_2(x)-r_1(x)$. Mas se $q_1(x)-q_2(x) \neq 0$, o grau do polinômio da esquerda, da igualdade anterior, é maior do que ou igual a $\delta g(x)$, enquanto que o grau do polinômio da direita $r_2(x)-r_1(x)$ é menor do que $\delta g(x)$, o que é um contradição. Portanto, $q_1(x) = q_2(x)$ e $r_1(x) = r_2(x)$.

■

Dados dois polinômios $p(x) = a_0x^n+a_1x^{n-1}+...+a_{n-1}x+a_n$, $a_0 \neq 0$, e $g(x) = x-a$, deseja-se construir um algoritmo cuja saída sejam os polinômios $q(x)$ e $r(x)$ do Teorema 3.

Considere $q(x) = q_0x^{n-1} + q_1x^{n-2} + q_2x^{n-3} + ... + q_{n-2}x + q_{n-1}$. Assim,

$$q(x) \cdot g(x) = q_0x^n + q_1x^{n-1} + q_2x^{n-2} + ... + q_{n-2}x^2 + q_{n-1}x - aq_0x^{n-1}-$$

$$-aq_1x^{n-2} - ... - aq_{n-3}x^2 - aq_{n-2}x - aq_{n-1} = q_0x^n + (q_1 - aq_0)x^{n-1}+$$

$$+(q_2 - aq_1)x^{n-2} + ... + (q_{n-1} - aq_{n-2})x - aq_{n-1}.$$

Tem-se $\delta(r(x)) = 0 \Leftrightarrow r(x) = k \in \mathbb{R} - \{0\}$ e também:

$$p(x) = q(x) \cdot g(x) + r(x) \Leftrightarrow$$

$$a_0x^n + a_1x^{n-1} + ... + a_{n-1}x + a_n =$$

$$= q_0x^n + (q_1 - aq_0)x^{n-1} + (q_2 - aq_1)x^{n-2} + ... + (q_{n-1} - aq_{n-2})x - aq_{n-1} + k \Leftrightarrow$$

$$a_0 = q_0$$

$$a_1 = q_1 - aq_0 \Rightarrow q_1 = aq_0 + a_1$$

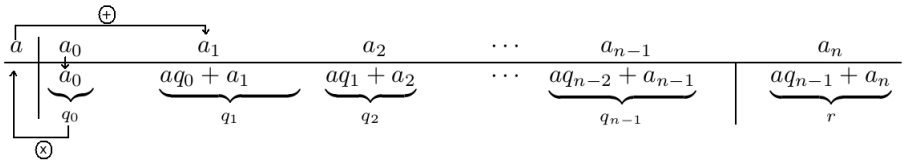
$$a_2 = q_2 - aq_1 \Rightarrow q_2 = aq_1 + a_2$$

$$\vdots$$

$$a_{n-1} = q_{n-1} - aq_{n-2} \Rightarrow q_{n-1} = aq_{n-2} + a_{n-1}$$

$$a_n = -aq_{n-1}k = r(x) - aq_{n-1} \Rightarrow r(x) = aq_{n-1} + a_n$$

Os cálculos efetuados acima representam uma sequência finita de instruções, bem definidas, para produzir o quociente e o resto da divisão de $p(x)$ por $g(x)$. Tal sequência representa o chamado dispositivo prático de Briot-Ruffini. Esquematicamente, pode-se representar esse dispositivo da seguinte forma:



Exemplo 37. Aplicando o dispositivo prático de Briot-Ruffini para a divisão de $p(x) = 2x^4 - 7x^2 + 3x - 1$ por $g(x) = x - 3$, obtém-se:

3	2	0	-7	3	-1
	$\downarrow$				
	2	$\underbrace{2 \cdot 3 + 0}_{6}$	$\underbrace{6 \cdot 3 - 7}_{11}$	$\underbrace{11 \cdot 3 + 3}_{36}$	$\underbrace{36 \cdot 3 - 1}_{107}$

Logo, $q(x) = 2x^3 + 6x^2 + 11x + 36$ e $r(x) = 107$.

■

Algoritmo Divisão de $p(x)$ por $g(x) = x - a$

1 Entrada: $\delta(p(x))$, os coeficientes de $p(x)$, $a_i,\ i \in \{0, 1, 2, ..., n\}$, e a.
2 $q_0 = a_0$
3 **Para** $i = 2$ **até** $n + 1$, **faça**
4 $\quad q(i-1) = a \cdot q(i-2) + a(i-1)$
5 **FimPara**
6 **Retorne**: "$q(x)$ é dado pela soma dos seguintes monômios:"
7 **Para** $j = 0$ **até** $n - 1$, **faça**
8 $\quad q(j)x\hat{}(n-1-j)$
9 **FimPara**
10 **Retorne:** "$r(x) = q(n)$."

As instruções das linhas 3, 4, 7 e 8 serão executadas, cada uma, n vezes. Portanto, o custo do algoritmo será dado por $f(n) = 4n + 3$.

3.4 Números Primos

Definição 4. Diz-se que um número natural p é um número primo se $p \neq 1$ e seus únicos divisores são 1 e p. Um número natural $n > 1$ que não é primo é denominado composto.

Exemplo 38. O número 13 é primo. De fato, os possíveis divisores de 13, diferentes de 1 e 13, são 2, 3, . . . , 11, 12 e nenhum deles divide o número 13.

O Lema seguinte afirma que para verificar se 13 é primo basta observar que 2 e 3 não dividem o número 13.

Lema 3. Se um natural $n > 1$ é composto então n admite um divisor primo menor do que ou igual a $\sqrt{n}$.

Demonstração:

Se n é composto então n possui um divisor a com $1 < a < n$. Logo $n = a \cdot b$, sendo $1 < b < n$. Sem perda de generalidade suponha $a \leqslant b$. Segue que $a \cdot a \leqslant a \cdot b = n$, isto é, $a^2 \leqslant n$, logo $a \leqslant \sqrt{n}$. Agora, um divisor de n é primo ou tem um divisor primo q. Nos dois casos n possui um divisor primo menor do que ou igual a $\sqrt{n}$.

■

O Lema 3 diz que um inteiro $n > 1$ é primo se n não é divisível por nenhum primo menor do que ou igual a $\sqrt{n}$.

Tal Lema é usado no algoritmo seguinte, cuja entrada é um número natural $n > 1$ e a saída diz se n é primo ou se n é composto.

```
Algoritmo Teste de Primalidade
1 Entrada: n ⩾ 2, (n ∈ ℤ)
2 F := 2
3 Se F > √n então
4       Retorne: "n é primo."
5       Pare.
6 FimSe
7 Se n/F ∈ ℤ então
8       Retorne: "F é fator de n."
9       Pare.
10 Senão
11      F := F + 1
12      Volte a executar a partir da linha 3.
13 FimSe
```

O lema 3 também permite criar um algoritmo (Crivo de Eratóstenes) para listar todos os números primos até um certo inteiro n. Por exemplo, listar todos os primos inferiores a 200.

Começa-se escrevendo todos os números de 2 até 200. Em seguida, risca-se todos os inteiros compostos que figuram nesta lista segundo o roteiro abaixo.

- 2 é primo, risque todos os números pares maiores do que 2, pois não são primos.
- Todos os números não riscados inferiores a $2^2 = 4$ são primos. Neste caso, obtém-se o primo 3. Risque todos os múltiplos de 3.
- Todos os números não riscados inferiores a $3^2 = 9$ são primos. Neste caso, obtém-se os primos 2, 3, 5 e 7. Risque todos os seus múltiplos que ainda não foram riscados.
- Todos os números não riscados inferiores a $7^2 = 49$ são primos. Neste caso, obtém-se os primos 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 e 47.
- Como $47^2 > 200$, todos os números não riscados na tabela são primos.

Conforme comentado acima, observa-se que alguns números são riscados mais de uma vez. Riscar os múltiplos de um primo p, significa riscar de p em p, isto é, riscar $2p = p+p$, $3p = p+p+p, \cdots$. Como os múltiplos de p, que também são múltiplos de primos menores do que p, já foram riscados da lista, pode-se começar a riscar de p em p, a partir do menor múltiplo de p, que não é múltiplo de um primo menor do que p, isto é, a partir de p^2. Assim pode-se riscar de p em p, a partir de p^2, diminuindo um pouco o trabalho.

Como o único primo par é o 2, deseja-se agora escrever um algoritmo que liste os primos ímpares menores do que ou iguais a um número ímpar n, de modo que um programador possa transcrever para uma linguagem de máquina.

A lista dos ímpares corresponderá a um vetor. A cada entrada do vetor estão associados dois números: o valor com que a entrada foi preenchida e o índice que determina sua posição no vetor.

Começa-se criando um vetor com $\frac{n-1}{2}$ posições. Essa expressão indica o número de ímpares entre 3 e n. A entrada do índice j do vetor corresponde ao j-ésimo número ímpar $2j + 1$. No início todas as entradas são preenchidas com o número 1 e riscar um número $2j + 1$ significa substituir o 1 por zero na posição j do vetor.

Crivo de Eratóstenes

- Entrada: Inteiro positivo ímpar n.
- Saída: lista dos primos ímpares menores do que ou iguais a n.
- Passo 1: Comece criando um vetor v de $\left(\frac{n-1}{2}\right)$ posições, cada uma das quais deve estar preenchida com o número 1 e tomando $P = 3$.
- Passo 2: Se $P^2 > n$ escreva os números $2j + 1$, para os quais a j-ésima entrada do vetor é 1 e pare. Senão vá para o passo 3.
- Passo 3: Se a posição $\frac{P-1}{2}$ do vetor está preenchida com 0, incremente P de 2 e volte ao passo 2. Senão vá para o passo 4.
- Passo 4: Atribua o valor P^2 a uma nova variável T. Substitua por 0 o valor na posição $\frac{T-1}{2}$ do vetor v e incremente T de $2P$. Repita estas duas instruções até ocorrer $T > n$. Depois incremente P de 2 e volte ao passo 2.

Observação 8. Como T e P são ímpares tem-se que $T + P$ é par. Como está sendo riscado de P em P, o ímpar a ser riscado em seguida a T será $T + 2P$.

O vetor inicial usado no Crivo de Eratóstenes ocupará muita memória se o número inteiro n for muito grande. Por outro lado, as operações envolvidas são simples e sua programação é muito fácil. Isto possibilita usar o crivo como um exemplo de algoritmo a ser implementado no ensino básico.

3.5 Multiplicação de Matrizes Quadradas

Definição 5. Dadas as matrizes quadradas $A = [a_{ij}]_{n\times n}$ e $B = [b_{ij}]_{n\times n}$, define-se o produto $C = A \times B$, $C = [c_{ij}]_{n\times n}$, por meio da seguinte expressão

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj},\ 1 \leqslant i,\ j \leqslant n.$$

Esse procedimento representa uma sequência finita de instruções, bem definidas, para produzir corretamente o valor de c_{ij}. Logo, trata-se de um algoritmo (clássico).

Analisando o custo desse algoritmo, por meio da quantidade de adições e multiplicações efetuadas, verifica-se que, para a descoberta do número de cada c_{ij}, do algoritmo na definição acima, são realizadas n multiplicações e $n-1$ adições. Assim, para preencher a matriz C, com todos os seus elementos c_{ij}, serão realizadas, no total, $n \times n^2 = n^3$ multiplicações e $(n-1) \times n^2 = n^3 - n^2$ adições. Então, tal algoritmo possui custo igual a $O(n^3)$.

No caso de matrizes de ordem 2, utilizando o algoritmo da Definição 5, obtém-se, na produção de C, um total de $n^3 = 2^3 = 8$ multiplicações, mas, em 1969, o matemático Volker Strassen descobriu outro algoritmo, efetuando 7 multiplicações, conforme o que segue.

Algoritmo de Strassen $(A_2,\ B_2)$

$p_1 := (a_{11} + a_{22}) \times (b_{11} + b_{22})$
$p_2 := (a_{21} + a_{22}) \times b_{11}$
$p_3 := a_{11} \times (b_{12} - b_{22})$
$p_4 := a_{22} \times (b_{21} - b_{11})$
$p_5 := (a_{11} + a_{12}) \times b_{22}$
$p_6 := (a_{21} - a_{11}) \times (b_{11} + b_{12})$
$p_7 := (a_{12} - a_{22}) \times (b_{21} + b_{22})$
$c_{11} := p_1 + p_4 - p_5 + p_7$
$c_{12} := p_3 + p_5$
$c_{21} := p_2 + p_4$
$c_{22} := p_1 + p_3 - p_2 + p_6$
Retorne: $C = [c_{ij}]_{2\times 2}$.

No algoritmo da Definição 5 eram realizadas no total $n^3-n^2 = 2^3-2^2 = 4$ adições. O algoritmo de Strassen realiza um total de 18 adições algébricas. Houve aumento no número de adições. Será que isso compromete o algoritmo de Strassen ao ponto de ser menos eficiente do que o algoritmo clássico? Em termos de multiplicação de matrizes quadradas de ordem 2 sim, pois seriam um total de 12 operações, no algoritmo clássico, contra 25, no algoritmo de Strassen. Acontece que o algoritmo de Strassen também será utilizado no produto de matrizes quadradas de ordem superior a 2, como será visto adiante. Neste caso, será verificado que, para n suficientemente grande, o algoritmo de Strassen é mais eficiente do que o algoritmo clássico.

Considere duas matrizes quadradas, $A = [a_{ij}]_{n\times n}$ e $B = [b_{ij}]_{n\times n}$, $n > 2$. Suponha que $n = 2^k$, $k > 1$, para que seja possível dividir tanto A quanto B em quatro blocos, representados por A_{ij} e B_{ij}, todos com mesmo tamanho, como abaixo:

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ - & - \\ A_{21} & A_{22} \end{array}\right] \text{ e } B = \left[\begin{array}{c|c} B_{11} & B_{12} \\ - & - \\ B_{21} & B_{22} \end{array}\right].$$

Sabe-se, da Álgebra Linear, que o produto de A por B pode ser realizado por blocos, ou seja,

$$C = A \times B = \left[\begin{array}{c|c} C_{11} & C_{12} \\ - & - \\ C_{21} & C_{22} \end{array}\right],$$

que também é um algoritmo, no qual o bloco C_{ij} é tal que $C_{ij} = A_{i1} \times B_{1j} + A_{i2} \times B_{2j}$.

Exemplo 39. Dadas as matrizes $A = \begin{bmatrix} 1 & 0 & 2 & 1 \\ 2 & 1 & -1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 3 & 1 & 0 \end{bmatrix}$ e $B = \begin{bmatrix} 0 & 1 & 2 & 0 \\ 1 & 2 & -1 & 2 \\ 1 & 1 & 0 & 0 \\ 2 & 0 & 0 & -2 \end{bmatrix}$, calcula-se $C = A \times B$ pelo algoritmo logo acima, da seguinte forma:

$A_{11} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$, $A_{12} = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix}$, $A_{21} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$ e $A_{22} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$.

$B_{11} = \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix}$, $A_{12} = \begin{bmatrix} 2 & 0 \\ -1 & 2 \end{bmatrix}$, $A_{21} = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$ e $A_{22} = \begin{bmatrix} 0 & 0 \\ 0 & -2 \end{bmatrix}$.

Assim,

$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21} = \begin{bmatrix} 4 & 3 \\ 0 & 3 \end{bmatrix}$.

$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22} = \begin{bmatrix} 2 & -2 \\ 3 & 2 \end{bmatrix}$.

$C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21} = \begin{bmatrix} 2 & 2 \\ 4 & 7 \end{bmatrix}$.

$C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22} = \begin{bmatrix} 4 & -2 \\ -3 & 6 \end{bmatrix}$.

$$\text{Portanto, } C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} 4 & 3 & 2 & -2 \\ 0 & 3 & 3 & 2 \\ 2 & 2 & 4 & -2 \\ 4 & 7 & -3 & 6 \end{bmatrix}.$$

Definição 6. Um ***algoritmo de divisão e conquista***, também chamado de recursivo, é aquele que desmembra o problema em vários subproblemas (dividir) semelhantes ao problema original, mas menores em tamanho, retomando-os (conquista) para obter as soluções deles, recursivamente, combinando-as com o objetivo de criar uma solução para o problema original.

Em matrizes quadradas de ordem $n = 2^k,\ k > 1$, o algorimto de Strassen pode ser aplicado recursivamente de modo a produzir blocos (submatrizes quadradas) de ordem 2, para que depois sejam feitas as multiplicações utilizando o procedimento Prodmat2. Assim, na primeira divisão ocorrida, A e B possuirão blocos que são matrizes quadradas de ordem $\frac{n}{2}$ cada. Na segunda divisão, A e B possuirão submatrizes quadradas de ordem $\frac{\frac{n}{2}}{2} = \frac{n}{4}$. O processo de divisão ocorrerá até que todos os blocos de A como os de B possuam ordem 2, para que seja iniciada a multiplicação da matrizes menores, para retornar a matriz $C = A \times B$. Trata-se portanto de um algoritmo de divisão e conquista.

Segue abaixo o algoritmo, chamado Prodmat, que utiliza o algoritmo de Strassen, recursivamente.

Prodmat

- Entrada: A ordem $n = 2^k$ das matrizes quadradas, A e B, bem como seus elementos.
- Saída: A matriz $C = A \times B$.
- Passo 1: Se $n = 2$ então aplique o procedimento Prodmat2 em A e B. Mas se $n > 2$ então faça $m := \frac{n}{2}$.
- Passo 2: Construa a matriz A_{11}, com as linhas de 1 a m e colunas de 1 a m, de A.
- Passo 3: Construa A_{12}, com as linhas de 1 a m e colunas de $m+1$ a n.
- Passo 4: Construa A_{21}, com as linhas de $m+1$ a n e colunas de 1 a m.
- Passo 5: Construa A_{22}, com as linhas de $m+1$ a n e colunas de $m+1$ a n.
- Passo 6: Construa B_{11}, com as linhas de 1 a m e colunas de 1 a m, de B.
- Passo 7: Construa B_{12}, com as linhas de 1 a m e colunas de $m+1$ a n.
- Passo 8: Construa B_{21}, com as linhas de $m+1$ a n e colunas de 1 a m.
- Passo 9: Construa B_{22}, com as linhas de $m+1$ a n e colunas de $m+1$ a n.
- Passo 10: Construa a matriz P_1 aplicando o algoritmo de Strassen para obter o produto das matrizes $(A_{11} + A_{22})$ e $(B_{11} + B_{22})$.
- Passo 11: Construa P_2 aplicando o algoritmo de Strassen para obter o produto das matrizes $(A_{21} + A_{22})$ e B_{11}.
- Passo 12: Construa P_3 aplicando o algoritmo de Strassen para obter o produto das matrizes A_{11} e $(B_{12} - B_{22})$.
- Passo 13: Construa P_4 aplicando o algoritmo de Strassen para obter o produto das matrizes A_{22} e $(B_{21} - B_{11})$.
- Passo 14: Construa P_5 aplicando o algoritmo de Strassen para obter o produto das matrizes $(A_{11} + A_{12})$ e B_{22}.

♦ Passo 15: Construa P_6 aplicando o algoritmo de Strassen para obter o produto das matrizes $(A_{21} - A_{11})$ e $(B_{11} + B_{12})$.
♦ Passo 16: Construa P_7 aplicando o algoritmo de Strassen para obter o produto das matrizes $(A_{12} - A_{22})$ e $(B_{21} + B_{22})$.
♦ Passo 17: Obtenha a matriz $C_{11} = P_1 + P_4 - P_5 + P_7$.
♦ Passo 18: Obtenha a matriz $C_{12} = P_3 + P_5$.
♦ Passo 19: Obtenha a matriz $C_{21} = P_2 + P_4$.
♦ Passo 20: Obtenha a matriz $C_{22} = P_1 + P_3 - P_2 + P_6$.
♦ Passo 21: Retorne $C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$.

Além da adição algébrica de números que Prodmat2 realiza, ocorrerão também adições algébricas de matrizes por Prodmat, segundo o seguinte algoritmo.

Adição Algébrica de Matrizes

- Entrada: A ordem m das matrizes quadradas A e B, bem como seus elementos, e as operações de adição e subtração.
- Saída: $A + B$ ou $A - B$.
- Passo 1: i assume valor 1.
  Observação 9. i indica a linha da matriz.
- Passo 2: Se $i \leqslant m$ então considere $j = 1$. Mas se $i > m$ então execute o passo 6.
  Observação 10. j indica a coluna da matriz.
  - Passo 3: Se $j > m$ então continue executando a partir do passo 5. Caso $j \leqslant m$ e a operação for de adição então faça $c_{ij} = a_{ij} + b_{ij}$. Mas se $j \leqslant m$ e a operação for de subtração então faça $c_{ij} = a_{ij} - b_{ij}$.
    Observação 11. $j > m$ indica que todos os números da linha i, de A, já foram somados ou subtraídos com seus correspondentes na linha i, de B.
  - Passo 4: Incremente j e volte a executar a partir do passo 3.
- Passo 5: Incremente i e volte a executar a partir do passo 2.
  Observação 12. Neste passo, avança-se para a próxima linha, tanto de A como de B, para que seja trabalhada a adição ou subtração dos números correspondentes.
- Passo 6: Retorne a matriz $A + B$ ou $A - B$, para a operação utilizada por Prodmat.

■

Exemplo 40. Sejam $A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 & 3 & 3 & 3 & 3 \\ 4 & 4 & 4 & 4 & 4 & 4 & 4 & 4 \\ 5 & 5 & 5 & 5 & 5 & 5 & 5 & 5 \\ 6 & 6 & 6 & 6 & 6 & 6 & 6 & 6 \\ 7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \end{bmatrix} = B$, o produto $C = A \times B$ é obtido, por Prodmat, da seguinte forma:

$$A=\underbrace{\left[\begin{array}{cc}\overbrace{\begin{array}{cccc}0&0&0&0\\1&1&1&1\\2&2&2&2\\3&3&3&3\\4&4&4&4\\5&5&5&5\\6&6&6&6\\7&7&7&7\end{array}}^{A_{11}=B_{11}} & \overbrace{\begin{array}{cccc}0&0&0&0\\1&1&1&1\\2&2&2&2\\3&3&3&3\\4&4&4&4\\5&5&5&5\\6&6&6&6\\7&7&7&7\end{array}}^{A_{12}=B_{12}}\\ \underbrace{}_{A_{21}=B_{21}} & \underbrace{}_{A_{22}=B_{22}}\end{array}\right]}_{i_1}=B;\ \underbrace{P_1}_{i_{10}}=\underbrace{\left[\begin{array}{cc}\overbrace{\begin{array}{cc}4&4\\6&6\\8&8\\10&10\end{array}}^{A^1_{11}} & \overbrace{\begin{array}{cc}4&4\\6&6\\8&8\\10&10\end{array}}^{A^1_{12}}\\ \underbrace{}_{A^1_{21}} & \underbrace{}_{A^1_{22}}\end{array}\right]}_{(A_{11}+A_{22})\times(B_{11}+B_{22})}\times\left[\begin{array}{cc}\overbrace{\begin{array}{cc}4&4\\6&6\\8&8\\10&10\end{array}}^{B^1_{11}} & \overbrace{\begin{array}{cc}4&4\\6&6\\8&8\\10&10\end{array}}^{B^1_{12}}\\ \underbrace{}_{B^1_{21}} & \underbrace{}_{B^1_{22}}\end{array}\right]=$$

$$\underbrace{\left[\begin{array}{cc}\overbrace{\begin{array}{cc}112&112\\168&168\\224&224\\280&280\end{array}}^{C^1_{11}} & \overbrace{\begin{array}{cc}112&112\\168&168\\224&224\\280&280\end{array}}^{C^1_{12}}\\ \underbrace{}_{C^1_{21}} & \underbrace{}_{C^1_{22}}\end{array}\right]}_{i_{21}};\ \underbrace{P^1_1}_{i_{10}}=\underbrace{\begin{bmatrix}336&336\\448&448\end{bmatrix}}_{(A^1_{11}+A^1_{22})\times(B^1_{11}+B^1_{22})};\ \underbrace{P^1_2}_{i_{11}}=\underbrace{\begin{bmatrix}160&160\\200&200\end{bmatrix}}_{(A^1_{21}+A^1_{22})\times B^1_{11}};\ \underbrace{P^1_3}_{i_{12}}=\underbrace{\begin{bmatrix}-32&-32\\-48&-48\end{bmatrix}}_{A^1_{11}\times(B^1_{12}-B^1_{22})};$$

$$\underbrace{P^1_4}_{i_{13}}=\underbrace{\begin{bmatrix}64&64\\80&80\end{bmatrix}}_{A^1_{22}\times(B^1_{21}-B^1_{11})};\ \underbrace{P^1_5}_{i_{14}}=\underbrace{\begin{bmatrix}114&114\\216&216\end{bmatrix}}_{(A^1_{11}+A^1_{12})\times B^1_{22}};\ \underbrace{P^1_6}_{i_{15}}=\underbrace{\begin{bmatrix}80&80\\80&80\end{bmatrix}}_{(A^1_{21}-A^1_{11})\times(B^1_{11}+B^1_{12})};\ \underbrace{P^1_7}_{i_{16}}=\underbrace{\begin{bmatrix}-144&-144\\-144&-144\end{bmatrix}}_{(A^1_{12}-A^1_{22})\times(B^1_{21}+B^1_{22})};\ \underbrace{C^1_{11}}_{i_{17}}=\overbrace{\begin{bmatrix}112&112\\168&168\end{bmatrix}}^{P^1_1+P^1_4-P^1_5+P^1_7};$$

$$\underbrace{C^1_{12}}_{i_{18}}=\overbrace{\begin{bmatrix}112&112\\168&168\end{bmatrix}}^{P^1_3+P^1_5};\ \underbrace{C^1_{21}}_{i_{19}}=\overbrace{\begin{bmatrix}224&224\\280&280\end{bmatrix}}^{P^1_2+P^1_4};\ \underbrace{C^1_{22}}_{i_{20}}=\overbrace{\begin{bmatrix}224&224\\280&280\end{bmatrix}}^{P^1_1+P^1_3-P^1_2+P^1_6}.$$

$$\underbrace{P_2}_{i_{11}}=\underbrace{\left[\begin{array}{cc}\overbrace{\begin{array}{cc}8&8\\10&10\\12&12\\14&14\end{array}}^{A^2_{11}} & \overbrace{\begin{array}{cc}8&8\\10&10\\12&12\\14&14\end{array}}^{A^2_{12}}\\ \underbrace{}_{A^2_{21}} & \underbrace{}_{A^2_{22}}\end{array}\right]}_{(A_{21}+A_{22})\times B_{11}}\times\left[\begin{array}{cc}\overbrace{\begin{array}{cc}0&0\\1&1\\2&2\\3&3\end{array}}^{B^2_{11}} & \overbrace{\begin{array}{cc}0&0\\1&1\\2&2\\3&3\end{array}}^{B^2_{12}}\\ \underbrace{}_{B^2_{21}} & \underbrace{}_{B^2_{22}}\end{array}\right]=\underbrace{\left[\begin{array}{cc}\overbrace{\begin{array}{cc}48&48\\60&60\\72&72\\84&84\end{array}}^{C^2_{11}} & \overbrace{\begin{array}{cc}48&48\\60&60\\72&72\\84&84\end{array}}^{C^2_{12}}\\ \underbrace{}_{C^2_{21}} & \underbrace{}_{C^2_{22}}\end{array}\right]}_{i_{21}};\ \underbrace{P^2_1}_{i_{10}}=\underbrace{\begin{bmatrix}120&120\\144&144\end{bmatrix}}_{(A^2_{11}+A^2_{22})\times(B^2_{11}+B^2_{22})};$$

$$\underbrace{P^2_2}_{i_{11}}=\underbrace{\begin{bmatrix}24&24\\28&28\end{bmatrix}}_{(A^2_{21}+A^2_{22})\times B^2_{11}};\ \underbrace{P^2_3}_{i_{12}}=\underbrace{\begin{bmatrix}-32&-32\\-40&-40\end{bmatrix}}_{A^2_{11}\times(B^2_{12}-B^2_{22})};\ \underbrace{P^2_4}_{i_{13}}=\underbrace{\begin{bmatrix}48&48\\56&56\end{bmatrix}}_{A^2_{22}\times(B^2_{21}-B^2_{11})};\ \underbrace{P^2_5}_{i_{14}}=\underbrace{\begin{bmatrix}80&80\\100&100\end{bmatrix}}_{(A^2_{11}+A^2_{12})\times B^2_{22}};\ \underbrace{P^2_6}_{i_{15}}=\underbrace{\begin{bmatrix}8&8\\8&8\end{bmatrix}}_{(A^2_{21}-A^2_{11})\times(B^2_{11}+B^2_{12})};$$

$$\underbrace{P^2_7}_{i_{16}}=\underbrace{\begin{bmatrix}-40&-40\\-40&-40\end{bmatrix}}_{(A^2_{12}-A^2_{22})\times(B^2_{21}+B^2_{22})};\ \underbrace{C^2_{11}}_{i_{17}}=\overbrace{\begin{bmatrix}48&48\\60&60\end{bmatrix}}^{P^2_1+P^2_4-P^2_5+P^2_7};\ \underbrace{C^2_{12}}_{i_{18}}=\overbrace{\begin{bmatrix}48&48\\60&60\end{bmatrix}}^{P^2_3+P^2_5};\ \underbrace{C^2_{21}}_{i_{19}}=\overbrace{\begin{bmatrix}72&72\\84&84\end{bmatrix}}^{P^2_2+P^2_4};\ \underbrace{C^2_{22}}_{i_{20}}=\overbrace{\begin{bmatrix}72&72\\84&84\end{bmatrix}}^{P^2_1+P^2_3-P^2_2+P^2_6}.$$

$$\underbrace{P_3}_{i_{12}} = \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{A^3_{11}}_{i_4}}^{i_2} & \overbrace{\underbrace{A^3_{12}}_{i_5}}^{i_3} \\ \underbrace{A^3_{21}}_{i_4} & \underbrace{A^3_{22}}_{i_5} \end{array}\right]}_{A_{11}\times(B_{12}-B_{22})} \times \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{B^3_{11}}_{i_8}}^{i_6} & \overbrace{\underbrace{B^3_{12}}_{i_9}}^{i_7} \\ \underbrace{B^3_{21}}_{i_8} & \underbrace{B^3_{22}}_{i_9} \end{array}\right]}_{A_{11}\times(B_{12}-B_{22})} = \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{C^3_{11}}_{i_{21}}}^{} & \overbrace{\underbrace{C^3_{12}}_{i_{21}}}^{} \\ \underbrace{C^3_{21}}_{i_{21}} & \underbrace{C^3_{22}}_{i_{21}} \end{array}\right]}_{i_{21}};$$

$$\underbrace{P^3_1}_{i_{10}} = \underbrace{\left[\begin{array}{cc} -32 & -32 \\ -64 & -64 \end{array}\right]}_{(A^3_{11}+A^3_{22})\times(B^3_{11}+B^3_{22})}; \underbrace{P^3_2}_{i_{11}} = \underbrace{\left[\begin{array}{cc} -32 & -32 \\ -48 & -48 \end{array}\right]}_{(A^3_{21}+A^3_{22})\times B^3_{11}}; \underbrace{P^3_3}_{i_{12}} = \underbrace{\left[\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array}\right]}_{A^3_{11}\times(B^3_{12}-B^3_{22})}; \underbrace{P^3_4}_{i_{13}} = \underbrace{\left[\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array}\right]}_{A^3_{22}\times(B^2_{21}-B^2_{11})}; \underbrace{P^3_5}_{i_{14}} = \underbrace{\left[\begin{array}{cc} 0 & 0 \\ -16 & -16 \end{array}\right]}_{(A^3_{11}+A^3_{12})\times B^3_{22}};$$

$$\underbrace{P^3_6}_{i_{15}} = \underbrace{\left[\begin{array}{cc} -32 & -32 \\ -32 & -32 \end{array}\right]}_{(A^3_{21}-A^3_{11})\times(B^3_{11}+B^3_{12})}; \underbrace{P^3_7}_{i_{16}} = \underbrace{\left[\begin{array}{cc} 32 & 32 \\ 32 & 32 \end{array}\right]}_{(A^3_{12}-A^3_{22})\times(B^3_{21}+B^2_{22})}; \underbrace{C^3_{11}}_{i_{17}} = \overbrace{\left[\begin{array}{cc} 0 & 0 \\ -16 & -16 \end{array}\right]}^{P^3_1+P^3_4-P^3_5+P^3_7}; \underbrace{C^3_{12}}_{i_{18}} = \overbrace{\left[\begin{array}{cc} 0 & 0 \\ -16 & -16 \end{array}\right]}^{P^3_3+P^3_5};$$

$$\underbrace{C^3_{21}}_{i_{19}} = \overbrace{\left[\begin{array}{cc} -32 & -32 \\ -48 & -48 \end{array}\right]}^{P^3_2+P^3_4}; \underbrace{C^3_{22}}_{i_{20}} = \overbrace{\left[\begin{array}{cc} -32 & -32 \\ -48 & -48 \end{array}\right]}^{P^3_1+P^3_3-P^3_2+P^3_6}.$$

$$\underbrace{P_4}_{i_{13}} = \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{A^4_{11}}_{i_4}}^{i_2} & \overbrace{\underbrace{A^4_{12}}_{i_5}}^{i_3} \\ \underbrace{A^4_{21}}_{i_4} & \underbrace{A^4_{22}}_{i_5} \end{array}\right]}_{A_{22}\times(B_{21}-B_{11})} \times \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{B^4_{11}}_{i_8}}^{i_6} & \overbrace{\underbrace{B^4_{12}}_{i_9}}^{i_7} \\ \underbrace{B^4_{21}}_{i_8} & \underbrace{B^4_{22}}_{i_9} \end{array}\right]}_{A_{22}\times(B_{21}-B_{11})} = \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{C^4_{11}}_{i_{21}}}^{} & \overbrace{\underbrace{C^4_{12}}_{i_{21}}}^{} \\ \underbrace{C^4_{21}}_{i_{21}} & \underbrace{C^4_{22}}_{i_{21}} \end{array}\right]}_{i_{21}}; \underbrace{P^4_1}_{i_{10}} = \underbrace{\left[\begin{array}{cc} 160 & 160 \\ 192 & 192 \end{array}\right]}_{(A^4_{11}+A^4_{22})\times(B^4_{11}+B^4_{22})};$$

$$\underbrace{P^4_2}_{i_{11}} = \underbrace{\left[\begin{array}{cc} 96 & 96 \\ 112 & 112 \end{array}\right]}_{(A^4_{21}+A^4_{22})\times B^4_{11}}; \underbrace{P^4_3}_{i_{12}} = \underbrace{\left[\begin{array}{cc} 0 & 0 \\ 0 & 0 \end{array}\right]}_{A^4_{11}\times(B^4_{12}-B^4_{22})}; \underbrace{P^4_4}_{i_{13}} = \underbrace{\left[\begin{array}{cc} 64 & 64 \\ 80 & 80 \end{array}\right]}_{A^4_{22}\times(B^4_{21}-B^4_{11})}; \underbrace{P^4_5}_{i_{14}} = \underbrace{\left[\begin{array}{cc} 64 & 64 \\ 80 & 80 \end{array}\right]}_{(A^4_{11}+A^4_{12})\times B^4_{22}}; \underbrace{P^4_6}_{i_{15}} = \underbrace{\left[\begin{array}{cc} 32 & 32 \\ 32 & 32 \end{array}\right]}_{(A^4_{21}-A^4_{11})\times(B^4_{11}+B^4_{12})};$$

$$\underbrace{P^4_7}_{i_{16}} = \underbrace{\left[\begin{array}{cc} -32 & -32 \\ -32 & -32 \end{array}\right]}_{(A^4_{12}-A^4_{22})\times(B^4_{21}+B^4_{22})}; \underbrace{C^4_{11}}_{i_{17}} = \overbrace{\left[\begin{array}{cc} 64 & 64 \\ 80 & 80 \end{array}\right]}^{P^4_1+P^4_4-P^4_5+P^4_7}; \underbrace{C^4_{12}}_{i_{18}} = \overbrace{\left[\begin{array}{cc} 64 & 64 \\ 80 & 80 \end{array}\right]}^{P^4_3+P^4_5}; \underbrace{C^4_{21}}_{i_{19}} = \overbrace{\left[\begin{array}{cc} 96 & 96 \\ 112 & 112 \end{array}\right]}^{P^4_2+P^4_4}; \underbrace{C^4_{22}}_{i_{20}} = \overbrace{\left[\begin{array}{cc} 96 & 96 \\ 112 & 112 \end{array}\right]}^{P^4_1+P^4_3-P^4_2+P^4_6}.$$

$$\underbrace{P_5}_{i_{14}} = \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{A^5_{11}}_{i_4}}^{i_2} & \overbrace{\underbrace{A^5_{12}}_{i_5}}^{i_3} \\ \underbrace{A^5_{21}}_{i_4} & \underbrace{A^5_{22}}_{i_5} \end{array}\right]}_{(A_{11}+A_{12})\times B_{22}} \times \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{B^5_{11}}_{i_8}}^{i_6} & \overbrace{\underbrace{B^5_{12}}_{i_9}}^{i_7} \\ \underbrace{B^5_{21}}_{i_8} & \underbrace{B^5_{22}}_{i_9} \end{array}\right]}_{(A_{11}+A_{12})\times B_{22}} = \underbrace{\left[\begin{array}{cc} \overbrace{\underbrace{C^5_{11}}_{i_{21}}}^{} & \overbrace{\underbrace{C^5_{12}}_{i_{21}}}^{} \\ \underbrace{C^5_{21}}_{i_{21}} & \underbrace{C^5_{22}}_{i_{21}} \end{array}\right]}_{i_{21}}; \underbrace{P^5_1}_{i_{10}} = \underbrace{\left[\begin{array}{cc} 88 & 88 \\ 176 & 176 \end{array}\right]}_{(A^5_{11}+A^5_{22})\times(B^5_{11}+B^5_{22})};$$

$$\underbrace{P^5_2}_{i_{11}} = \underbrace{\left[\begin{array}{cc} 72 & 72 \\ 108 & 108 \end{array}\right]}_{(A^5_{21}+A^5_{22})\times B^5_{11}}; \underbrace{P^5_3}_{i_{12}} = \underbrace{\left[\begin{array}{cc} 0 & 0 \\ -8 & -8 \end{array}\right]}_{A^5_{11}\times(B^5_{12}-B^5_{22})}; \underbrace{P^5_4}_{i_{13}} = \underbrace{\left[\begin{array}{cc} 16 & 16 \\ 24 & 24 \end{array}\right]}_{A^5_{22}\times(B^5_{21}-B^5_{11})}; \underbrace{P^5_5}_{i_{14}} = \underbrace{\left[\begin{array}{cc} 0 & 0 \\ 52 & 52 \end{array}\right]}_{(A^5_{11}+A^5_{12})\times B^5_{22}}; \underbrace{P^5_6}_{i_{15}} = \underbrace{\left[\begin{array}{cc} 72 & 72 \\ 72 & 72 \end{array}\right]}_{(A^5_{21}-A^5_{11})\times(B^5_{11}+B^5_{12})};$$

$$\underbrace{P_7^5}_{i_{16}} = \underbrace{\begin{bmatrix} -104 & -104 \\ -104 & -104 \end{bmatrix}}_{(A_{12}^5 - A_{22}^5)\times(B_{21}^5+B_{22}^5)}; \underbrace{C_{11}^5}_{i_{17}} = \overbrace{\begin{bmatrix} 0 & 0 \\ 44 & 44 \end{bmatrix}}^{P_1^5+P_4^5-P_5^5+P_7^5}; \underbrace{C_{12}^5}_{i_{18}} = \overbrace{\begin{bmatrix} 0 & 0 \\ 44 & 44 \end{bmatrix}}^{P_3^5+P_5^5}; \underbrace{C_{21}^5}_{i_{19}} = \overbrace{\begin{bmatrix} 88 & 88 \\ 132 & 132 \end{bmatrix}}^{P_2^5+P_4^5}; \underbrace{C_{22}^5}_{i_{20}} = \overbrace{\begin{bmatrix} 88 & 88 \\ 132 & 132 \end{bmatrix}}^{P_1^5+P_3^5-P_2^5+P_6^5}.$$

$$\underbrace{P_6}_{i_{15}} = \underbrace{\begin{bmatrix} \overbrace{\underbrace{A_{11}^6}_{i_4}}^{i_2} & \overbrace{\underbrace{A_{12}^6}_{i_5}}^{i_3} \\ \underbrace{A_{21}^6}_{i_4} & \underbrace{A_{22}^6}_{i_5} \end{bmatrix}}_{(A_{21}-A_{11})\times(B_{11}+B_{12})} \times \underbrace{\begin{bmatrix} \overbrace{\underbrace{B_{11}^6}_{i_8}}^{i_6} & \overbrace{\underbrace{B_{12}^6}_{i_9}}^{i_7} \\ \underbrace{B_{21}^6}_{i_8} & \underbrace{B_{22}^6}_{i_9} \end{bmatrix}}_{(A_{21}-A_{11})\times(B_{11}+B_{12})} = \underbrace{\begin{bmatrix} \overbrace{\underbrace{48\ 48\\48\ 48}}^{C_{11}^6} & \overbrace{\underbrace{48\ 48\\48\ 48}}^{C_{12}^6} \\ \underbrace{48\ 48\\48\ 48}_{C_{21}^6} & \underbrace{48\ 48\\48\ 48}_{C_{22}^6} \end{bmatrix}}_{i_{21}}; \underbrace{P_1^6}_{i_{10}} = \underbrace{\begin{bmatrix} 96 & 96 \\ 96 & 96 \end{bmatrix}}_{(A_{11}^6+A_{22}^6)\times(B_{11}^6+B_{22}^6)}; \underbrace{P_2^6}_{i_{11}} =$$

$$\underbrace{\begin{bmatrix} 16 & 16 \\ 16 & 16 \end{bmatrix}}_{(A_{21}^6+A_{22}^6)\times B_{11}^6}; \underbrace{P_3^6}_{i_{12}} = \underbrace{\begin{bmatrix} -32 & -32 \\ -32 & -32 \end{bmatrix}}_{A_{11}^6\times(B_{12}^6-B_{22}^6)}; \underbrace{P_4^6}_{i_{13}} = \underbrace{\begin{bmatrix} 32 & 32 \\ 32 & 32 \end{bmatrix}}_{A_{22}^6\times(B_{21}^6-B_{11}^6)}; \underbrace{P_5^6}_{i_{14}} = \underbrace{\begin{bmatrix} 80 & 80 \\ 80 & 80 \end{bmatrix}}_{(A_{11}^6+A_{12}^6)\times B_{22}^6}; \underbrace{P_6^6}_{i_{15}} = \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}_{(A_{21}^6-A_{11}^6)\times(B_{11}^6+B_{12}^6)}; \underbrace{P_7^6}_{i_{16}} =$$

$$\underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}_{(A_{12}^6-A_{22}^6)\times(B_{21}^6+B_{22}^6)}; \underbrace{C_{11}^6}_{i_{17}} = \overbrace{\begin{bmatrix} 48 & 48 \\ 48 & 48 \end{bmatrix}}^{P_1^6+P_4^6-P_5^6+P_7^6}; \underbrace{C_{12}^6}_{i_{18}} = \overbrace{\begin{bmatrix} 48 & 48 \\ 48 & 48 \end{bmatrix}}^{P_3^6+P_5^6}; \underbrace{C_{21}^6}_{i_{19}} = \overbrace{\begin{bmatrix} 48 & 48 \\ 48 & 48 \end{bmatrix}}^{P_2^6+P_4^6}; \underbrace{C_{22}^6}_{i_{20}} = \overbrace{\begin{bmatrix} 48 & 48 \\ 48 & 48 \end{bmatrix}}^{P_1^6+P_3^6-P_2^6+P_6^6}.$$

$$\underbrace{P_7}_{i_{16}} = \underbrace{\begin{bmatrix} \overbrace{\underbrace{A_{11}^7}_{i_4}}^{i_2} & \overbrace{\underbrace{A_{12}^7}_{i_5}}^{i_3} \\ \underbrace{A_{21}^7}_{i_4} & \underbrace{A_{22}^7}_{i_5} \end{bmatrix}}_{(A_{21}-A_{11})\times(B_{11}+B_{12})} \times \underbrace{\begin{bmatrix} \overbrace{\underbrace{B_{11}^7}_{i_8}}^{i_6} & \overbrace{\underbrace{B_{12}^7}_{i_9}}^{i_7} \\ \underbrace{B_{21}^7}_{i_8} & \underbrace{B_{22}^7}_{i_9} \end{bmatrix}}_{(A_{21}-A_{11})\times(B_{11}+B_{12})} = \underbrace{\begin{bmatrix} \overbrace{\underbrace{-176\ -176\\-176\ -176}}^{C_{11}^7} & \overbrace{\underbrace{-176\ -176\\-176\ -176}}^{C_{12}^7} \\ \underbrace{-176\ -176\\-176\ -176}_{C_{21}^7} & \underbrace{-176\ -176\\-176\ -176}_{C_{22}^7} \end{bmatrix}}_{i_{21}};$$

$$\underbrace{P_1^7}_{i_{10}} = \underbrace{\begin{bmatrix} -352 & -352 \\ -352 & -352 \end{bmatrix}}_{(A_{11}^7+A_{22}^7)\times(B_{11}^7+B_{22}^7)}; \underbrace{P_2^7}_{i_{11}} = \underbrace{\begin{bmatrix} -144 & -144 \\ -144 & -144 \end{bmatrix}}_{(A_{21}^7+A_{22}^7)\times B_{11}^7}; \underbrace{P_3^7}_{i_{12}} = \underbrace{\begin{bmatrix} 32 & 32 \\ 32 & 32 \end{bmatrix}}_{A_{11}^7\times(B_{12}^7-B_{22}^7)}; \underbrace{P_4^7}_{i_{13}} = \underbrace{\begin{bmatrix} -32 & -32 \\ -32 & -32 \end{bmatrix}}_{A_{22}^7\times(B_{21}^7-B_{11}^7)}; \underbrace{P_5^7}_{i_{14}} = \underbrace{\begin{bmatrix} -208 & -208 \\ -208 & -208 \end{bmatrix}}_{(A_{11}^7+A_{12}^7)\times B_{22}^7};$$

$$\underbrace{P_6^7}_{i_{15}} = \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}_{(A_{21}^7-A_{11}^7)\times(B_{11}^7+B_{12}^7)}; \underbrace{P_7^7}_{i_{16}} = \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}}_{(A_{12}^7-A_{22}^7)\times(B_{21}^7+B_{22}^7)}; \underbrace{C_{11}^7}_{i_{17}} = \overbrace{\begin{bmatrix} -176 & -176 \\ -176 & -176 \end{bmatrix}}^{P_1^7+P_4^7-P_5^7+P_7^7}; \underbrace{C_{12}^7}_{i_{18}} = \overbrace{\begin{bmatrix} -176 & -176 \\ -176 & -176 \end{bmatrix}}^{P_3^7+P_5^7}; \underbrace{C_{21}^7}_{i_{19}} =$$

$$\overbrace{\begin{bmatrix} -176 & -176 \\ -176 & -176 \end{bmatrix}}^{P_2^7+P_4^7}; \underbrace{C_{22}^7}_{i_{20}} = \overbrace{\begin{bmatrix} -176 & -176 \\ -176 & -176 \end{bmatrix}}^{P_1^7+P_3^7-P_2^7+P_6^7}.$$

$$\underbrace{C_{11}}_{i_{17}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 28 & 28 & 28 & 28 \\ 56 & 56 & 56 & 56 \\ 84 & 84 & 84 & 84 \end{bmatrix}; \underbrace{C_{12}}_{i_{18}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 28 & 28 & 28 & 28 \\ 56 & 56 & 56 & 56 \\ 84 & 84 & 84 & 84 \end{bmatrix}; \underbrace{C_{21}}_{i_{19}} = \begin{bmatrix} 112 & 112 & 112 & 112 \\ 140 & 140 & 140 & 140 \\ 168 & 168 & 168 & 168 \\ 196 & 196 & 196 & 196 \end{bmatrix}; \underbrace{C_{22}}_{i_{20}} = \begin{bmatrix} 112 & 112 & 112 & 112 \\ 140 & 140 & 140 & 140 \\ 168 & 168 & 168 & 168 \\ 196 & 196 & 196 & 196 \end{bmatrix}.$$

$$\underbrace{C = A \times B = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}}_{i_{21}} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 28 & 28 & 28 & 28 & 28 & 28 & 28 & 28 \\ 56 & 56 & 56 & 56 & 56 & 56 & 56 & 56 \\ 84 & 84 & 84 & 84 & 84 & 84 & 84 & 84 \\ 112 & 112 & 112 & 112 & 112 & 112 & 112 & 112 \\ 140 & 140 & 140 & 140 & 140 & 140 & 140 & 140 \\ 168 & 168 & 168 & 168 & 168 & 168 & 168 & 168 \\ 196 & 196 & 196 & 196 & 196 & 196 & 196 & 196 \end{bmatrix}.$$

Seja $M(n)$ o número de multiplicações utilizadas por Prodmat, para calcular o produto de duas matrizes quadradas de ordem $n = 2^k$ $(k = log_2 n)$. Se $k = 1$ então ocorrerão 7 multiplicações, pois chama-se diretamente Prodmat2. Caso $k \geqslant 2$, Prodmat será chamado recursivamente para multiplicar matrizes quadradas de ordem $\frac{n}{2}$, sete vezes, mais 7 multiplicações de Prodmat2, para cada ordem, sendo o número total de multiplicações, nesta etapa, igual a $7 \cdot M\left(\frac{n}{2}\right) = 7 \cdot M\left(\frac{2^k}{2}\right) = 7 \cdot M(2^{k-1})$. Isso fornece a seguinte recorrência

$$M\left(2^k\right) = \begin{cases} 7, & se\ k = 1 \\ \underbrace{7}_{\substack{Chamadas\ recursivas \\ P_1\ até\ P_7}} \times \underbrace{M\left(2^{k-1}\right)}_{\substack{Custo\ da\ chamada\ recursiva \\ para\ uma\ multiplicação}}, & se\ k > 1 \end{cases}.$$

Assim,

$$\begin{array}{rcccccc} M\left(2^k\right) & = & 7M\left(2^{k-1}\right) & = & 7^2 M\left(2^{k-2}\right) & = & 7^3 M\left(2^{k-3}\right) \\ & = & \cdots & = & 7^{k-1} M\left(2^{k-(k-1)}\right) & = & 7^{k-1} M\left(2^1\right) \\ & = & 7^{k-1} \cdot 7 & = & 7^k & = & 7^{log_2 n} \\ & = & n^{log_2 7} & & & & \end{array}$$

Veja que o número de multiplicações foi reduzido de n^3 para $n^{log_2 7}$, que é uma economia considerável, para n suficientemente grande. Mesmo havendo aumento no número de adições algébricas, Prodmat realizará menos operações de adição algébrica do que o algoritmo clássico faz, para n suficientemente grande. De fato, seja $A(n)$ o número de adições algébricas que Prodmat realiza para matrizes quadradas de ordem $n = 2^k$. Se $n = 2$ então ocorrerão 18 adições algébricas por meio de Prodmat2. Caso $k > 1$, Prodmat será chamado recursivamente sete vezes com matrizes quadradas de ordem $\frac{n}{2}$, para P_1, P_2, ..., P_6 e P_7, sendo $A\left(\frac{n}{2}\right) = A\left(\frac{2^k}{2}\right) = A\left(2^{k-1}\right)$ o número total de adições algébricas de matrizes ocorridas nessa chamada. Além disso, ocorrem ainda $\left(\frac{n}{2}\right)^2 = \left(\frac{2^k}{2}\right)^2 = \left(2^{k-1}\right)^2$ adições algébricas dos elementos das matrizes A e B, do algoritmo "Adição Algébrica de Matrizes", o qual é usado 18 vezes para cada chamada do prodmat. Isso fornece a seguinte recorrência

$$A(2^k) = \begin{cases} 18, & se\ k = 1 \\ \underbrace{7}_{\substack{Chamadas\ recursivas \\ P_1\ até\ P_7}} \times \underbrace{A\left(2^{k-1}\right)}_{\substack{Custo\ de\ Prodmat \\ com\ adições\ algébricas \\ implícitas\ para \\ a\ produção\ de\ (P_i)}} + \underbrace{18}_{\substack{18\ chamadas \\ para\ adição \\ algébrica}} \times \underbrace{\left(2^{k-1}\right)^2}_{\substack{Custo\ de\ uma \\ adição\ explícita \\ de\ matrizes \\ de\ ordem\ \frac{n}{2}}}, & se\ k > 1 \end{cases}.$$

Assim,

$$
\begin{aligned}
& A\left(2^k\right) = \\
= \;& 7 \cdot A\left(2^{k-1}\right) + 18 \cdot \left(2^{k-1}\right)^2 \\
= \;& 7^2 \cdot A\left(2^{k-2}\right) + 7 \cdot 18 \cdot \left(2^{k-2}\right)^2 + 18 \cdot \left(2^{k-1}\right)^2 \\
= \;& 7^3 \cdot A\left(2^{k-3}\right) + 7^2 \cdot 18 \cdot \left(2^{k-3}\right)^2 + 7 \cdot 18 \cdot \left(2^{k-2}\right)^2 + 18 \cdot \left(2^{k-1}\right)^2 \\
= \;& 7^4 \cdot A\left(2^{k-4}\right) + 7^3 \cdot 18 \cdot \left(2^{k-4}\right)^2 + 7^2 \cdot 18 \cdot \left(2^{k-3}\right)^2 + 7 \cdot 18 \cdot \left(2^{k-2}\right)^2 + 18 \cdot \left(2^{k-1}\right)^2 \\
& \vdots \\
= \;& 7^{k-1} \cdot A\left(2^{k-(k-1)}\right) + 7^{k-2} \cdot 18 \cdot \left(2^{k-(k-1)}\right)^2 + ... + 7^3 \cdot 18 \cdot \left(2^{k-4}\right) + \\
& +7^2 \cdot 18 \cdot \left(2^{k-3}\right)^2 + 7 \cdot 18 \cdot \left(2^{k-2}\right)^2 + 18 \cdot \left(2^{k-1}\right)^2 \\
= \;& 7^{k-1} \cdot A\left(2^1\right) + 7^{k-2} \cdot 18 \cdot \left(2^1\right)^2 + ... + 7^3 \cdot 18 \cdot \left(2^{k-4}\right) + 7^2 \cdot 18 \cdot \left(2^{k-3}\right)^2 + \\
& +7 \cdot 18 \cdot \left(2^{k-2}\right)^2 + 18 \cdot \left(2^{k-1}\right)^2 \\
= \;& 7^{k-1} \cdot 18 + 7^{k-2} \cdot 18 \cdot \left(2^1\right)^2 + ... + 7^3 \cdot 18 \cdot \left(2^{k-4}\right) + 7^2 \cdot 18 \cdot \left(2^{k-3}\right)^2 + \\
& +7 \cdot 18 \cdot \left(2^{k-2}\right)^2 + 18 \cdot \left(2^{k-1}\right)^2 \\
= \;& 18 \cdot \left[\underbrace{7^{k-1} + 7^{k-2} \cdot \left(2^1\right)^2 + ... + 7^3 \cdot \left(2^{k-4}\right)^2 + 7^2 \cdot \left(2^{k-3}\right)^2 + 7 \cdot \left(2^{k-2}\right)^2 + \left(2^{k-1}\right)^2}_{\textit{Soma dos termos da PG de razão } \frac{4}{7}} \right] \\
= \;& 18 \cdot 7^{k-1} \frac{\left(\frac{4}{7}\right)^k - 1}{\frac{4}{7} - 1} = 6 \cdot \left(7^k - 4^k\right).
\end{aligned}
$$

Portanto, $A(n) = 6 \cdot \left(7^k - 4^k\right) < 6 \cdot 7^k = 6 \cdot 7^{log_2 n} = 6 \cdot n^{log_2 7} < n^3 - n^2$, para n suficientemente grande.

Assim, $M(n) + A(n) < n^{log_2 7} + 6 \cdot n^{log_2 7} < n^3 + (n^3 - n^2) = 2n^3 - n^2$, sendo $M(n) + A(n)$ e $2n^3 - n^2$ os respectivos totais de multiplicações e adições algébricas dos algoritmos Prodmat e clássico.

Suponha agora que n não seja uma potência de base 2. Então, existe $k \in \mathbb{N}$ tal que $2^{k-1} < n < 2^k$. Para poder aplicar o Prodmat na multiplicação de matrizes quadradas com essa ordem n, altera-se a ordem dessa matriz de n para 2^k, preenchendo com zeros as $2^k - n$ linhas e colunas extras criadas, segundo o esquema abaixo:

$$
A'_{2^k \times 2^k} = \left[\begin{array}{ccc|ccccc} & & & 0 & 0 & \cdots & 0 \\ & A_{n\times n} & & 0 & 0 & \cdots & 0 \\ & & & \cdots & \cdots & \cdots & \cdots \\ \hline 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots \end{array}\right] \text{ e } B'_{2^k \times 2^k} = \left[\begin{array}{ccc|ccccc} & & & 0 & 0 & \cdots & 0 \\ & B_{n\times n} & & 0 & 0 & \cdots & 0 \\ & & & \cdots & \cdots & \cdots & \cdots \\ \hline 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots \end{array}\right].
$$

Assim, $C' = A' \times B'$. Após a multiplicação, as posições acrescentadas estarão todas preenchidas com zero. Então, para se obter $C = A \times B$, basta eliminar tais posições.

Exemplo 41. Dadas as matrizes $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}_{3\times 3}$ e $B = \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \end{bmatrix}_{3\times 3}$, tem-se $C = A \times B = \begin{bmatrix} 6 & 6 & 6 \\ 6 & 6 & 6 \\ 6 & 6 & 6 \end{bmatrix}_{3\times 3}$. Mas $2^1 < 3 < 2^2$ e conforme o esquema acima, tem-se:

$$A' = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{2^2 \times 2^2} \text{ e } B' = \begin{bmatrix} 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 2 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{2^2 \times 2^2} \Rightarrow$$

$$C' = A' \times B' = \begin{bmatrix} 6 & 6 & 6 & 0 \\ 6 & 6 & 6 & 0 \\ 6 & 6 & 6 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{2^2 \times 2^2} .$$

Removendo-se a quarta linha e a quarta coluna de C', obtém-se C.

Capítulo 4

Exemplos Clássicos de Algoritmos Computacionais

4.1 Localização de Item em uma Lista

Imagine um arquivo de uma escola contendo fichas para cada aluno, identificadas pelos nomes e dispostas em ordem alfabética de sobrenomes, nas quais são registradas informações. Cada ficha contém o número de matrícula do aluno. Considere a situação de ter que localizar uma determinada ficha conhecendo apenas o número de matrícula do aluno, para registrar tais informações. Supondo haver n fichas na gaveta, o número mínimo de fichas verificadas será igual a 1 se o número procurado for o da primeira ficha. Caso tal número encontra-se na última ficha, serão realizadas n comparações no máximo.

Se a busca de uma determinada ficha começa pela comparação com a primeira ficha da gaveta, passando para a sucessora, até encontrá-la, então a primeira ficha exigirá uma comparação de localização, a segunda duas, a terceira três, e assim sucessivamente. O número total de comparações, para todos os alunos, será $1+2+3+...+k+...+n = \frac{n(n+1)}{2}$.

Se são gastos dois segundos para fazer uma comparação e um minuto para registrar todas as informações em uma ficha de arquivo, para que valor de n será gasto mais tempo fazendo comparações do que gravando novas informações? O tempo total necessário para todas as comparações será $\frac{n(n+1)}{2}.2 = n(n+1)$ segundos. O tempo total exigido para gravar as informações em todas as fichas será de $60n$ segundos. Assim, gasta-se mais tempo com a localização da ficha do que com a gravação se $n(n+1) > 60n$, isto é, $n > 59$.

Dada uma sequência de números $A = (a_1,\ a_2, ...,\ a_n)$, pode-se localizar a posição do número S em A, isto é, encontrar um índice i tal que $a_i = S$, se tal i existir, por meio do seguinte algoritmo.

Algoritmo Localizar em Sequência

i_1: Entrada: n, os elementos de A e S.
i_2: **Para** $i = 1$ **até** n, **faça**
    i_3: **Se** $a_i = S$ **então** retorne i e pare.
i_4: **Retorne:** "S não é um elemento de A" e pare.

Contando as comparações no passo i_3, verifica-se que o pior caso irá ocorrer tanto se S não for elemento de A ou se $S = a_n$, requerendo, em ambos os casos, n comparações. Assim, o custo desse algoritmo é $O(n)$.

Se os números da sequência A estiverem dispostos na ordem crescente,

$$a_1 < a_2 < ... < a_n,$$

pode-se criar um algoritmo de localização eficiente, o qual funcionará conforme o seguinte exemplo:

Exemplo 42. Na localização do número 18, na sequência $A = (2, 4, 6, 8, 10, 12, 14, 16, 18)$, ocorrerão os seguintes passos:

- ♦ A sequência A é dividida em duas sequências $(2, 4, 6, 8, 10)$ e $(12, 14, 16, 18)$.
- ♦ Compara-se 18 com o primeiro número da sequência $(12, 14, 16, 18)$. Como $18 > 12$, divide-se tal sequência em duas sequências $(12, 14)$ e $(16, 18)$.
- ♦ Compara-se 18 com o primeiro número da sequência $(16, 18)$. Como $18 > 16$, divide-se $(16, 18)$ em duas sequências (16) e (18).
- ♦ Compara-se 18 com o número da sequência (18). Como são iguais, é retornada a posição 9, na qual está o número procurado.

■

Logo abaixo está o algoritmo para esse procedimento.

Localização Eficiente em Sequência Ordenada

- Entrada: O número procurado x e os números da sequência $A = (a_1,\ a_2, ..., a_n)$, sendo $a_1 < a_2 < ... < a_n$.
- Saída: A posição $i \in \{1, 2, ..., n\}$ de x na sequência ou x não está na sequência.
- Passo 1: Considere $i = 1$, representando a posição inicial na sequência A.
- Passo 2: Considere $j = n$, representando a posição final na sequência A.
  Observação 13. Os números i e j serão utilizados para se obter a posição $\left\lfloor \frac{i+j}{2} \right\rfloor$, responsável por dividir a sequência em duas subsequências de mesmo comprimento ou, se não for possível, havendo apenas um elemento a mais em uma das duas.
- Passo 3: Enquanto $i < j$ faça $m := \left\lfloor \frac{i+j}{2} \right\rfloor$.
  Observação 14. A condição $i < j$ se verdadeira, dirá que a sequência analisada até o momento, $\left(a_i,\ a_{i+1}, ...,\ a_{\left\lfloor \frac{i+j}{2} \right\rfloor},\ a_{\left\lfloor \frac{i+j}{2} \right\rfloor+1}, ...,\ a_j\right)$, não é unitária. Daí, como os números da sequência estão dispostos em ordem crescente, compara-se x com o termo $a_m = a_{\left\lfloor \frac{i+j}{2} \right\rfloor}$ dessa sequência. Se x pertence à sequência A e $x > a_m$ então x estará na subsequência $C = \left(a_{\left\lfloor \frac{i+j}{2} \right\rfloor+1},\ a_{\left\lfloor \frac{i+j}{2} \right\rfloor+2}, ...,\ a_j\right)$, não sendo necessário o desperdício de tempo fazendo buscas na subsequência $B = \left(a_i,\ a_{i+1}, ...,\ a_{\left\lfloor \frac{i+j}{2} \right\rfloor}\right)$. Agora se $x < a_m$ então x estará em $B = \left(a_i,\ a_{i+1}, ...,\ a_{\left\lfloor \frac{i+j}{2} \right\rfloor}\right)$, bastando efetuar buscas somente nesta subsequência.

- Passo 4: Se $x > a_m$ então faça i assumir o valor numérico da expressão algébrica $m+1$, isto é, $i := m+1$.

  Observação 15. O passo 4 será executado sempre dentro do passo 3 (*loop*). Como x não está em B, a busca será realizada em C. O *loop* enquanto será executado novamente de modo a dividir C, se não for unitária, em duas novas subsequências, $B = \left(a_i,\ a_{i+1}, ...,\ a_{\lfloor \frac{i+j}{2} \rfloor}\right)$ e $C = \left(a_{\lfloor \frac{i+j}{2} \rfloor + 1},\ a_{\lfloor \frac{i+j}{2} \rfloor + 2}, ...,\ a_j\right)$, para buscas. Veja que o nome da subsequência C é mantido, sendo a nova subsequência C uma subsequência da antiga subsequência C, mas com menor tamanho, pois o índice i foi aumentado, consequentemente o índice $\left\lfloor \frac{i+j}{2} \right\rfloor + 1$ também é aumentado.

- Passo 5: Senão faça $j := m$.

  Observação 16. O passo 5 também será executado sempre dentro do passo 3. Neste caso, como x não está em C, a busca será realizada em B. Novamente o *loop* enquanto será executado e a condição $i < j$ afirmará que B tem pelo menos dois elementos. Se a afirmação for verdadeira, B será dividida em duas novas subsequências, $B = \left(a_i,\ a_{i+1}, ...,\ a_{\lfloor \frac{i+j}{2} \rfloor}\right)$ e $C = \left(a_{\lfloor \frac{i+j}{2} \rfloor + 1},\ a_{\lfloor \frac{i+j}{2} \rfloor + 2}, ...,\ a_j\right)$, para buscas. Aqui também é mantido o nome da subsequência B, sendo a nova subsequência B uma subsequência da antiga subsequência B, mas com menor tamanho, pois o índice j foi diminuído, consequentemente o índice $\left\lfloor \frac{i+j}{2} \right\rfloor$ também é diminuído.

  O laço (*loop*) enquanto provocará divisões nas subsequências até obter uma subsequência unitária $(a_i)_{i \in \{1,2,...,n\}}$, na qual ocorrerá a busca final.

- Passo 6: Se $x = a_i$ retorne: "x está na posição i. Caso contrário, retorne: "x não está na sequência A".

- Passo 7: Pare.

Exemplo 43. Aplicando os passos logo acima na sequência $A = (\overset{a_1}{1}, \overset{a_2}{2}, \overset{a_3}{2}, \overset{a_4}{3}, \overset{a_5}{5}, \overset{a_6}{6}, \overset{a_7}{8})$, para verificar se 9 está em A, tem-se:

- Passo 1: $i := 1$.

- Passo 2: $j := 7$.

- Passo 3: Dos passos 1 e 2, as posições i e j de A são tais que $i < j$. Portanto A não é uma sequência unitária e $m := \left\lfloor \frac{i+j}{2} \right\rfloor = \left\lfloor \frac{1+7}{2} \right\rfloor = \lfloor 4 \rfloor = 4$. Execute o passo 4, para dividir A em duas subsequências.

  - Passo 4: $x = 9$ e $a_m = a_4 = 3$, com $x > a_m$ $(9 > 3)$. Então, i assume o valor numérico da expressão algébrica $m+1$, para $m = 4$ $(m + 1 = 4 + 1 = 5)$, ou seja, $i := 5$.

    Observação 17. Até aqui, obtém-se as subsequências $B = (\overset{a_1}{1}, \overset{a_2}{2}, \overset{a_3}{2}, \overset{a_4}{3})$ e $C = (\overset{a_5}{5}, \overset{a_6}{6}, \overset{a_7}{8})$. A busca de $x = 9$ será efetuada na subsequência $C = (\overset{a_5}{5}, \overset{a_6}{6}, \overset{a_7}{8})$, pois $x > a_4$. Deve-se executar novamente o passo 3, para comparar i com j e verificar se C tem pelo menos dois elementos.

- Passo 3: Dos passos 2 e 4, anteriores, as posições i e j de C são tais que $i < j$. Portanto C não é unitária e $m := \left\lfloor \frac{i+j}{2} \right\rfloor = \left\lfloor \frac{5+7}{2} \right\rfloor = \lfloor 6 \rfloor = 6$. Execute o passo 4, para dividir C em duas subsequências.

  - Passo 4: $a_m = a_6 = 6$ e $x > a_m$ $(9 > 6)$. Agora, i assume o valor numérico da expressão algébrica $m + 1$, para $m = 6$ $(m + 1 = 6 + 1 = 7)$, ou seja, $i := 7$.

    Observação 18. Até aqui, obtém-se as subsequências $B = (\overset{a_5}{5}, \overset{a_6}{6})$ e $C = (\overset{a_7}{8})$. A busca de $x = 9$ será efetuada em $C = (\overset{a_7}{8})$, pois $x > a_6$. Deve-se executar novamente o passo 3, para comparar i com j e verificar se C é unitária.

- Passo 3: Do passo 4, imediatamente anterior a este passo, e do passo 2, tem-se $i = 7$ e $j = 7$. Já que a condição $i < j$ assumiu valor falso $(i \not< j)$, a sequência C, da observação 13, é unitária. Então o passo 3 não será mais executado. Execute o passo 6, para fazer a comparação final.

- Passo 6: Como $x = 9$ e $a_i = a_7 = 8$ são diferentes, x não está na sequência A.

- Passo 7: Nenhum passo do algoritmo será executado novamente, pois o procedimento acaba aqui.

Veja abaixo o pseudocódigo desse algoritmo:

Algoritmo Localizar item em Sequência Ordenada (n, $a_1 < a_2 < ... < a_n$, S).

```
1 -> (n, a_1 < a_2 < ... < a_n, S) é a entrada, i é
2 o índice do primeiro termo da lista e j é o índi-
3 ce do último.
4 i := 1
5 j := n
6 Enquanto i < j, faça
7       m := ⌊(i+j)/2⌋
8       Se x > a_m, então
9             i := m + 1
10      Senão
11            j := m
12      FimSe
13 FimEnquanto
14 Se x = a_i, então
15      localização := i
16 Senão
17      localização := 0
18 FimSe
19 FimProcedimento
```

Deseja-se calcular o custo desse algoritmo quando $n \mapsto \infty$. Para facilitar tal cálculo, basta considerar $n = 2^k$, pois a subsequência $\left(2^k\right)_{k \in \mathbb{N}}$ é tal que $k \mapsto \infty \Rightarrow 2^k \mapsto \infty$. Assim, a quantidade de elementos da sequência abaixo representará a quantidade de vezes em que o *loop* "**Enquanto**" será executado ($k+1$ vezes), já que a função desse *loop* é sempre trabalhar com uma "metade" da sequência anterior. O primeiro elemento dessa sequência $\left(2^k\right)$ representa o tamanho da lista original, e cada elemento posterior a ele é o tamanho de cada uma das sublistas menores obtidas, nas quais se procura o x.

$$(2^k,\ 2^{k-1}, ..., 2, 1).$$

Tem-se que $n = 2^k \Leftrightarrow k = log_2 n$, logo
$k + 1 = log_2 n + 1 = O(log_2 n) + O(1) = O(log_2 n)$. Portanto, o custo desse algoritmo é $O(log_2 n)$.

4.2 Ordenação dos Elementos (Números) de um Vetor

Nesta seção, objetiva-se organizar, em ordem crescente, os elementos de um vetor $A = [a_1,\ a_2, ...,\ a_n]$. Assim, serão apresentados alguns algoritmos que solucionam este mesmo problema e, para cada um deles, será feita a análise de eficiência, no pior caso.

Será efetuado o teste de mesa de alguns dos algoritmos de ordenação, para que se possa compreender melhor tais algoritmos.

4.2.1 Ordenação Bolha

A *ordenação bolh*a, também conhecida como *bubble-sort* ou *ordenação por flutuação*, trata-se de um algoritmo que apresenta um procedimento de ordenação por permutação de dois números de posições consecutivas do vetor $A = [\overset{A[1]}{a_1},\ \overset{A[2]}{a_2},\ \overset{A[3]}{a_3},\ \overset{A[4]}{a_4}, ...,\ \overset{A[n]}{a_n}]$, começando pelas posições $A[1]$ e $A[2]$, depois por $A[2]$ e $A[3]$, e assim sucessivamente até que o maior número ocupe a posição $A[n]$. Esse procedimento é novamente executado no vetor alterado $[\overset{A[1]}{a_1},\ \overset{A[2]}{a_2},\ \overset{A[3]}{a_3},\ \overset{A[4]}{a_4}, ...,\ \overset{A[n-1]}{a_{n-1}}]$, até que o maior número ocupe a posição $A[n-1]$, e assim sucessivamente de modo que todos os números de A estejam dispostos em ordem crescente.

Exemplo 44. Na ordenação do vetor $A = [\overset{A[1]}{9},\ \overset{A[2]}{5},\ \overset{A[3]}{6},\ \overset{A[4]}{2}]$, o algoritmo de ordenação bolha fará o seguinte:

1. Compara 9 com 5 e ordena-os, permutando-os.
2. Em $[\overset{A[1]}{5},\ \overset{A[2]}{9},\ \overset{A[3]}{6},\ \overset{A[4]}{2}]$, compara 9 com 6 e permuta-os.
3. Em $[\overset{A[1]}{5},\ \overset{A[2]}{6},\ \overset{A[3]}{9},\ \overset{A[4]}{2}]$, compara 9 com 2 e permuta-os.
4. Em $[\overset{A[1]}{5},\ \overset{A[2]}{6},\ \overset{A[3]}{2}]$, compara 5 com 6, não permutando-os, pois já estão ordenados.
5. Depois compara 6 com 2, permutando-os.
6. Em $[\overset{A[1]}{5},\ \overset{A[2]}{2}]$, compara 5 com 2, permutando-os.
7. Retorna o vetor $A = [\overset{A[1]}{2},\ \overset{A[2]}{5},\ \overset{A[3]}{6},\ \overset{A[4]}{9}]$.

Ordenação Bolha

- Entrada: A quantidade n de números do vetor A e seus elementos $a_1,\ a_2,\ ...,\ a_n$.
- Saída: Os elementos de A dispostos em ordem crescente.
- Passo 1: Comece considerando $i = 1$.
  - Passo 2: Comece considerando $j = 1$.
    - Passo 3: Compare os termos a_j e a_{j+1}. Se $a_j > a_{j+1}$ então permute-os, para dispô-los em ordem crescente.
    - Passo 4: Incremente j. Se j assumir valor $n - i + 1$ então pare. Mas se $j < n - i + 1$ então volte a executar a partir do passo 3.

Observação 19. A condição de parada de execução dos passos 2 e 3, $j = n - i + 1$, se verdadeira, indica que todos os números do vetor A, usado para cada i, foram percorridos.

A execução completa dos passos 3 e 4, iniciando com $j = 1$, que vai sendo incrementado gradativamente, até que se pare a execução deles para $j = n-i+1$, só garante que o maior número de A ocupará a sua última posição, sem poder saber se o novo vetor obtido já está totalmente ordenado. Então outra análise deve ser feita, mas no vetor obtido eliminando a última posição, $A[n - i + 1]$, ficando esta e seu número reservados para compor a saída do algoritmo.

- Passo 5: Incremente i. Se i assumir valor n então pare. Mas se $i < n$ então volte a executar a partir do passo 2.

  Observação 20. i é incrementado para que seja possível fazer a análise do vetor A. A condição de parada de execução do passo 5, $i = n$, se verdadeira, indica que A, cuja ordenação de seus números deveria ser verificada a partir do passo 2, já está ordenado, podendo-se executar o passo 6.

- Passo 6: Retorne o vetor A, com seus n números já dispostos em ordem crescente.

Exemplo 45. Utilizando os passos acima, da ordenação bolha, para ordenar os elementos do vetor $A = [\overset{A_1}{8}, \overset{A_2}{4}, \overset{A_3}{7}, \overset{A_4}{1}]$, tem-se o esquema abaixo:

	A[1]	A[2]	A[3]	A[4]			A[1]	A[2]	A[3]	A[4]			A[1]	A[2]	A[3]	A[4]			
(a)	8	4	7	1		(d)	4	7	1	8		(g)	1	4	7	8	(Saída)		
Permuta						Não Permuta													
	A[1]	A[2]	A[3]	A[4]			A[1]	A[2]	A[3]	A[4]									
(b)	4	8	7	1		(e)	4	7	1	8									
Permuta						Permuta													
	A[1]	A[2]	A[3]	A[4]			A[1]	A[2]	A[3]	A[4]									
(c)	4	7	8	1		(f)	4	1	7	8									
Permuta						Permuta													

■

Logo abaixo, segue a versão desse algoritmo escrita por pseudocódigo.

Procedimento Ordenação Bolha

```
1  Entrada: n e a_1, a_2, ..., a_n.
2  Para i = 1 até n − 1, faça
3    Para j = 1 até n − i, faça
4        Se (A[j] > A[j + 1]), então
5            Temp := A[j];
6            A[j] := A[j + 1];
7            A[j + 1] := Temp;
8        FimSe
9    FimPara
10 FimPara
11 Para i = 1 até n, faça
12   Retorne: "O vetor na forma ordenada é:"
13   Retorne: a[i] = A[i]
14 FimPara
15 FimProcedimento
```

Observação 21. Na execução desse algoritmo, feita pelo computador, haverá gravação e regravação de número em determinada posição. Para não se perder o número que estava armazenado nesta posição, antes da regravação, utiliza-se a variável auxiliar *Temp*. Na expressão $A[j] > A[j + 1]$, deseja-se verificar se o número da posição $A[j]$ é maior do que o

número da posição $A[j+1]$. Na expressão $Temp := A[i]$, é atribuído à variável $Temp$ o número da posição $A[j+1]$. Na expressão $A[j] := A[j+1]$, será colocado o número da posição $A[j+1]$ na posição $A[j]$. Agora, na expressão $A[j+1] := Temp$, coloca-se o número representado pela variável auxiliar $Temp$ na posição $A[j+1]$.

4.2.1.1 Custo da Ordenação Bolha

Em qualquer caso, o *loop* interno "**Para**", desse algoritmo, será executado $n-i$ vezes, para cada $i \in \{1, 2, ..., (n-1)\}$ do *loop* externo "**Para**". Assim,

$$\underbrace{\sum_{i=1}^{n-1} \underbrace{(n-i)}_{Polinômio\ de\ grau\ 1}}_{Polinômio\ de\ grau\ 2\ em\ n-1\ (Corolário\ 2)} = \underbrace{b_2(n-1)^2 + b_1(n-1) + b_0 = O(n^2)}_{Exemplo\ 25}.$$

4.2.2 Ordenação Por Seleção

Dado um vetor $A = [\overset{A[1]}{a_1}, \overset{A[2]}{a_2}, \overset{A[3]}{a_3}, \overset{A[4]}{a_4}, ..., \overset{A[n]}{a_n}]$, para dispor seus números em ordem crescente, o algoritmo de ordenação por seleção, também conhecido como *selection-sort*, irá localizar o menor número a_i, $i \in \{1, 2, 3, ..., n\}$, de A e colocará a_i na primeira posição $A[1]$, permutando (trocando) os números das posições $A[i]$ e $A[1]$, desde que a posição $A[1]$ já não esteja ocupada pelo menor número. No novo vetor, representado pelas posições restantes: $A[2]$, $A[3]$,..., $A[n]$, nenhuma delas ocupadas por a_i, procurará o menor número a_j, $j \in \{1, 2, 3, ..., n\}$, $j \neq i$ e colocará a_j na segunda posição $A[2]$, permutando os números das posições $A[j]$ e $A[2]$, desde que a posição $A[2]$ já não esteja ocupada pelo menor número. Esse procedimento vai se repetindo em vetores cada vez menores até que se tenha os números de A ordenados. Veja logo abaixo os passos desse algoritmo.

Exemplo 46. Na ordenação do vetor $A = [\overset{A[1]}{9}, \overset{A[2]}{5}, \overset{A[3]}{6}, \overset{A[4]}{2}]$, o algoritmo de ordenação por seleção fará o seguinte:

1. Compara 9 com 5 e ordena-os, permutando-os.
2. Em $[\overset{A[1]}{5}, \overset{A[2]}{9}, \overset{A[3]}{6}, \overset{A[4]}{2}]$, compara 5 com 6, mas não os permuta pois $5 < 6$.
3. Depois compara 5 com 2 e os permuta, fornecendo o vetor $[\overset{A[1]}{2}, \overset{A[2]}{9}, \overset{A[3]}{6}, \overset{A[4]}{5}]$.
4. Em $A_1 = [\overset{A_1[1]=A[2]}{9}, \overset{A_1[2]=A[3]}{6}, \overset{A_1[3]=A[4]}{5}]$, compara 9 com 6 e permuta-os.
5. Em $A_1 = [\overset{A_1[1]=A[2]}{6}, \overset{A_1[2]=A[3]}{9}, \overset{A_1[3]=A[4]}{5}]$, compara 6 com 5 e os permuta fornecendo o vetor $A_1 = [\overset{A_1[1]=A[2]}{5}, \overset{A_1[2]=A[3]}{9}, \overset{A_1[3]=A[4]}{6}]$.
6. Em $A_2 = [\overset{A_2[1]=A[3]}{9}, \overset{A_2[2]=A[4]}{6}]$, compara 9 com 6 e permuta-os, fornecendo o vetor $A_2 = [\overset{A_2[1]=A[3]}{6}, \overset{A_2[2]=A[4]}{9}]$.
7. Retorna o vetor $A = [\overset{A[1]}{2}, \overset{A[2]}{5}, \overset{A[3]}{6}, \overset{A[4]}{9}]$.

Ordenação por Seleção

- Entrada: O número n de elementos do vetor A e os números $a_1,\ a_2,\ ...,\ a_n$.
- Saída: Os elementos de A dispostos em ordem crescente.
- Passo 1: Considere $i = 1$.

  Observação 22. i representa a posição na qual deseja-se colocar o menor número do vetor atual obtido ou do mais recente.

  - Passo 2: Considere $j = i + 1$.

    Observação 23. j representa a posição dos números da direita, os quais serão comparados, um a um, com o número de $A[i]$.

    * Passo 3: Se $a_i > a_j$ então permute-os nas suas posições.
    * Passo 4: Incremente j e volte a executar a partir do passo 3, parando no caso em que j assume valor $n + 1$.

      Observação 24. Na eminência de executar o passo 5, o menor número de A já estará ocupando a posição correta $A[i]$.

- Passo 5: Incremente i e volte a executar a partir do passo 2, parando no caso em que i assume valor n, para executar o passo 6.

  Observação 25. i é incrementado para que j seja o responsável pela varredura do vetor atual A_{i-1}, composto pelas posições $A[i]$, $A[i+1]$,..., $A[n]$, estando $A[i-1]$, com o seu número, reservados para a saída.

- Passo 6: Retorne o vetor A, com seus n números já dispostos em ordem crescente.

Exemplo 47. Utilizando os passos acima, da ordenação por seleção, para ordenar os elementos do vetor $A = [\overset{A_1}{8}, \overset{A_2}{4}, \overset{A_3}{7}, \overset{A_4}{1}]$, obtém-se o seguinte esquema:

(a) $\overset{A[1]}{\underline{8}}\ \overset{A[2]}{\underline{4}}\ \overset{A[3]}{7}\ \overset{A[4]}{1}$ Permuta

(b) $\overset{A[1]}{\underline{4}}\ \overset{A[2]}{8}\ \overset{A[3]}{\underline{7}}\ \overset{A[4]}{1}$ Não Permuta

(c) $\overset{A[1]}{\underline{4}}\ \overset{A[2]}{8}\ \overset{A[3]}{7}\ \overset{A[4]}{\underline{1}}$ Permuta

(d) $\overset{A[1]}{1}\ \overset{A[2]}{\underline{8}}\ \overset{A[3]}{\underline{7}}\ \overset{A[4]}{4}$ Permuta

(e) $\overset{A[1]}{1}\ \overset{A[2]}{\underline{7}}\ \overset{A[4]}{8}\ \overset{A[3]}{\underline{4}}$ Permuta

(f) $\overset{A[1]}{1}\ \overset{A[2]}{4}\ \overset{A[3]}{\underline{8}}\ \overset{A[4]}{\underline{7}}$ Permuta

(g) $\overset{A[1]}{1}\ \overset{A[2]}{4}\ \overset{A[3]}{7}\ \overset{A[4]}{8}$ (Saída)

Esse algoritmo pode ser descrito por meio de pseudocódigo, conforme o que segue abaixo.

```
1 Procedimento Selection-Sort (n, A)
2 Saída: Os números de A dispostos em
3 ordem crescente.
4 Para i = 1 até n − 1, faça
5   min := i
6   Para j := i + 1, até n, faça
7     Se A[j] < A[min], então
8       min := j
9     FimSe
10  FimPara
11  Temp := A[i];
12  A[i] := A[min];
```

```
13 A[min] := Temp;                    17 na forma ordenada é:"
14 FimPara                            18   Retorne: a[i] = A[i]
15 Para i = 1 até n, faça             19 FimPara
16   Retorne: "O vetor                20 FimProcedimento
```

4.2.2.1 Custo da Ordenação Por Seleção

O algoritmo de ordenação por seleção foi projetado de forma que, no *loop* duplo, o *loop* interno seja executado $n-(i+1)+1 = n-i$ vezes, para cada $i \in \{1, 2, ..., (n-1)\}$ do *loop* externo. Dessa forma, obtém-se a seguinte relação:

$$\underbrace{\sum_{i=1}^{n-1} \underbrace{(n-i)}_{Polinômio\ de\ grau\ 1}}_{Polinômio\ de\ grau\ 2\ em\ n-1\ (Corolário\ 2)} = \underbrace{b_2(n-1)^2 + b_1(n-1) + b_0 = O(n^2)}_{Exemplo\ 25}.$$

4.2.3 Ordenação Por Inserção

Para ordenar o vetor $A = [\overset{A[1]}{a_1}, \overset{A[2]}{a_2}, \overset{A[3]}{a_3}, \overset{A[4]}{a_4}, ..., \overset{A[n]}{a_n}]$, o algoritmo de ordenação por inserção, também conhecido como *insertion-sort*, ordenará primeiramente a_1 e a_2, depois inserirá a_3 na sua posição ordenada com relação aos dois primeiros elementos, já ordenados. Novamente, inserirá a_4 na sua posição ordenada com relação aos três primeiros elementos já ordenados. Este procedimento é repetido quantas vezes for necessário, inserindo a_j na sua posição ordenada com relação aos $j-1$ primeiros elementos, já ordenados, até que todos os elementos do vetor estejam ordenados de forma crescente.

Exemplo 48. Na ordenação do vetor $A = [\overset{A[1]}{9}, \overset{A[2]}{5}, \overset{A[3]}{6}, \overset{A[4]}{2}]$, o algoritmo de ordenação por seleção fará o seguinte:

1. Compara 5 com 9 e ordena-os, permutando-os.
2. Em $A = [\overset{A[1]}{5}, \overset{A[2]}{9}, \overset{A[3]}{6}, \overset{A[4]}{2}]$, compara 6 com 9, pemutando-os.
3. Em $A = [\overset{A[1]}{5}, \overset{A[2]}{6}, \overset{A[3]}{9}, \overset{A[4]}{2}]$, compara 6 com 5, mas não os permuta, pois já estão ordenados.
4. Depois compara 2 com 9, permutando-os.
5. Em $A = [\overset{A[1]}{5}, \overset{A[2]}{6}, \overset{A[3]}{2}, \overset{A[4]}{9}]$, compara 2 com 6, permutando-os.
6. Em $A = [\overset{A[1]}{5}, \overset{A[2]}{2}, \overset{A[3]}{6}, \overset{A[4]}{9}]$, compara 2 com 5, permutando-os.
7. Retorna o vetor $A = [\overset{A[1]}{2}, \overset{A[2]}{5}, \overset{A[3]}{6}, \overset{A[4]}{9}]$.

Ordenação por Inserção

- Entrada: O tamanho n do vetor A e os elementos $a_1,\ a_2,\ ...,\ a_n$ dele.
- Saída: Os elementos de A dispostos em ordem crescente.
- Passo 1: Faça a variável j assumir o valor 2.

Observação 26. j indica a posição em A, também representada por $A[j]$, do número que será inserido em alguma posição da esquerda, ficando ordenado com os números que antes estavam a sua esquerda, já ordenados.

- Passo 2: Faça a variável i assumir o valor numérico da expressão algébrica $j-1$.

  Observação 27. i é uma variável auxiliar, que também indica a posição no vetor, encarregada de fazer o número da posição $A[j]$, comentado na observação anterior, deslocar para a esquerda, por meio de uma permutação com o seu antecessor $A[j-1]$, segundo o passo 3, se necessário for, ou seja, se não estiverem dispostos em ordem crescente, para estabelecer tal ordem.

- Passo 3: Compare o número x, da posição $A[i]$, com o número y, da posição $A[i+1]$ (sucessor), no vetor A atual. Se $x > y$ então permute-os nas posições $A[i]$ e $A[i+1]$.

  Observação 28. Neste passo serão ordenados os números das posições $A[i]$ e $A[i+1]$.

- Passo 4: Decremente i, ou seja, diminua uma unidade de i. Se i for maior do que 0 então volte a executar a partir do passo 3, considerando o novo vetor obtido até o presente momento. Mas se $i = 0$ então pare e execute o passo 5.

  Observação 29. A condição de parada $i = 0$ indica que o número da posição $A[j]$, no vetor, utilizado no passo 2 anterior, já ocupou sua posição correta com relação aos números ordenados que estavam à sua esquerda.

  O próximo passo é reponsável por indicar o sucessor do número de $A[j]$, para que tal sucessor seja ordenado com os números a sua esquerda, já ordenados.

- Passo 5: Incremente j. Se j for menor do que o valor de $n+1$ então volte a executar a partir do passo 2. Mas se j assumir o valor de $n+1$ então pare e execute o passo 6.

- Passo 6: Retorne o vetor A, com seus n números já dispostos em ordem crescente.

Exemplo 49. Utilizando os passos acima, da ordenação por inserção, para ordenar os elementos do vetor $A = [\overset{A_1}{8}, \overset{A_2}{4}, \overset{A_3}{7}, \overset{A_4}{1}]$, obtém-se o esquema abaixo:

	$A[1]$	$A[2]$	$A[3]$	$A[4]$			$A[1]$	$A[2]$	$A[3]$	$A[4]$			$A[1]$	$A[2]$	$A[3]$	$A[4]$	
(a)	8	4	7	1	(d)	4	7	8	1	(g)	1	4	7	8	(Saída)
	Permuta						Permuta										
	$A[1]$	$A[2]$	$A[3]$	$A[4]$			$A[1]$	$A[2]$	$A[3]$	$A[4]$							
(b)	4	8	7	1	(e)	4	7	1	8								
	Permuta						Permuta										
	$A[1]$	$A[2]$	$A[3]$	$A[4]$			$A[1]$	$A[2]$	$A[3]$	$A[4]$							
(c)	4	7	8	1	(f)	4	1	7	8								
	Não Permuta						Permuta										

O algoritmo de ordenação por inserção é escrito por pseudocódigo da seguinte forma:

Procedimento Insertion-Sort $(n,\ A)$
–> $(n,\ A)$ representa a entrada, o valor de n e os números de A.
Para $j = 2$ **até** n, **faça**
    $chave\ := A[j];$
    $i\ := j - 1;$
    Enquanto $i > 0$ **e** $A[i] > chave$, **faça**
        $A[i+1]\ := A[i];$
        $i\ := i - 1;$
        $A[i+1]\ := chave;$
    FimEnquanto
FimPara
FimProcedimento

Observação 30. Na execução desse algoritmo, feita pelo computador, haverá gravação e regravação de número em determinada posição. Para não se perder o número que estava armazenado nesta posição, antes da regravação, utiliza-se a variável auxiliar *chave*.

4.2.3.1 Custo da Ordenação Por Inserção (Pior Caso)

O pior caso para o algoritmo de ordenação por inserção ocorre se os seus elementos formarem uma sequência de números estritamente decrescente, pois aumentará o número de permutações necessárias para ordenar o vetor A. Isso significa que as instruções do *loop* "**Enquanto**" serão executadas um número maior de vezes. Ou seja, na ordenação, por inserção, do vetor $A = [\overset{A[1]}{a_1}, \overset{A[2]}{a_2}, \overset{A[3]}{a_3}, \overset{A[4]}{a_4}, ..., \overset{A[n]}{a_n}]$, tal que $a_1 > a_2 > a_3 > a_4 > ... > a_n$, tem-se o seguinte:
Etapa 1. Ocorre para $j = 2$, fazendo i assumir dois valores: 1 e 0, havendo uma permutação.
Etapa 2. Ocorre para $j = 3$, fazendo i assumir três valores: 2, 1 e 0, havendo duas permutações.

$$\vdots$$

Etapa n − 1. Ocorre para $j = n$, fazendo i assumir n valores: $(n-1)$, $(n-2)$,..., 2, 1 e 0, havendo $n - 1$ permutações.

Na execução completa do algoritmo de ordenação por inserção, o número total de vezes em que i assume um valor é superior ao número total de vezes em que j assume um valor e também é superior ao número total de permutações.

Como o valor de j é a quantidade de valores que i assumirá, em cada etapa, pode se dizer que o total de valores assumidos por i, na execução completa desse algoritmo, para o pior caso, é dado pela seguinte relação:

$$\sum_{j=2}^{n} j = \underbrace{2 + 3 + 4 + ... + n}_{\text{Soma dos termos da PA de razão 1}} = \frac{(2+n)(n-1)}{2} = O(n^2).$$

4.2.4 Ordenação Por Intercalação (Merge)

O algoritmo de ordenação por intercalação, também conhecido como merge, ordena o vetor $C = [\overset{C[1]}{c_1}, \overset{C[2]}{c_2}, ..., \overset{C[n+m]}{c_{n+m}}]$, fazendo a junção de seus dois vetores $A = [\underset{=c_1}{\overset{A[1]}{a_1}}, \underset{=c_2}{\overset{A[2]}{a_2}}, ..., \underset{=c_n}{\overset{A[n]}{a_n}}]$ e $B = [\underset{=c_{n+1}}{\overset{B[1]}{b_1}}, \underset{=c_{n+2}}{\overset{B[2]}{b_2}}, ..., \underset{=c_{n+m}}{\overset{B[m]}{b_m}}]$, ambos já ordenados, formando um novo vetor C, cujos números estarão dispostos em ordem crescente, da seguinte forma: após fornecer o tamanho de cada vetor, bem como os seus números, como entrada, compara-se os primeiros números de cada vetor, a_1 com b_1, sendo o menor deles reservado para a posição $C[1]$ de C. Sem perda de generalidade, suponha que o menor número seja b_1. Então b_1 será colocado na posição $C[1]$ ($C[1] := b_1$). O novo vetor B será $B = [\underset{=c_{n+2}}{\overset{B[1]}{b_2}}, \underset{=c_{n+3}}{\overset{B[2]}{b_3}}, ..., \underset{=c_{n+m}}{\overset{B[m-1]}{b_m}}]$, com a exclusão de b_1, pois ele já foi reservado para primeira posição da saída. Novamente se compara os primeiros números dos vetores $A = [\underset{=c_1}{\overset{A[1]}{a_1}}, \underset{=c_2}{\overset{A[2]}{a_2}}, ..., \underset{=c_n}{\overset{A[n]}{a_n}}]$ e $B = [\underset{=c_{n+2}}{\overset{B[1]}{b_2}}, \underset{=c_{n+3}}{\overset{B[2]}{b_3}}, ..., \underset{=c_{n+m}}{\overset{B[m-1]}{b_m}}]$, a_1 com b_2, sendo o menor deles reservado para a posição $C[2]$. Novamente sem perda de generalidade, suponha que o menor número seja a_1. Então $C[2] := a_1$. Agora o novo vetor A será $A = [\underset{=c_2}{\overset{A[1]}{a_2}}, \underset{=c_3}{\overset{A[2]}{a_3}}, ..., \underset{=c_n}{\overset{A[n-1]}{a_n}}]$, com a exclusão de a_1, pois ele já foi reservado para a saída. Esse procedimento é repetido nos vetores atuais A e B, cada vez mais reduzidos em tamanho, sempre comparando os primeiros números de cada, até obter o vetor C ordenado por completo. Se A tiver mais números do que B, significa que $n > m$, então os números finais que sobrarão em A, sem sofrerem comparação, serão inseridos, na ordem em que estão, nas posições finais de C, começando pela posição $C[2m]$. Mas se B tiver mais números do que A, significa que $n < m$, então os números finais que sobrarão em B, sem sofrerem comparação, serão inseridos, na ordem em que estão, nas posições finais de C, começando pela posição $C[2n]$. Finalmente, o novo vetor C será retornado como solução.

Exemplo 50. Para ordenar o vetor $C = [\overset{C[1]}{5}, \overset{C[2]}{6}, \overset{C[3]}{7}, \overset{C[4]}{8}, \overset{C[5]}{1}, \overset{C[6]}{6}]$, o algoritmo de ordenação por intercalação fará o seguinte:

1. Considerar o vetores $A = [\overset{A[1]=C[1]}{5}, \overset{A[2]=C[2]}{6}, \overset{A[3]=C[3]}{7}, \overset{A[4]=C[4]}{8}]$ e $B = [\overset{B[1]=C[5]}{1}, \overset{B[2]=C[6]}{6}]$.
2. Compara 5 com 1 e ocupa a posição $C[1]$ com 1.
3. Em $A = [\overset{A[1]=C[1]}{5}, \overset{A[2]=C[2]}{6}, \overset{A[3]=C[3]}{7}, \overset{A[4]=C[4]}{8}]$ e $B_1 = [\overset{B_1[1]=C[6]}{6}]$, compara 5 com 6 e ocupa $C[2]$ com 5.
4. Em $A_1 = [\overset{A_1[1]=C[2]}{6}, \overset{A_1[2]=C[3]}{7}, \overset{A_1[3]=C[4]}{8}]$ e $B_1 = [\overset{B_1[1]=C[6]}{6}]$, compara 6 com 6 e ocupa $C[3]$ com o 6 de $C[2]$.
5. Em $A_2 = [\overset{A_2[1]=C[3]}{7}, \overset{A_2[2]=C[4]}{8}]$ e $B_1 = [\overset{B_1[1]=C[6]}{6}]$, compara 7 com 6 e ocupa $C[4]$ com 6.
6. Sobram 7 e 8, em $A_2 = [\overset{A_2[1]=C[3]}{7}, \overset{A_2[2]=C[4]}{8}]$. Então 7 ocupa $C[5]$ e 8 ocupa $C[6]$.
7. Retorna o vetor $C = [\overset{C[1]}{1}, \overset{C[2]}{5}, \overset{C[3]}{6}, \overset{C[4]}{6}, \overset{C[5]}{7}, \overset{C[6]}{8}]$.

Ordenação por Intercalação

- Entrada: O tamanho n do vetor A, bem como seus números, o tamanho m do vetor B e seus números.
- Saída: O vetor C, composto por todos os números de A e de B, ordenado por completo.
- Passo 1: Considere $i = 1$, $k = 1$ e $j = 1$.

  Observação 31. i indicará a posição em A, k indicará a posição em C e j a posição em B.

  - Passo 2 (*loop*): Enquanto qualquer uma das posições $A[i]$ e $B[j]$ não deixar de existir, faça:
    - Passo 3: Se $A[i] \leqslant B[j]$ então
      - Passo 4: Armazene o número da posição $A[i]$ na posição $C[k]$.
      - Passo 5: Incremente i e k e volte a executar a partir do passo 2.
    - Passo 6: Senão se $A[i] > B[j]$ então
      - Passo 7: Armazene o número da posição $B[j]$ na posição $C[k]$.
      - Passo 8: Incremente j e k e volte a executar a partir do passo 2.
  - Passo 9 (*loop*): Enquanto a posição $A[i]$ existir e a posição $B[j]$ não existir faça:
    - Passo 10: Armazene o número da posição $A[i]$ na posição $C[k]$.
    - Passo 11: Incremente i e k e volte a executar a partir do passo 9.
  - Passo 12 (*loop*): Enquanto a posição $A[i]$ não existir e a posição $B[j]$ existir faça:
    - Passo 13: Armazene o número da posição $B[j]$ na posição $C[k]$.
    - Passo 14: Incremente j e k e volte a executar a partir do passo 12.
- Passo 15: Retorne o vetor C, com seus $n+m$ números já dispostos em ordem crescente.

Exemplo 51. Aplicando os passos do algoritmo acima no vetor $C = [\overset{C[1]}{2}, \overset{C[2]}{4}, \overset{C[3]}{1}, \overset{C[4]}{2}, \overset{C[5]}{3}]$, obtém-se:

(a) $\overset{A[1]}{2}\ \overset{A[2]}{4}\quad \overset{B[1]}{1}\quad \overset{B[2]}{2}\ \overset{B[3]}{3}$

Levado a $C[1]$

$C = [\overset{C[1]}{1}, C[2], C[3], C[4], C[5]]$

(b) $\overset{A[1]}{2}\quad \overset{A[2]}{4}\ \overset{B'[1]}{2}\ \overset{B'[2]}{3}$

Levado a $C[2]$

$C = [\overset{C[1]}{1}, \overset{C[2]}{2}, C[3], C[4], C[5]]$

(c) $\overset{A'[1]}{4}\quad \overset{B'[1]}{2}\quad \overset{B'[2]}{3}$

Levado a $C[3]$

$C = [\overset{C[1]}{1}, \overset{C[2]}{2}, \overset{C[3]}{2}, C[4], C[5]]$

(d) $\overset{A'[1]}{4}\quad \overset{B''[1]}{3}$

Levado a $C[4]$

$C = [\overset{C[1]}{1}, \overset{C[2]}{2}, \overset{C[3]}{2}, \overset{C[4]}{3}, C[5]]$

(e) $\overset{A'[1]}{4}$

Levado a $C[4]$

$C = [\overset{C[1]}{1}, \overset{C[2]}{2}, \overset{C[3]}{2}, \overset{C[4]}{3}, \overset{C[5]}{4}]$

Por pseudocódigo, tem-se:

```
1 Procedimento MERGE (n, A, m, B)
2 i := 1, j := 1 e k := 1.
3 Enquanto i ⩽ n e j ⩽ m faça
4       Se a_i ⩽ b_j então faça
5             c_k := a_i
6             i := i + 1
7             k := k + 1
8 Senão faça
9             c_k := b_j
10            j := j + 1
11            k := k + 1
12 FimSe
13 FimEnquanto
14 Se i > n então faça
15      Para p := j até m, faça
16            c_k := b_p
17            k := k + 1
18      FimPara
19 Senão faça
20      Para p := i até n, faça
21            c_k := a_p
22            k := k + 1
23      FimPara
24 FimSe
```

Ao serem executadas as instruções das linhas 3 e 4, do algoritmo acima, ocorrem três comparações. Se a instrução da linha 3 assumir valor falso, então ocorrerá apenas duas comparações. Neste caso, pode-se obter a terceira comparação se for contada a instrução da linha 14. A quantidade de vezes em que as instruções das linhas 3 e 4 ocorrem não supera n. Então, no caso das linhas 3, 4 e 8, pode-se dizer que o número máximo de comparações é $3n = O(n)$. Portanto, o custo desse algoritmo é $O(n)$.

Para ordenar um vetor C, em ordem crescente, por intercalação, o qual possa não apresentar dois vetores ordenados, A e B, utiliza-se o procedimento Merge-Sort, recursivamente, para primeiramente dividir o vetor original em vários vetores menores (etapa de divisão), depois vai combinando sempre dois a dois vetores (etapa de conquista), para neles aplicar o procedimento de ordenação por intercalação (Merge).

Exemplo 52. Na ordenação dos elementos do vetor $C = [\overset{C[1]}{1}, \overset{C[2]}{3}, \overset{C[3]}{2}, \overset{C[4]}{7}, \overset{C[5]}{6}, \overset{C[6]}{8}, \overset{C[7]}{4}]$, o procedimento merge-sort fará o seguinte:

1. Divide C em dois vetores, $A = [\overset{A[1]=C[1]}{1}, \overset{A[2]=C[2]}{3}, \overset{A[3]=C[3]}{2}, \overset{A[4]=C[4]}{7}]$ e $B = [\overset{B[1]=C[5]}{6}, \overset{B[2]=C[6]}{8}, \overset{B[3]=C[7]}{4}]$.
2. Divide A em dois vetores, $A = [\overset{A[1]=C[1]}{1}, \overset{A[2]=C[2]}{3}]$ e $B = [\overset{B[1]=C[3]}{2}, \overset{B[2]=C[4]}{7}]$.
3. Divide $A = [\overset{A[1]=C[1]}{1}, \overset{A[2]=C[2]}{3}]$ em dois vetores, $A = [\overset{A[1]=C[1]}{1}]$ e $B = [\overset{B[1]=C[2]}{3}]$.
4. Verifica que A e B são unitários e os mescla, por meio do procedimento merge, retornando o vetor ordenado $A = [\overset{A[1]}{1}, \overset{A[2]}{3}]$.
5. Divide o vetor $B = [\overset{B[1]=C[3]}{2}, \overset{B[2]=C[4]}{7}]$ em dois vetores, $A = [\overset{A[1]=C[3]}{2}]$ e $B = [\overset{B[1]=C[4]}{7}]$.
6. Verifica que A e B são unitários e os mescla, por meio do procedimento merge, retornando o vetor ordenado $B = [\overset{B[1]}{2}, \overset{B[2]}{7}]$.
7. Mescla os vetores $A = [\overset{A[1]}{1}, \overset{A[2]}{3}]$ e $B = [\overset{B[1]}{2}, \overset{B[2]}{7}]$, por meio do procedimento merge, fornecendo o vetor ordenado $A = [\overset{A[1]}{1}, \overset{A[2]}{2}, \overset{A[3]}{3}, \overset{A[4]}{7}]$.

8. Divide $B = [\overset{B[1]=C[5]}{6}, \overset{B[2]=C[6]}{8}, \overset{B[3]=C[7]}{4}]$ em dois vetores, $A = [\overset{A[1]=C[5]}{6}, \overset{A[2]=C[6]}{8}]$ e $B = [\overset{B[1]=C[7]}{4}]$.
9. Divide $A = [\overset{A[1]=C[5]}{6}, \overset{A[2]=C[6]}{8}]$ em dois vetores, $A = [\overset{A[1]=C[5]}{6}]$ e $B = [\overset{B[1]=C[6]}{8}]$.
10. Verifica que A e B são unitários e os mescla, por meio do procedimento merge, retornando o vetor ordenado $A = [\overset{A[1]}{6}, \overset{A[2]}{8}]$.
11. Verifica que $B = [\overset{B[1]=C[7]}{4}]$ é unitário e o mescla com o vetor $A = [\overset{A[1]}{6}, \overset{A[2]}{8}]$, por meio do procedimento merge, retornando o vetor ordenado $B = [\overset{B[1]}{4}, \overset{B[2]}{6}, \overset{B[3]}{8}]$.
12. Mescla os vetores $A = [\overset{A[1]}{1}, \overset{A[2]}{2}, \overset{A[3]}{3}, \overset{A[4]}{7}]$ e $B = [\overset{B[1]}{4}, \overset{B[2]}{6}, \overset{B[3]}{8}]$, por meio do procedimento merge, retornando o vetor ordenado $C = [\overset{C[1]}{1}, \overset{C[2]}{2}, \overset{C[3]}{3}, \overset{C[4]}{4}, \overset{C[5]}{6}, \overset{C[6]}{7}, \overset{C[7]}{8}]$.

■

Dando um caráter mais formal para esse procedimento, obtém-se o algoritmo abaixo.

Merge-Sort

- Entrada: O tamanho n do vetor C, bem como seus números.
- Saída: O vetor C ordenado.
- Passo 1: Se C for unitário ($n = 1$) então pare.

  Observação 32. Neste passo, finaliza-se a etapa de divisão do vetor atual, ao se obter um vetor unitário. Essa finalização, se ocorrida para o vetor A, dará início à nova etapa de divisão do vetor atual B. Agora se ocorrida para o vetor B, dará início ao procedimento Merge, para mesclar A e B atuais, retornando um só vetor ordenado.
- Passo 2: Mas se C não for unitário, faça m assumir o valor da expressão $\lceil \frac{n}{2} \rceil$.

  Observação 33. m é uma variável auxiliar usada para a divisão do vetor que se quer ordenar em dois vetores A e B.

  - Passo 3: Considere $i = 0$.

    Observação 34. i indica a posição no vetor atual, que se deseja ordenar.

    * Passo 4: Armazene o número da posição i em $A[i]$.
    * Passo 5: Incremente i. Se $i < m$ então volte a executar a partir do passo 4. Caso contrário, execute o próximo passo.

      Observação 35. Os passos 4 e 5 são responsáveis por fornecer o vetor A.
  - Passo 6: Considere $i = m$.

    * Passo 7: Armazene o número da posição i em $B[i - m]$.
    * Passo 8: Incremente i. Se $i < n$ então volte a executar a partir do passo 7. Caso contrário, execute o próximo passo.

Observação 36. Os passos 7 e 8 são responsáveis por fornecer o vetor B.

- Passo 9: Aplique novamente o procedimento **Merge-Sort** no vetor A.
- Passo 10: Aplique novamente o procedimento **Merge-Sort** no vetor B.
- Passo 11: Aplique o procedimento **Merge** nos vetores atuais, A e B, já ordenados, para retornar um só vetor ordenado.

■

O teste de mesa realizado no exemplo abaixo é longo, porém necessário, para que se possa entender cada passo e como cada número é organizado de forma a retornar o vetor C totalmente ordenado.

Exemplo 53. Aplicando o procedimento Merge-Sort no vetor $C = \overset{C[0]}{[38}, \overset{C[1]}{27}, \overset{C[2]}{43}, \overset{C[3]}{3}, \overset{C[4]}{9}]$, para ordená-lo, obtém-se:

- (1) Entrada: $n = 5$ e $c_0 = 38$, $c_1 = 27$, $c_2 = 43$, $c_3 = 3$ e $c_4 = 9$.
- Saída: O vetor C ordenado.
- Passo 1: O vetor C não é unitário. Então continue.
- Passo 2: m assume o valor da expressão $\left\lceil \frac{n}{2} \right\rceil = \left\lceil \frac{5}{2} \right\rceil = \lceil 2,5 \rceil = 3$.
  - Passo 3: $i = 0$.
    - Passo 4: 38 é o número da posição $i = 0$. Este número agora é armazenado na posição $A[i] = A[0]$, do vetor A.
    - Passo 5: i é incrementado e passa a valer 1, que é menor do que $m = 3$. Então volte a executar a partir do passo 4.
    - Passo 4: 27 é o número da posição $i = 1$. Este número é armazenado na posição $A[i] = A[1]$.
    - Passo 5: i é incrementado e passa a valer 2, que é menor do que $m = 3$. Então volte a executar a partir do passo 4.
    - Passo 4: 43 é o número da posição $i = 2$. Este número é armazenado na posição $A[i] = A[2]$.
    - Passo 5: i é incrementado e passa a valer 3, que é que é igual a m. Então execute o próximo passo.
  - Passo 6: $i = m = 3$.
    - Passo 7: 3 é o número da posição $i = 3$. Este número é armazenado na posição $B[i - m] = B[3 - 3] = B[0]$.
    - Passo 8: i é incrementado e passa a valer 4, que é menor do que $n = 5$. Então volte a executar a partir do passo 7.
    - Passo 7: 9 é o número da posição $i = 4$. Este número é armazenado na posição $B[i - m] = B[4 - 3] = B[1]$.

* Passo 8: i é incrementado e passa a valer 5, que é igual a n. Então execute o próximo passo.

  Observação 37. Vetores obtidos: $A = [\overset{A[0]}{38}, \overset{A[1]}{27}, \overset{A[2]}{43}]$ e $B = [\overset{B[0]}{3}, \overset{B[1]}{9}]$.

– (2) Passo 9: Volte a aplicar o procedimento Merge-Sort no vetor $A = [\overset{A[0]}{38}, \overset{A[1]}{27}, \overset{A[2]}{43}]$.

– Entrada: $n_1 = 3$ e $a_0 = 38$, $a_1 = 27$ e $a_2 = 43$.

– Saída: O vetor A ordenado.

– Passo 1: O vetor A não é unitário. Então continue.

– Passo 2: m_1 assume o valor da expressão $\lceil \frac{n_1}{2} \rceil = \lceil \frac{3}{2} \rceil = \lceil 1,5 \rceil = 2$.

* Passo 3: $i = 0$.
  - Passo 4: 38 é o número da posição $i = 0$. Este número é armazenado na posição $A[i] = A[0]$, de A.
  - Passo 5: i é incrementado e passa a valer 1, que é menor do que $m_1 = 2$. Então volte a executar a partir do passo 4.
  - Passo 4: 27 é o número da posição $i = 1$. Este número é armazenado na posição $A[i] = A[1]$, de A.
  - Passo 5: i é incrementado e passa a valer 2, que é igual a m_1. Então execute o próximo passo.
* Passo 6: $i = m_1 = 2$.
  - Passo 7: 43 é o número da posição $i = 2$. Este número é armazenado na posição $B[i - m_1] = B[2 - 2] = B[0]$.
  - Passo 8: i é incrementado e passa a valer 3, que é igual a n_1. Então execute o próximo passo.

    Observação 38. Vetores obtidos: $A = [\overset{A[0]}{38}, \overset{A[1]}{27}]$ e $B = [\overset{B[0]}{43}]$.
* Passo 9: Volte a aplicar o procedimento Merge-Sort no vetor $A = [\overset{A[0]}{38}, \overset{A[1]}{27}]$.
* Entrada: $n_2 = 2$ e $a_0 = 38$, $a_1 = 27$.
* Saída: O vetor A ordenado.
* Passo 1: O vetor A não é unitário. Então continue.
* Passo 2: m_2 assume o valor da expressão $\lceil \frac{n_2}{2} \rceil = \lceil \frac{2}{2} \rceil = \lceil 1 \rceil = 1$.

. Passo 3: $i = 0$.

  * Passo 4: 38 é o número da posição $i = 0$. Este número é armazenado na posição $A[i] = A[0]$, de A.

  * Passo 5: i é incrementado e passa a valer 1, que é igual a m_2. Então execute o próximo passo.

. Passo 6: $i = m_2 = 1$.

  * Passo 7: 27 é o número da posição $i = 1$. Este número é armazenado na posição $B[i - m_2] = B[1 - 1] = B[0]$.

  * Passo 8: i é incrementado e passa a valer 2, que é igual a n_2. Então execute o próximo passo.

Observação 39. Vetores obtidos: $A = [\overset{A[0]}{38}]$ e $B = [\overset{B[0]}{27}]$.

. Passo 9: Volte a aplicar o procedimento Merge-Sort no vetor $A = [\overset{A[0]}{38}]$.

- Entrada: $n_3 = 2$ e $a_0 = 38$.
- Saída: O vetor A ordenado.
- Passo 1: O vetor A é unitário. Então pare, pois ele já está ordenado.

. Passo 10: Volte a aplicar o procedimento Merge-Sort no vetor $B = [\overset{B[0]}{27}]$.

- Entrada: $n_4 = 1$ e $b_0 = 27$.
- Saída: O vetor B ordenado.
- Passo 1: O vetor B é unitário. Então pare, pois ele já está ordenado.

. Passo 11: O procedimento Merge é aplicado nos vetores $A = [\overset{A[0]}{38}]$ e $B = [\overset{B[0]}{27}]$, da observação 39, mas já ordenados, fornecendo o vetor ordenado $A = [\overset{A[0]}{27}, \overset{A[1]}{38}]$, que será utilizado pelo procedimento Merge do procedimento iniciado em (2).

* Passo 10: Volte a aplicar o procedimento Merge-Sort no vetor $B = [\overset{B[0]}{43}]$, da observação 38.
* Entrada: $n_5 = 1$ e $b_0 = 43$.
* Saída: O vetor B ordenado.
* Passo 1: O vetor B é unitário. Então pare, pois ele já está ordenado.
* Passo 11: O procedimento Merge é aplicado nos vetores $A = [\overset{A[0]}{27}, \overset{A[1]}{38}]$ e $B = [\overset{B[0]}{43}]$, da observação 38, mas já ordenados, fornecendo o vetor ordenado $A = [\overset{A[0]}{27}, \overset{A[1]}{38}, \overset{A[2]}{43}]$, que será utilizado pelo procedimento Merge do procedimento iniciado em (1).

– Passo 10: Volte a aplicar o procedimento Merge-Sort no vetor $B = [\overset{B[0]}{3}, \overset{B[1]}{9}]$, da observação 37.

– Entrada: $n_6 = 2$ e $b_0 = 3$ e $b_1 = 9$.

– Saída: O vetor B ordenado.

– Passo 1: O vetor B não é unitário. Então continue.

* Passo 2: m_3 assume o valor da expressão $\left\lceil \frac{n_6}{2} \right\rceil = \left\lceil \frac{2}{2} \right\rceil = \lceil 1 \rceil = 1$.

. Passo 3: $i = 0$.

* Passo 4: 3 é o número da posição $i = 0$. Este número é armazenado na posição $A[i] = A[0]$, de A.

* Passo 5: i é incrementado e passa a valer 1, que é igual a m_3. Então execute o próximo passo.

. Passo 6: $i = m_3 = 1$.

* Passo 7: 9 é o número da posição $i = 1$. Este número é armazenado na posição $B[i - m_3] = B[1 - 1] = B[0]$.

* Passo 8: i é incrementado e passa a valer 2, que é igual a n_6. Então execute o próximo passo.

Observação 40. Vetores obtidos: $A = [\overset{A[0]}{3}]$ e $B = [\overset{B[0]}{9}]$.

. Passo 9: Volte a aplicar o procedimento Merge-Sort no vetor $A = [\overset{A[0]}{3}]$.

. Entrada: $n_7 = 1$ e $a_0 = 3$.

. Saída: O vetor A ordenado.

. Passo 1: O vetor A é unitário. Então pare, pois ele já está ordenado.

. Passo 10: Volte a aplicar o procedimento Merge-Sort no vetor $B = [\overset{B[0]}{9}]$.

. Entrada: $n_8 = 1$ e $b_0 = 9$.

. Saída: O vetor B ordenado.

. Passo 1: O vetor B é unitário. Então pare, pois ele já está ordenado.

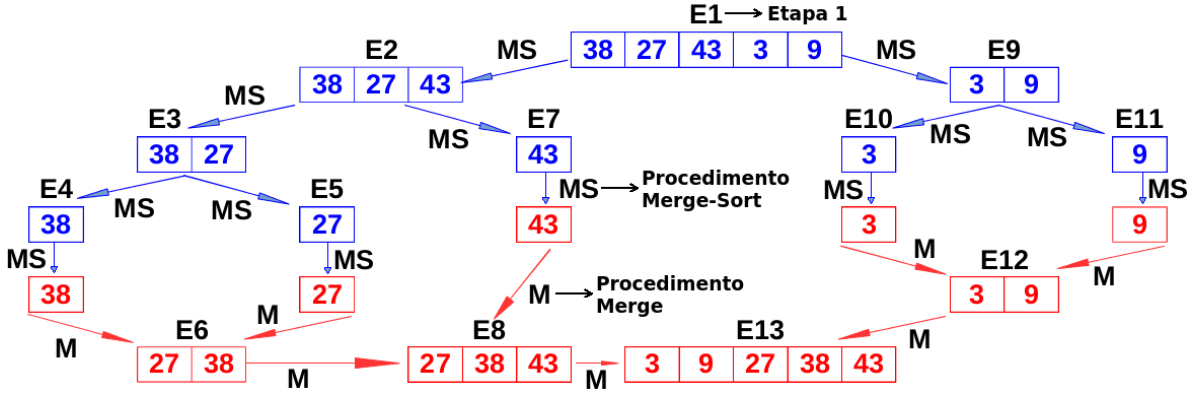
. Passo 11: O procedimento Merge é aplicado nos vetores $A = [\overset{A[0]}{3}]$ e $B = [\overset{B[0]}{9}]$, da observação 40, mas já ordenados, fornecendo o vetor ordenado $B = [\overset{B[0]}{3}, \overset{B[1]}{9}]$, que será utilizado pelo procedimento Merge do procedimento iniciado em (1).

– Passo 11: O procedimento Merge é aplicado nos vetores $A = [\overset{A[0]}{27}, \overset{A[1]}{38}, \overset{A[2]}{43}]$ e $B = [\overset{B[0]}{3}, \overset{B[1]}{9}]$, da observação 37, mas já ordenados, fornecendo o vetor ordenado $C = [\overset{C[0]}{3}, \overset{C[1]}{9}, \overset{C[1]}{27}, \overset{C[1]}{38}, \overset{C[1]}{43}]$, que é a saída.

■

De forma simplificada, tem-se:

Figura 4.2.1: Merge-Sort



Por pseudocódigo, tem-se:

```
1 Procedimento Merge-Sort(n, C)
2 Se n = 1 então
3       Pare, pois o vetor ou vetor já está ordenado.
4 Senão Se n > 1, então
5       m := ⌈n/2⌉
6       Para i = 0 até m − 1, faça
7             A[i] : C[i]
8       Para i = m até n − 1, faça
9             B[i − m] : C[i]
10      Merge-Sort(m, A)
11      Merge-Sort(n − m, B)
12      Merge(m, A, n − m, B)
13FimSe
```

4.2.4.1 Custo da Ordenação Por Intercalação (Pior Caso)

Dos procedimentos Merge e Merge-Sort, obtém-se a seguinte recorrência:

$$T(n) = \begin{cases} O(1), & se\ n = 1 \\ T\left(\lceil \frac{n}{2} \rceil\right) + T\left(\lfloor \frac{n}{2} \rfloor\right) + O(n), & se\ n > 1 \end{cases},$$

na qual $O(1)$ é o custo da instrução da linha 2 do procedimento Merge-Sort, $T\left\lceil \frac{n}{2} \right\rceil$ e $T\left\lfloor \frac{n}{2} \right\rfloor$ são os respectivos tempos de execução das instruções posicionadas nas linhas 10 e 11, desse algoritmo, e $O(n)$ é o custo do procedimento Merge (linha 12).

Para a análise assintótica O, pode-se considerar $n = 2^k$. Assim,

$$T(2^k) = \begin{cases} 1, & se\ k = 0 \\ 2.T\left(2^{k-1}\right) + 2^k, & se\ k > 0 \end{cases},$$

Logo, para $k > 0$, tem-se:

$$\begin{aligned} T\left(2^k\right) &= 2T\left(2^{k-1}\right) + 2^k \\ &= 2^2T\left(2^{k-2}\right) + 2 \cdot 2^{k-1} + 2^k = 2^2T\left(2^{k-2}\right) + 2 \cdot 2^k \\ &= 2^3T\left(2^{k-3}\right) + 2^2 \cdot 2^{k-2} + 2 \cdot 2^k = 2^3T\left(2^{k-3}\right) + 3 \cdot 2^k \\ &\vdots \quad \vdots \\ &= 2^kT\left(2^{k-k}\right) + k \cdot 2^k \\ &= 2^kT\left(1\right) + k \cdot 2^k \\ &= 2^k + k \cdot 2^k \\ &= 2^kk + 2^k \end{aligned}$$

Logo,

$$T(n) = T\left(2^k\right) = 2^kk + 2^k = n.log_2n + n \Rightarrow$$

$$T(n) = n.log_2 n + n.$$

Para $n \geqslant 2$, tem-se:

$$nlog_2 n + n = n\,(log_2 n + 1) \leqslant n.\,(log_2 n + log_2 n) = 2n \cdot log_2 n.$$

Portanto,

$$T(n) = O\,(n.log_2 n).$$

Capítulo 5

Considerações Finais

Um aluno sabendo construir um pseudocódigo e tendo habilidade com determinada linguagem de programação, pode ser estimulado, pelo professor, a implementar algoritmos computacionais, para solucionar problemas da matemática elementar. Além disso, poderá analisar os algoritmos que criou para solucionar um mesmo problema, por meio de funções que calculam o número de passos requeridos por cada algoritmo, sendo capaz de escolher o que possuir a menor ordem de crescimento, para uma entrada suficientemente grande.

Trabalhando com o teste de mesa para cada algoritmo implementado, e tentando sempre descobrir métodos eficientes para solucionar problemas da matemática básica, o aluno terá a oportunidade de desenvolver seu raciocínio, podendo assim abordar conteúdos matemáticos com agilidade.

O tema aqui abordado propõe a sua integração com o currículo da educação básica, tanto por tratar de matemática básica como por apresentar uma aplicação da matemática num contexto real computacional.

Referências Bibliográficas

[1] ALBERTSON, Michael O.; HUTCHINSON, Joan P.. DISCRETE MATHEMATICS WITH ALGORITHMS. John Wiley & Sons, 1988.

[2] BARROSO, Leônidas Conceição; BARROSO, Magali Maria de Araújo; FILHO, Frederico Ferreira Campos; CARVALHO, Márcio Luiz Bunte de e MAIA, Miriam Lourenço. CÁLCULO NUMÉRICO (COM APLICAÇÕES), 2ª edição. Editora Harbra LTDA, 1987.

[3] BOLDRINI, José Luiz; COSTA, Sueli I. Rodrigues; FIGUEIREDO, Vera Lúcia e WETZLER, Henry G. ÁLGEBRA LINEAR -3ª Edição. Editora Harbra LTDA, 1986.

[4] COUTINHO, S. C. Números Inteiros e Criptografia RSA, Coleção Computação e Matemática, SBM e IMPA, 2000.

[5] DOMINGUES, Hygino H. e IEZZI, Gelson. Álgebra Moderna. Editora Atual, 1982.

[6] ESQUINCA, Josiane C. Pedrini. Aritmética: Código de Barras e Outras Aplicações de Congruências. Trabalho de Conclusão de Curso, 2013. Matemática em Rede Nacional - Mestrado Profissional. Universidade Federal de Mato Grosso do Sul.

[7] GUIDORIZZI, Hamilton Luiz, UM CURSO DE CÁLCULO, 5ª edição, volume 1. Livros Técnicos e Científicos Editora S.A., 1985.

[8] IEZZI, Gelson. FUNDAMENTOS DE MATEMÁTICA ELEMENTAR, 5ª edição, volume 6. Atual Editora, 1985.

[9] JAMIL, George Leal e GOUVÊA, Bernardo Andrade. Linux Para Profissonais: Do Básico à Conexão em Redes. Axcel Books do Brasil Editora LTDA, 2006. ISBN: 85-7323-252-8.

[10] KNUTH, Donald Ervin. THE ART OF COMPUTER PROGRAMMING, 2d edition, volume 2. ADDISON-WESLEY PUBLISHING COMPANY, 1938.

[11] LEITHOLD, Louis. O Cálculo com Geometria Analítica, 3ª edição, volumes 1 e 2. Editora Harbra LTDA, 1994.

[12] LIMA, Elon Lages, Curso de Análise (Projeto Euclides), volume 1, 11ª edição. Associação Insituto Nacional de Matemática Pura e Aplicada, 2004. ISBN: 85-244-0118-4.

[13] LUCCHESI, Cláudio L.; SIMON, Imre Simon Istvan; SIMON, Janos; KOWALTOWSKI, Tomasz. Aspectos Teóricos da Computação (Projeto Euclides). Instituto de Matemática Pura e Aplicada, 1979.

[14] MOREIRA, Carlos Gustavo T. de Araújo; MARTINEZ, Fabio E. Brochero e SALDANHA, Nicolau Corção. TÓPICOS DE TEORIA DOS NÚMEROS (Coleção Profmat). Sociedade Brasileira de Matemática, 1ª edição, 2012.

[15] OLIVEIRA, Maykon Costa de. Aritmética: Criptografia e Outras Aplicações de Congruência. Trabalho de Conclusão de Curso, 2013. Matemática em Rede Nacional - Mestrado Profissional. Universidade Federal de Mato Grosso do Sul.

[16] ROSEN, Kenneth H. Discret Mathematics and its applications, 3rd ed. McGraw-Hill, Inc., 1995.

[17] RUGGIERO, Márcia A. Gomes e LOPES, Vera Lúcia da Rocha. Cálculo Numérico - Aspectos Teóricos e Computacionais, 2ª Edição. MAKRON Books do Brasil Editora Ltda, 1997. ISBN: 85-346-0204-2.

[18] VELOSO, Paulo A. S. e TOSCANI, Laira Vieira. Complexidade de Algoritmos: Análise, Projeto e Métodos. Editora Sagra Luzzatto, 1ª edição, 2001.

[19] http://www.ime.usp.br/~song/mac5710/slides/01complex.pdf, acessado em 25/09/2014.

[20] http://pt.wikipedia.org/wiki/Complexidade_computacional, acessado em 25/09/2014.

[21] http://www.cin.ufpe.br/~joa/menu_options/school/cursos/ppd/aulas/complexidade.pdf, acessado em 25/09/2014.

[22] http://www.inf.ufrgs.br/~prestes/Courses/Complexity/aula1.pdf, acessado em 25/09/2014.

Apêndice

A.1 Implementação do algoritmo intercalar, para dois vetores de mesmo tamanho, em linguagem shell

```
#!/bin/sh
echo "Entre com o tamanho
dos dois vetores:"
read n
for((i=0;i<n;i++))
do
    echo "Digite o número da
    posição $i no primeiro arranjo:"
read a[$i]
done
for((i=0;i<n;i++))
do
    echo "Digite o número da
    posição $i no segundo arranjo:"
read b[$i]
done
echo "Merge Sort"
j=0
k=0
i=0
let num=n+n
for((x=0;x<num;x++))
do
    if [ ${a[$j]} -le ${b[$k]} ]
    then c[$i]=${a[$j]}
        i=$(expr $i+1)
        j=$(expr $j+1)
else
        c[$i]=${b[$k]}
        i=$(expr $i+1)
        k=$(expr $k+1)
fi
if [ $j -eq $n -o $k -eq $n ]
then
    break
fi
done
for((;j<n;))
do
    c[$i]=${a[$j]}
    i=$(expr $i+1)
    j=$(expr $j+1)
done
for((;k<n;))
do
    c[$i]=${b[$k]}
    i=$(expr $i+1)
    k=$(expr $k+1)
done
echo "Números ordenados:"
for((i=0;i<10;i++))
do
    echo ${c[$i]}
done
```

A.2 Divisão Euclidiana - Shell Script

```
#!/bin/bash
echo “Digite o valor de a (dividendo):”;
read a
echo “Digite o valor de b (divisor):”;
read b
q=‘expr $a / $b‘
r=‘expr $a - $b \* $q‘
echo “O resto da divisão de $a por $b é igual a $r.”;
if [ $r -eq 0 ] then
    echo “$b divide $a, pois o resto é igual a zero.”;
```

```
11 else
12        echo "$b não divide $a, pois o resto é diferente de zero.";
13 fi
```

A.3 Algoritmo de Euclides - Shell Script

```
1 #!/bin/bash
2 echo "Procedimento: Algoritmo de Euclides - (a, b).";
3 echo "Digite o valor de a:";
4 read a
5 echo "Digite o valor de b:";
6 read b
7 if [ $a == 1 ] then
8       echo "($a, $b) = 1.";
9       exit
10 elif [ $b == 1 ] then
11       echo ($a, $b) = 1.";
12       exit
13 fi
14 if [ $a -le $b ] then
15       x=$a
16 else
17       x=$b
18 fi
19 m=$a
20 n=$b
21 q=`expr $a / $b`
22 r=`expr $a - $b \* $q`
23 a=$b
24 b=$r
25 while [ $r != 0 ];
26 do
27       x=$r
28       q=`expr $a / $b`
29       r=`expr $a - $b \* $q`
30       a=$b
31       b=$r
32 done
33 echo "($m, $n) = $x.";
```

A.4 Algoritmo Estendido de Euclides - Shell Script

```
1 #!/bin/bash
2 echo "Algoritmo Estendido de Eucli-
3 des para o cálculo do mdc(a, b).";
4 echo "Digite o valor de a:";
5 read a
6 echo "Digite o valor de b:";
7 read b
8 if [ $b == 0 ] then
9 d=$a
10 x=1;
11 y=0;
12 echo "($a, $b) = $a, x = 1 e y = 0.";
13 fi
14 r=$a
15 rr=$b
16 u=1;
17 v=0;
18 uu=0;
19 vv=1;
20 while [ $rr != 0 ];
21 do
22       q=`expr $r / $rr`;
23       rs=$r
24       us=$u
25       vs=$v
26       r=$rr
27       u=$uu
28       v=$vv
29       rr=`expr $rs - $q \* $rr`;
30       uu=`expr $us - $q \* $u`;
31       vv=`expr $vs - $q \* $vv`;
32 done
33 c=`expr $a \* $u + $b \* $v`;
34 echo "($a, $b) = $c.";
35 echo " x = $u e y = $v.";
```