# Tópicos de Programação

Régis S. Santos

2013

# Sumário

# Capítulo 1

# Introdução

Livros recomendados: [1, Cormen], [2, Varizani], [3, Feofiloff].

## 1.1 Lembrando algumas estruturas importantes

### Um tipo de variável diferente

`unsigned int` $= \mathbb{N}$

### Registro em C

Para criar novos tipos de variáveis.

**Exemplo 1.1** Um registro `x` com três campos inteiros.

```
struct{
  int dia, mes, ano;
} x;
```

**Exemplo 1.2** Ponto no plano cartesiano.

```
struct{
  int x,y;
} p1;
```

**Exemplo 1.3** Dando nome ao registro.

```
struct ponto{
  int x,y;
};
struct ponto p1;
struct ponto p2;
```

**Exemplo 1.4** Acessando os campos do registro.

```
p1.x = 3;
p1.y = 2;
```

## 1.2 Ponteiro

Endereço: &

**Exemplo 1.5** `x` armazenando o endereço de `y`.

```
int *x;
int y = 40;
/* x armazena o endereco de y*/
x = &y;
printf("%d%d",y,*x);
```

## 1.3 Esqueleto em C

```
#include<stdio.h>
int main(){
  /* declaracao de variaveis */
  /* corpo do programa */
  return 0;
}
```

## 1.4 Vetores

**Notação**: Se a lista de números está armazenada nas posições $0, 1, \ldots, n-1$, diremos que $v[0 \ldots n-1]$ é um *vetor* de inteiros.

Devemos ter $0 \leqslant n \leqslant N$.

Se $n = 0$, então $v$ está vazio.

Se $n = N$, então $v$ está cheio.

## 1.5 Operações com vetores

### Busca

Dado um inteiro $x$ e um vetor de inteiros $v[0 \ldots n-1]$, considere o problema de encontrar um índice $k$ tal que $v[k] = x$.

**Exemplo 1.6** Sejam $x = 987, N = 8$ e $n = 5$.

| | $0$ | | | | $n-1$ | | | $N$ |
|---|---|---|---|---|---|---|---|---|
| $v$ | 222 | 555 | 343 | 988 | 531 | 333 | 987 | 999 |

Então $k = -1$.

```
/* Esta funcao recebe um numero x e um vetor v[0..n-1]
   tal que v[k] = x. Se tal k nao existe, devolve -1. */

int Busca(int x, int v[], int n){
  int k;
  k = n - 1;
  while (k >= 0 && v[k] != x)
    k--;
  return k;
}
```

### Remoção

Consiste em retirar do vetor $v[0 \dots n-1]$ o elemento que tem índice $k$ e fazer com que o vetor resultante tenha índices $0, 1, \dots, n-2$.

**Exemplo 1.7** Vamos remover o elemento de índice $k = 3$.

| | 0 | 1 | 2 | 3 | 4 | | | $N$ |
|---|---|---|---|---|---|---|---|---|
| $v$ | 222 | 555 | 343 | ~~988~~ | 531 | 333 | 987 | 999 |

| | 0 | 1 | 2 | 3 | | | | $N$ |
|---|---|---|---|---|---|---|---|---|
| $v$ | 222 | 555 | 343 | 531 | 531 | 333 | 987 | 999 |

```
/* Remove o elemento de indice k do vetor v[0...n-1]
   e devolve o novo valor de n. A funcao supoe 0 <= k < n.*/

int Remove(int k, int v[], int n){
  int j;
  for (j = k; j < n - 1; j++)
    v[j] = v[j+1];
  return n - 1;
}
```

### Inserção

Consiste em introduzir um novo elemento $y$ entre a posição de índice $k-1$ e $k$ no vetor $v[0 \dots n-1]$. Isto faz sentido somente quando $1 \leqslant k \leqslant n-1$.

Quando $k = 0$, inserimos no início de $v$.

Quando $k = n$, inserimos no final de $v$.

```
/* Insere y entre as posicoes k-1 e k de v[0...n-1]
   e devolve o novo valor de n. A funcao supoe 0 <= k <= n.*/

int Insere(int k, int y, int v[], int n){
  int j;
  for (j = n; j > k; j--)
    v[j] = v[j-1];
  v[k] = y;
  return n + 1;
}
```

**Exemplo 1.8** Sejam $k = 2$ e $y = 133$.

| | 0 | 1 | 2 | 3 | 4 | | | $N$ |
|---|---|---|---|---|---|---|---|---|
| $v$ | 222 | 555 | 343 | 988 | 531 | 333 | 987 | 999 |

| | 0 | 1 | 2 | 3 | 4 | 5 | | $N$ |
|---|---|---|---|---|---|---|---|---|
| $v$ | 222 | 555 | 133 | 343 | 988 | 531 | 987 | 999 |

### Alocação dinâmica de vetor em C

```
int *v;
v = (int*) malloc(100*sizeof(int));
```

## 1.6 Matrizes bidimensionais

São estruturas de dados implementadas como vetores de vetores, com $m$ linhas e $n$ colunas.

```
A[i][j] = 1;
```

### Alocação dinâmica de matriz em C

```
int **A;
int i;
A = (int**) malloc(m*sizeof(int*));
for (i = 0; i < m; i++)
  A[i] = (int*) malloc(n*sizeof(int));
```

### Busca em matriz

Dado um inteiro $x$ e uma matriz de inteiros $A$ com $m$ linhas e $n$ colunas, encontre dois índices $kl$ e $kc$ tal que $A[kl][kc] = x$.

```
void BuscaMatriz(int x, int **A, int m, int n, int *kl, int *kc){
  for (*kl = n-1; *kl >= 0; (*kl)--) {
    *kc = n - 1;
    while (*kc >= 0 && A[*kl][*kc] != x)
      (*kc)--;
    if (*kc != -1)
      return;
  }
}
```

## 1.7 Exercícios

**Exercício 1** Criar um registro *IndiceMatriz* com campos inteiros *linha* e *coluna* e reescrever a função `BuscaMatriz` usando tal registro.

# Capítulo 2

# Análise Assintótica

Análise de algoritmos, olhe para expressões como $n + 10$ e $n^2 + 1$ ignorando valores pequenos para $n$. O que interessa são os valores *grandes* para $n$.

Dizemos que as funções $n^2$, $\frac{3}{4}n^2$, $9n^2$, $1000n^2$, $\frac{1}{100}n^2 + 100n$, crescem com a mesma velocidade e por isso são *equivalentes*, quando $n$ é muito grande.

Na análise assintótica todas as funções mencionadas anteriomente estão na mesma *ordem de grandeza*.

Nos interessa apenas olhar para as funções assintóticamente *não negativas*, isto é, funções $f$ tais que $f(n) \geqslant 0$ para todo $n$ suficientemente grande.

## 2.1 Ordem O

**Definição 2.1** Dadas funções assintóticamentes não negativas $f$ e $g$, dizemos que $f$ *está na ordem O de g* e escrevemos $f = O(g)$, se $f(n) \leqslant cf(n)$ para algum número positivo $c$ e para todo $n$ suficientemente grande. Ou seja, existe um número positivo $c$ e um número $n_0$ tal que $f(n) \leqslant cg(n), \forall n > n_0$.

**Exemplo 2.1** Se $f(n) \leqslant 999g(n), \forall n \geqslant 1000$ então $f$ está na ordem O de $g$. $f = O(g)$.

**Exemplo 2.2** Suponha que $f(n) = 3 + \frac{2}{n}$ e que $g(n) = n^0 = 1$

**Solução**:
$f(n) = 3 + \frac{2}{n} \leqslant 3 + 1 = 4$ □

**Exemplo 2.3** Suponha que $f(n) = n^3$ e que $g(n) = 200n^2$.

**Solução**:
Suponha que $f = O(g)$. Então existe um número positivo $c$ e um número $n_0$ tal que $f(n) \leqslant cg(n), \forall n > n_0$.

Dividindo por $c$ ambos os lados da desigualdade, temos
$\frac{1}{c}f(n) \leqslant g(n), \forall n > n_0$.

Substituindo $f$ e $g$, temos $\frac{1}{c}n^3 \leqslant 200n^2, \forall n > n_0$.

$$\frac{1}{c}n \leqslant 200, \forall n > n_0 \Rightarrow \frac{n}{200} \leqslant c, \forall n > n_0$$

Absurdo. Portanto, $f$ não está na ordem $O(g)$. □

## 2.2 Ordem Omega

Quando escrevemos $f = O(g)$ queremos dizer que "a função $f$ está sempre por baixo, ou tocando a função $g$", a partir de um determinado ponto.

Quando escrevemos $f = \Omega(g)$ queremos dizer que "a função $f$ está por cima, ou tocando a função $g$", a partir de um determinado ponto.

**Definição 2.2** Dadas funções assintóticamente não negativas $f$ e $g$, dizemos que $f$ *está na ordem Omega de* $g$ ($f = \Omega(g)$), se $f(n) \geqslant cg(n)$ para algum $c$ positivo e para todo $n$ suficientemente grande. Ou seja, $\exists c > 0$ e $n_0$ tais que $f(n) \geqslant cg(n), \forall n > n_0$.

**Exemplo 2.4** Se $f(n) \geqslant \dfrac{g(n)}{1000}, \forall n \geqslant 888$, então $f$ está na ordem Omega de $g$.

**Solução**:

Devemos provar que se $f = O(g)$, então $g = \Omega(f)$.

$\frac{1}{c}f(n) \leqslant g(n), \forall n > n_0$. □

**Exemplo 2.5** Prove que $100 \lg n - 10n + 2n \lg n$ está na ordem de Omega de $n \lg n$.

**Solução**:

$$f(n)=100 \lg n-10n+n \lg n+n \lg n \geqslant 100 \lg n-10n+10n+n \lg n \geqslant 1n \lg n, \forall n \geqslant 1024$$

$$\therefore f(n) = \Omega(n \lg n)$$

□

## 2.3 Ordem Theta

**Definição 2.3** Dizemos que $f = \Theta(g)$, se $f = O(g)$ e $f = \Omega(g)$. Isto significa que existem números positivos $c$ e $d$, tais que $cg(n) \leqslant f(n) \leqslant dg(n)$, para todo $n$ suficientemente grande.

# Capítulo 3

# Recorrência

**Definição 3.1** Uma *recorrência* é uma expressão escrita em termos dela mesma.

Por exemplo, $F(n) = F(n-1) + 3n + 2$. O valor de $F(n)$ depende do valor de $F(n-1)$.

Resolver uma recorrência $F$ é encontrar uma "fórmula fechada" para $F$ que não depende de $F$ e que dependa somente dos seus parâmetros.

**Exemplo 3.1** Suponha uma recorrência com valor inicial $F(1) = 1$ e $F(n) = F(n-1) + 3n + 2$.

**Solução**:

| $n$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $F(n)$ | 1 | 9 | 20 | x | x |

A fórmula é

$$F(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4$$

Façamos a prova por indução:

Para $n = 1$, temos $F(1) = \frac{3}{2}.1^2 + \frac{7}{2}.1 - 4 = 1$.

Tome $n > 1$ e suponha que a fórmula fechada vale para $n - 1$.

$$\begin{aligned}
F(n) &= F(n-1) + 3n + 2 \\
&= \frac{3}{2}(n-1)^2 + \frac{7}{2}(n-1) - 4 + 3n + 2 \\
&= \frac{3}{2}(n^2 - 2n + 1) + \frac{7}{2}(n-1) - 4 + 3n + 2 \\
&= \frac{3n^2 - 6n + 3 + 7n - 7 - 8 + 6n + 4}{2} \\
&= \frac{3}{2}n^2 + \frac{7}{2}n - 4
\end{aligned}$$

Vamos mostrar que $F(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4 = \Theta(n^2)$.
Primeiro vamos mostrar que $F(n) = O(n^2)$.

$$\begin{aligned}
&F(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4 \\
&2F(n) = 3n^2 + 7n - 8 \\
&2F(n) \leqslant 3n^2 + 7n, \forall n \geqslant 0 \\
&2F(n) \leqslant 3n^2 + 7n^2 = 10n^2, \forall n \geqslant 0 \\
&F(n) \leqslant 5n^2, \forall n \geqslant 0
\end{aligned}$$

Logo $n_0 = 0$ e $c = 5$.
Vamos mostrar que $F(n) = \Omega(n^2)$.

$$F(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4 \geqslant \frac{3}{2}n^2, \forall n > 2$$

Logo $n_0 = 2$ e $c = \frac{3}{2}$.

$\square$

**Exemplo 3.2** Suponha a recorrência $F(n) = F\left(\frac{n}{2}\right) + 3$ com valor inicial $F(1) = 5$.

**Solução**:

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $F(n)$ | 5 | 8 | ND | 11 | ND | ND |

Podemos escrever $n = 2^k, k \in \mathbb{N}$.

$$F(n) = \begin{cases} F(1) = 5 \\ F(2^k) = F(2^{k-1}) + 3 \end{cases}$$

| $k$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| $F(2^k)$ | 5 | 8 | 11 | x |

$$\begin{aligned}
F(2^k) &= F(2^{k-1}) + 3 \\
&= F(2^{k-2}) + 2.3 \\
&= F(2^{k-3}) + 3.3 \\
&= F(2^{k-4}) + 4.3 \\
&= F(2^{k-5}) + 5.3
\end{aligned}$$

Continuando este processo até o expoente de 2, dentro de $F$, for igual a 0, temos:

$$
\begin{aligned}
F(2^k) &= F(2^{k-k}) + k.3 \\
&= F(1) + k.3 \\
&= 5 + 3k
\end{aligned}
$$

Escrevendo $F$ em função de $n$, temos:

$$
\begin{aligned}
n &= 2^k \\
\lg n &= \lg 2^k \\
\lg n &= k
\end{aligned}
$$

Portanto, $F(n) = 5 + 3\lg n$. ($\lg = \log_2$)

Mostre que $F(n) = 5 + 3\lg n = \Theta(\lg n)$
Vamos mostrar que $F(n) = O(\lg n)$.

$$
\begin{aligned}
F(n) &= 3\lg n + 5 \\
&\leqslant 3 + \lg n + 5\lg n \\
&= 8\lg n, \forall n > 1
\end{aligned}
$$

Vamos mostrar que $F(n) = \Omega(\lg n)$.

$$F(n) = 3\lg n + 5 \leqslant 3\lg n, \forall n \geqslant 1$$

$\square$

# Capítulo 4

# Algoritmos Recursivos

Algoritmos recursivos são aplicados em problemas com uma *estrutura recursiva.*

**Exemplo 4.1 (Soma)** Calcule a soma dos elementos do vetor $A[1 \ldots n]$ recursivamente, $n \geqslant 0$.

**Solução**: [1]

Soma elementos do vetor

```
função Soma(A,n)
    se n = 0 então
        devolve 0
    senão
        devolve Soma(A,n-1) + A[n]
    fim se
fim função
```

Aplicando a recursão no final de $A$. ($A[k \ldots n]$)

Soma recursiva

```
função Soma2(A,k,n)
    se k > n então
        devolve 0
    senão
        devolve Soma2(A,k+1,n) + A[k]
    fim se
fim função
```

□

[1] http://en.wikibooks.org/wiki/LaTeX/Algorithms_and_Pseudocode#While-loops

**Exemplo 4.2 (Máximo de um vetor)** Escreva um algoritmo iterativo e um recursivo para o seguinte problema: encontrar o valor de um maior elemento do vetor $A[1 \dots n]$.

**Solução**:
**Algoritmo iterativo**

Máximo

```
função Max(A,n)
    x ← A[1]
    para i ← 2 até n faça
        se A[i] > x então
            x ← A[i]
        fim se
        devolve x
    fim para
fim função
```

**Algoritmo recursivo**

Máximo recursivo

```
função MaxRec(A,n)
    se n = 1 então
        devolve A[1]
    senão
        x ← MaxRec(A,n-1)
        se A[n] > x então
            devolve A[n]
        senão
            devolve x
        fim se
    fim se
fim função
```

□

**Exemplo 4.3 (Busca simples em vetor)** Escreva um algoritmo iterativo e recursivo para resolver o problema de busca simples no vetor $A[1 \ldots n]$.

**Solução**:
**Algoritmo iterativo**

Busca iterativa

```
função Busca(A,n,x)
    i ← 1
    enquanto i ⩽ n e A[i] ≠ x faça
        i ← i + 1
        se i > n então
            devolve nao
        senão
            devolve sim
        fim se
    fim enquanto
fim função
```

**Algoritmo recursivo**

Busca recursiva

```
função BuscaRec(A,n,x)
    se n = 0 então
        devolve nao
    senão
        se A[n] = x então
            devolve sim
        senão
            devolve BuscaRec(A,n-1,x)
        fim se
    fim se
fim função
```

Uma outra forma de busca pode ser dividindo o vetor ao meio. Considere $n = 2^i$.

Considere um vetor onde o 1º elemento é $k$ e o ultimo é $n$.

Note que o tamanho do vetor é $n - k + 1$.

Busca recursiva dividindo o vetor ao meio

```
função BuscaRec2(A,k,n,x)
    se n − k + 1 = 0 então
    senão
        se A[n] = x então
            devolve sim
        senão
            devolve nao
        fim se
        devolve (BuscaRec(A,k,k + (n−k+1)/2 − 1,x))
        BuscaRec(A,k + (n−k+1)/2,n,x)
    fim se
fim função
```

□

**Exemplo 4.4 (Remoção em um vetor)** Remoção em um vetor $A[1 \ldots n]$.

**Solução**:

Remoção recursiva

```
função RemoveRec(A,k,n)
    se k = n então
        devolve n − 1
    senão
        A[k] ← A[k + 1]
    fim se
    devolve RemoveRec(A,k+1,n)
fim função
```

□

**Exemplo 4.5 (Inserção)** Inserção em um vetor $A[1 \ldots n]$.

**Solução**:

Inserção recursiva

```
função InsereRec(k,y,A,n)
    se k = n + 1 então
        A[n + 1] = y
    senão
        A[n + 1] = A[n]
    fim se
    InsereRec(k,y,A,n-1)
    devolve n + 1
fim função
```

□

## 4.1 Exercícios

**Exercício 2 (Fatorial)** Escreva um algoritmo recursivo que recebe um valor inteiro $n$ e devolve $n!$.

$$n! = \begin{cases} 1 & \text{, se } n = 0 \\ n * (n-1) & \text{, se } n > 0 \end{cases}$$

**Solução**:

```
função Fatorial(n)
    se n = 0 então
        devolve 1
    senão
        devolve n* Fatorial(n - 1)
    fim se
fim função
```

□

**Exercício 3 (Fibonacci)** Escreva um algoritmo recursivo que recebe um valor inteiro $n$ e devolve `fib(n)`, onde

$$\mathtt{fib(n)} = \begin{cases} 0 & \text{, se } n = 0 \\ 1 & \text{, se } n = 1 \\ \mathtt{fib(n - 1)} + \mathtt{fib(n - 2)} & \text{, se } n > 1 \end{cases}$$

**Solução**:

```
função FIB(n)
    se n = 0 então
        devolve 0
    senão
        se n = 1 então
            devolve 1
        senão
            devolve fib(n - 1) + fib(n - 2)
        fim se
    fim se
fim função
```

□

**Exercício 4 (Torre de Hanói)** O problema consiste de três pinos *Origem, auxiliar, destino* e uma torre com $n$ discos no pino *Origem* para o pino *Destino* usando o pino *Auxiliar* como auxiliar.

**Solução**:

```
função MOVETORRE(n,origem,destino,auxiliar)
    se n = 1 então
        escreva "mova disco de ", origem, "para ", destino
    senão
        moveTorre(n - 1,origem,auxiliar,destino)
        escreva "mova disco de ", origem, "para ", destino
        moveTorre(n - 1,auxiliar,destino,origem)
    fim se
fim função
```

□

## 4.2 Ordenação

**Exemplo 4.6** Rearranjar um vetor de inteiros $a[1 \dots n]$ de modo que ele fique em ordem crescente.

**Solução**:

Ordenação por Inserção

**função** ORDENACAOINSER(A,n)<br>
    **para** j $\leftarrow$ 2 **até** n **faça** $\triangleright\ n$<br>
        $x \leftarrow a[j]$ $\triangleright\ n-1$<br>
        $i \leftarrow j-1$ $\triangleright\ n-1$<br>
        **enquanto** $i > 0$ e $A[i] > x$ **faça** $\triangleright\ \dfrac{(n-1)(n-2)}{2}$<br>
            $A[i+1] \leftarrow A[i]$ $\triangleright\ \dfrac{n(n-1)}{2}$<br>
            $i \leftarrow i-1$ $\triangleright\ \dfrac{n(n-1)}{2}$<br>
        **fim enquanto**<br>
        $A[i+1] \leftarrow x$ $\triangleright\ n-1$<br>
    **fim para**<br>
**fim função**

Calculo do tempo:

$$T(n) = n + n - 1 + n - 1 + \frac{(n-1)(n+2)}{2} + n(n-1) + n - 1$$
$$T(n) = \frac{3}{2}n^2 + \frac{7}{2}n - 4$$

Mostrar que $T(n) = \Theta(n^2)$. □

## 4.3 Ordenação por Inserção Recursiva

**Solução**:

Ordenação por Inserção Recursiva

```
função OrdenacaoInserRec(A,n)
    se n > 1 então                              ▷ 1
        OrdenacaoInserRec(A,n-1)                ▷ T(n − 1)
        x ← A[n]                                ▷ 1
        i ← n − 1                               ▷ 1
        enquanto i > 0 e A[i] > x faça          ▷ n
            A[i + 1] ← A[i]                     ▷ n − 1
            i ← i − 1                           ▷ n − 1
        fim enquanto
        A[i + 1] ← x                            ▷ 1
    fim se
fim função
```

$$\begin{cases} T(1) = 1 \\ T(n) = T(n-1) + 3n + 2 \end{cases}$$

Temos que $T(n) = an^2 + bn + c$.

| $n$ | 1 | 2 | 3 |
|---|---|---|---|
| $T(n)$ | 1 | 9 | 20 |

$$\begin{cases} T(1) = a + b + c = 1 \\ T(2) = 4a + 2b + c = 9 \\ T(3) = 9a + 3b + c = 20 \end{cases}$$

Resolvendo o sistema linear, temos que $a = \dfrac{3}{2}, b = \dfrac{7}{2}$ e $c = -4$.

Portanto, $T(n) = \dfrac{3}{2}n^2 + \dfrac{7}{2}n - 4$. □

**Exemplo 4.7 (Ordenação de um vetor de inteiros)** Considere um algoritmo com a seguinte ideia: no início de cada iteração $A[i \dots j{-}1]$ está em ordem crescente e contém os elementos pequenos de $A$; o vetor $A[j \dots n]$ contém os elementos grandes. Escreva uma versão iterativa do algoritmo.

**Solução**:

Dica: suponha um algoritmo que encontre um menor elemento no vetor $A[p \dots q]$, `Min(A,p,q)`.

Suponha um algoritmo que troque de posição dois elementos de $A$, `Troque(A,p,q)`.

---

Ordenação por Seleção

---

**função** ORDSELEC(A,n) ▷ $T(n)$<br>
    $k \leftarrow$ `Min(A,1,n)` ▷ $3(n)$<br>
    `Troca(A,1,k)` ▷ $3$<br>
    **para** j $\leftarrow$ 2 **até** n - 1 **faça** ▷ $n-1$<br>
        $k \leftarrow$ `Min(A,j,n)` ▷ $(n-2)T'(n-j+1)(n-2).3(n-j+1) \leqslant (n-2).3n$<br>
        `Troca(A,j,k)` ▷ $(n-2).3$<br>
    **fim para**<br>
**fim função**

---

**função** TROCA(A,p,q)<br>
    $x \leftarrow A[p]$<br>
    $A[p] \leftarrow A[q]$<br>
    $A[q] \leftarrow x$<br>
**fim função**

$T(n) = 3$

**função** MIN(A,p,q) ▷ $T(n')$<br>
    $k \leftarrow p$ ▷ $1$<br>
    **para** j $\leftarrow$ p + 1 **até** q **faça** ▷ $n'$<br>
        **se** $A[j] < A[k]$ **então** ▷ $n'-1$<br>
            $k \leftarrow j$ ▷ $n'-1$<br>
        **fim se**<br>
        **devolve** $k$ ▷ $1$<br>
    **fim para**<br>
**fim função**

$T(n') = 3n'$

Então,

$$T(n) \leqslant 3n + 3 + n - 1 + 3n^2 - 6n + 3n - 6$$
$$T(n) \leqslant 3n^2 + n - 4$$

□

**Exemplo 4.8 (Algoritmo MergeSort)** Ordenar um vetor de inteiros $A[p \ldots r]$.

**Solução**:

MergeSort

**função** MERGESORT(A,p,r) $\triangleright\ T'(n)$
  **se** $p < r$ **então** $\triangleright\ 1$
    $q \leftarrow \left\lfloor \frac{p+r}{2} \right\rfloor$ $\triangleright\ 1$
    `MergeSort(A,p,q)` $\triangleright\ T'(\lceil n/2 \rceil)$
    `MergeSort(A,q+1,r)` $\triangleright\ T'(\lfloor n/2 \rfloor)$
    `Intercala(A,p,q,r)` $\triangleright\ 6n+5$
  **fim se**
**fim função**

**função** INTERCALA(A,p,q,r) $\triangleright\ T(n)$
  **para** i $\leftarrow$ p **até** q **faça** $\triangleright\ n+2$
    $B[i] \leftarrow A[i]$ $\triangleright\ n$
  **fim para**
  **para** j $\leftarrow$ q+1 **até** r **faça**
    $B[r+q+1-j] \leftarrow A[j]$
  **fim para**
  $i \leftarrow p$ $\triangleright\ 1$
  $j \leftarrow r$ $\triangleright\ 1$
  **para** k $\leftarrow$ p **até** r **faça** $\triangleright\ n+1$
    **se** $B[i] \leqslant B[j]$ **então** $\triangleright\ n$
      $A[k] \leftarrow B[i]$ $\triangleright\ 2n$
      $i \leftarrow i+1$
    **senão**
      $A[k] \leftarrow B[j]$
      $j \leftarrow j-1$
    **fim se**
  **fim para**
**fim função**

$T(n) = 6n+5$.

$$\begin{cases} T'(1) = 1 \\ T'(n) = T'(\lceil n/2 \rceil) + T'(\lfloor n/2 \rfloor) + 6n + 7 \end{cases}$$

Depois de simplificar $T'$, temos

$$T'(n) = 2T'(n/2) + 6n + 7$$
$$T'(n) \leqslant cn \lg n, n \geqslant n_0$$

| $n$ | 1 | 2 | 3 |
|---|---|---|---|
| $T'(n)$ | 1 | 21 | 47 |

Temos, $T'(n) \leqslant 16n \lg n$ □

**Exemplo 4.9 (Algoritmo QuickSort)** Ordenar um vetor de inteiros $A[p \dots r]$. Usar a estratégia de divisão e conquista.

**Solução**:

Separe

```
função Separe(A,p,r)                              ▷ T(n) = 5n + 9
    x ← A[p]
    i ← p − 1
    j ← r + 1
    enquanto 0 = 0 faça
        repita
            j ← j − 1
        até que A[j] ⩽ x
        repita
            i ← i + 1
        até que A[i] ⩽ x
        se i < j então
            Troca(A,i,j)
        senão
            devolve j
        fim se
    fim enquanto
fim função
```

QuickSort

```
função QuickSort(A,p,r)
    se p < r então
        q ← Separe(A,p,r)
        QuickSort(A,p,q)
        QuickSort(A,q+1,r)
    fim se
fim função
```

□

# Exemplos de Pseudocódigo no LaTeX

Valor Absoluto

```
função Absoluto(x)
    se x < 0 então
        devolve −x
    senão
        devolve x
    fim se
fim função
```

Exemplo do `for`

```
para i ← 1 até n faça
    A[i] ← i + 1                    ▷ Preenche o vetor
fim para
```

Exemplo do `while`

```
enquanto i ⩽ n faça
    i ← i + 1
fim enquanto
```

# Referências Bibliográficas

[1] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2001.

[2] S. Dasgupta, C. Papadimitriou, and U. Vazirani. Algorithms. 2006.

[3] P. Feofiloff. *Algoritmos em linguagem C*. Elsevier Brazil, 2009.

# Índice Remissivo