

Universidade Tecnológica Federal do Paraná (UTFPR)  
Departamento Acadêmico de Informática (DAINF)  
Estrutura de Dados I  
Professor: Rodrigo Minetto  
Lista de exercícios (complexidade de algoritmos)

---

**Exercícios (seleção): necessário entregar **TODOS** (moodle)!**

---

---

**Exercício 1)** Sejam as funções:

- $f(n) = n + \lg n$
- $g(n) = 5$
- $h(n) = n \lg n$
- $l(n) = n^2 + h(n)$ .

Marque V ou F.

- ( )  $2n \in O(n)$
- ( )  $n^2 \in O(n)$
- ( )  $n^2 \in O(n \lg n)$
- ( )  $f(n) \in O(l(n))$
- ( )  $g(n) \in \Omega(f(n))$
- ( )  $h(n) \in O(n^2)$
- ( )  $l(n) \in \Omega(h(n))$
- ( )  $g(n) \in O(1)$
- ( )  $l(n) \in \Theta(n^2)$

---

**Exercício 2)** Julgue os itens da coluna a direita como verdadeiro (V) ou falso (F) baseado nos conceitos vistos na aula de notação assintótica:

Q1 ( )	$f(n) = 20501$	$g(n) = 1$	$f(n) \in O(g(n))?$
Q2 ( )	$f(n) = n^2 + n$	$g(n) = 0.000001n^3$	$f(n) \in \Omega(g(n))?$
Q3 ( )	$f(n) = 2^{2n} + 1000$	$g(n) = 4^n + n^{100}$	$f(n) \in O(g(n))?$
Q4 ( )	$f(n) = \log(n^{100})$	$g(n) = n \log n$	$f(n) \in \Theta(g(n))?$
Q5 ( )	$f(n) = n \log n + 3^n + n$	$g(n) = n^2 + n + \log n$	$f(n) \in \Omega(g(n))?$
Q6 ( )	$f(n) = n \log n + n^2$	$g(n) = \log n + n^2$	$f(n) \in \Theta(g(n))?$
Q7 ( )	$f(n) = n \log n$	$g(n) = (\log n)^2$	$f(n) \in O(g(n))?$

---

**Exercício 3)** Determine a complexidade dos fragmentos de código abaixo, utilizando a notação assintótica  $\mathcal{O}$ :

```

i)   int i, soma = 0;
      for (i = 0; i < n; i++)
          soma++;

ii)  int i, j, soma = 0;
      for (i = 0; i < n; i++)
          for (j = 0; j < n; j++)
              soma++;

iii) int i, j, soma = 0;
      for (i = 0; i < n; i++)
          for (j = 0; j < n*n; j++)
              soma++;

```

---

**Exercício 4)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Não use o computador para simular (faça teste de mesa).

```

int func (int V[], int n, int k) {
    int i;
    for (i = 0; i < n; i++)
        if (V[i] == k)
            return i;
    return -1;
}

```

---

**Exercício 5)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Não use o computador para simular (faça teste de mesa).

```

int func (int n) {
    int i = 2;
    while ((i * i) <= n)
        if ((n % i) == 0)
            return 0;
        else
            i++;
    return 1;
}

```

---

**Exercício 6)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Suponha que os valores em  $V$  são aleatórios. Não use o computador para simular.

```

int func (int V[], int n) {
    int i, j, k = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if ((V[j] < V[i]) && (V[j] < V[k]))
                k = j;
    return k;
}

```

Você consegue otimizar o código acima para reduzir a complexidade?

---

**Exercício 7)** Coloque os algoritmos e funções a seguir em ordem crescente assintoticamente (do menor para o maior tempo). Se duas funções/algoritmos são da mesma ordem, indique que elas (eles) são iguais (note que  $\lg n$  é equivalente a  $\log_2$ ):

$2^n$ ,  $n - n^2 + 5n^3$ ,  $2^{n+1}$ ,  $\lg n$ ,  $n^3$ ,  $n \lg n$ ,  $n^2$ ,  $\sqrt{n}$ , 42,  $n$ ,  $(3/2)^n$ ,  $n!$ ,  $n^3 + \lg n$ , e  $4^{\lg n}$ .

---

**Exercício 8)** Os algoritmos  $W$ ,  $X$ ,  $Y$  e  $Z$  possuem tempo de execução no pior caso de  $20n \log_{10} n$ ,  $5n^2$ ,  $0.005n^3$  e  $500n$ , respectivamente. Responda as seguintes questões:

- Qual a notação assintótica destes quatro algoritmos?
- Utilizando a resposta da questão anterior, qual o ordem destes quatro algoritmos (do melhor para o pior).
- Utilizando o custo exato de cada algoritmo (não na forma assintótica), qual o ordem destes quatro algoritmos, do melhor para o pior, para 30 elementos?
- Utilizando o custo exato de cada algoritmo (não na forma assintótica), qual o ordem destes quatro algoritmos, do melhor para o pior, para 100.000 elementos?

---

**Exercícios (aprofundamento): não é necessário entregar mas é importante estudar!**

---

---

**Exercício 9)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Não use o computador para simular.

```
int func (int V[], int n, int k) {
    int i = 0;
    int j = n-1;
    while (i <= j) {
        int t = (i + j)/2;
        if (V[t] == k)
            return t;
        else if (V[t] < k)
            i = t + 1;
        else
            j = t - 1;
    }
    return -1;
}
```

Suponha que  $V$  tenha os valores em ordem crescente.

---

**Exercício 10)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Não use o computador para simular.

```

int func (int x, int y, int z) {
    if (x >= y) {
        if (x >= z)
            return x;
        else
            return z;
    }
    else {
        if (y >= z)
            return y;
        else
            return z;
    }
}

```

---

**Exercício 11)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Não use o computador para simular.

```

void func (int A[][n], int B[][n], int C[][n]) {
    int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            C[i][j] = A[i][j] + B[i][j];
}

```

---

**Exercício 12)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Não use o computador para simular.

```

void func (int A[][n], int B[][n], int C[][n]) {
    int i, j, k;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            C[i][j] = 0;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            for (k = 0; k < n; k++)
                C[i][j] += A[i][k] * B[k][j];
}

```

---

**Exercício 13)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Não use o computador para simular.

```

int func (int x, int n) {
    int r = 1, i;
    for (i = 0; i < n; i++)
        r = r * x;
    return r;
}

```

Você consegue otimizar o código acima para reduzir a complexidade?

---

**Exercício 14)** Descreva a funcionalidade do fragmento de código a seguir e a complexidade associada utilizando a notação assintótica. Não use o computador para simular.

```
int func (char *x, char *y) {
    if (strlen(x) != strlen(y))
        return 0;
    for (int i = 0; i < strlen(x); i++) {
        for (int j = 0; j < strlen(y); j++) {
            if (x[i] == y[j]) {
                y[j] = ' ';
                break;
            }
        }
    }
    for (int j = 0; j < strlen(y); j++)
        if (y[j] != ' ')
            return 0;
    return 1;
}
```

Você consegue otimizar o código acima para reduzir a complexidade?

---

**Exercício 15)** Ordene as funções a seguir por ordem crescente de complexidade.

$$f_1(n) = 2^{2^{1000000}} \quad f_2(n) = 2^{100000n} \quad f_3(n) = \binom{n}{2} \quad f_4(n) = n\sqrt{n}$$

---

**Exercício 16)** Analise o custo computacional dos algoritmos a seguir, que calculam o valor de um polinômio de grau  $n$ , da forma:  $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$ , onde os coeficientes são números de ponto flutuante armazenados no vetor  $a[0..n]$ , e o valor de  $n$  é maior que zero. Todos os coeficientes podem assumir qualquer valor, exceto o coeficiente  $a_n$  que é diferente de zero.

Algoritmo 1:

```
soma = a[0]
Repita para i = 1 até n
    Se a[i] != 0.0 então
        potência = x
        Repita para j = 2 até i
            potência = potência * x
        Fim repita
        soma = soma + a[i] * potencia
    Fim se
Fim repita
Imprima(soma)
```

Algoritmo 2:

```
soma = a[n]
Repita para i = n-1 até 0 passo -1
    soma = soma * x + a[i]
Fim repita
Imprima(soma)
```

Com base nos algoritmos 1 e 2, avalie as asserções a seguir e a relação proposta entre elas.

**I. Os algoritmos possuem a mesma complexidade assintótica.**

Porque

**II. Para o melhor caso, ambos os algoritmos possuem complexidade  $O(n)$**

A respeito dessas asserções, assinale a opção correta.

- a) As asserções I e II são proposições verdadeiras, e a II é uma justificativa correta da I.
- b) As asserções I e II são proposições verdadeiras, mas a II não é uma justificativa correta da I.
- c) A asserção I é uma proposição verdadeira, e a II é uma proposição falsa.
- d) A asserção I é uma proposição falsa, e a II é uma proposição verdadeira.
- e) As asserções I e II são proposições falsas.