UFRN
UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

**CENTRO DE TECNOLOGIA**
**DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO**

Disciplina: Projeto e Engenharia de Software
Professor: Eduardo de Lucena Falcão
Tópicos: Notações Assintóticas e Complexidade de Algoritmos Iterativos

1. Defina e explique o significado das notações $O$, $\Omega$, $o$, $\omega$ e $\Theta$, para o tempo de execução de algoritmos.
   *Essas notações são utilizadas, em contextos de tempo de execução de algoritmos, para explicar relações entre duas funções:*
   - *Notação O (big-O): uma função qualquer $f(n)$ será limitada superiormente por uma função qualquer $g(n)$ se existirem constantes positivas $c$ e $n_0$ tal que $f(n) \leq c \cdot g(n)$ para todo $n \geq n_0$. Assim sendo, dizemos que $f(n) \in O(g(n))$ .*
   - *Notação Ω (big-ômega): uma função qualquer $f(n)$ será limitada inferiormente por uma função qualquer $g(n)$ se existirem constantes positivas $c$ e $n_0$ tal que $f(n) \geq c \cdot g(n)$ para todo $n \geq n_0$. Assim sendo, dizemos que $f(n) \in \Omega(g(n))$ .*
   - *Notação Θ: uma função qualquer $f(n)$ será limitada inferiormente e superiormente por uma função qualquer $g(n)$ se existirem constantes positivas $c_1$, $c_2$ e $n_0$ tal que $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ para todo $n \geq n_0$. Assim sendo, dizemos que $f(n) \in \Theta(g(n))$, ou ainda $f(n) \in \Theta(g(n))$ se $f(n) \in O(g(n))$ e $f(n) \in \Omega(g(n))$.*
   - *Notações o (little-o) e ω (little-ômega): essas notações são parecidas às $O(g(n))$ e $\Omega(g(n))$, mas essas primeiras são limites assintóticos inferior e superior que podem ser folgados ou apertados. Para limites inferior e superior que são sempre folgados utilizamos as letras minúsculas $o(g(n))$ e $\omega(g(n))$. Sendo assim: $f(n) \in o(g(n))$ se existirem constantes positivas $c$ e $n_0$ tal que $f(n) < c \cdot g(n)$ para todo $n \geq n_0$; e $f(n) \in \omega(g(n))$ se existirem constantes positivas $c$ e $n_0$ tal que $f(n) > c \cdot g(n)$ para todo $n \geq n_0$.*
2. Prove ou mostre um contra-exemplo para cada uma das seguintes afirmações.
   a. se f(n) ∈ O(g(n)) então g(n) ∈ O(f(n))
      f(n) = n, g(n)=n² ⇒ n ∈ O(n²); n² ∉ O(n)
   b. f(n) + g(n) ∈ Θ(min(f(n),g(n)))
      $f(n) = n, g(n) = n^2$
      $min(f(n), g(n)) = min(n, n^2) = n$
      $f(n) + g(n) = n^2 + n \notin \Theta(n)$
   c. se f(n) ∈ O(g(n)) então g(n) ∈ Ω(f(n))
      $f(n) = n^2, g(n) = n^2 \Rightarrow n^2 \in O(n^2) \Rightarrow n^2 \in \Omega(n^2)$

3. Explique por que a afirmação "o tempo de execução do algoritmo A é pelo menos $O(n^2)$ não faz sentido".

   *Não faz sentido pois a notação big-O já representa o pior caso, então não tem como ser "pelo menos" $O(n^2)$, uma vez que isso dá a entender que ainda há um caso pior.*

4. Prove que:
   a. $8n + 128 = O(n)$

      $f(n) = 8n + 128,\ g(n) = n$

      $f(n) \leq c \cdot g(n) \Rightarrow 8n + 128 \leq cn,\ para\ c = 9$

      $\Rightarrow (9 - 8)n \geq 128 \Rightarrow n = 128 \Rightarrow n_0 = 128$

      *Sendo assim, para* $c = 9$ *e* $n_0 = 128$, *foi demonstrado que* $f(n) = 8n + 128$ *é limitado superiormente por* $g(n) = n$

   b. $8n + 128 = O(n^2)$

      $f(n) = 8n + 128,\ g(n) = n^2$

      $f(n) \leq c \cdot g(n) \Rightarrow 8n + 128 \leq cn^2,\ para\ c = 1$

      $\Rightarrow n^2 - 8n - 128 \geq 0 \Rightarrow n' \leq (-8)\ e\ n'' \geq 16 \Rightarrow n_0 = 16$

      *Sendo assim, para* $c = 1\ e\ n_0 = 16$, *foi demonstrado que* $f(n) = 8n + 128$ *é limitado superiormente por* $g(n) = n^2$

   c. $n^2/2 - 3n = \Theta(n^2)$

      $f(n) = \frac{n^2}{2} - 3n,\ g(n) = n^2$

      $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \Rightarrow c_1 n^2 \leq \frac{n^2}{2} - 3n \leq c_2 n^2$

      $para\ c_1 = \frac{1}{4}\ e\ c_2 = \frac{1}{2},\ temos:$

      $\frac{1}{4}n^2 \leq \frac{1}{2}n^2 - 3n \leq \frac{1}{8}n^2$

      $\Rightarrow (\frac{1}{4} - \frac{1}{2}) + 3n \leq 0 \Rightarrow -\frac{1}{4}n^2 + 3n \leq 0 \Rightarrow n(\frac{1}{4}n - 3) \geq 0 \Rightarrow n' \geq 0,\ n'' \geq 12$

      $\Rightarrow \frac{1}{2}n^2 - 2n \leq \frac{1}{2}n^2 \Rightarrow 2n \geq 0 \Rightarrow n''' \geq 0$

      *Sendo assim, para* $c_1 = \frac{1}{4}$, $c_2 = \frac{1}{2}$ *e* $n_0 = 12$, *foi demonstrado que* $f(n) = \frac{n^2}{2} - 3n$ *é limitado superiormente e inferiormente por* $g(n) = n^2$

   d. $5n^3 + 3n = \Omega(n^2)$

5. Analise detalhadamente a complexidade dos seguintes algoritmos e responda as perguntas de forma objetiva.

   a. busca

```
int busca(int v[], int tamanho, int val) {
        for (int i = 0; i < tamanho; i++) {
                if (v[i] == val)
                        return i;
        }
        return -1;
}
```

- Quantas vezes são executadas as seguintes instruções?

| | |
|---|---|
| int i = 0 | *1* |
| i < tamanho | *tamanho + 1* |
| i++ | *tamanho* |
| if (v[i] == val) | *tamanho* |
| return i | *1* |
| return -1 | *1* |

- Qual a complexidade de tempo no pior caso e no melhor caso?
  *Pior caso: O(n); melhor caso:* $\Omega(1)$

   b. insertion-sort

```
void insertionSort(int* v, int tamanho) {
   for (int i = 1; i < tamanho; i++) {
        for (int j = i; j > 0 && v[j - 1] > v[j]; j--) {
                int temp = v[j - 1];
                v[j - 1] = v[j];
                v[j] = temp;
        }
   }
}
// analisando j (para as comparações):
   // quando i = 1 ⇒ j ∈ {1, 0} ⇒ 2 comparações
   // quando i = 2 ⇒ j ∈ {2, 1, 0} ⇒ 3 comparações
   // quando i = 3 ⇒ j ∈ {3, 2, 1, 0} ⇒ 4 comparações
   // ...
   // quando i = n-2 ⇒ j ∈ {n − 2, n − 3, ..., 1, 0} ⇒ n-1 comparações
   // quando i = n-1 ⇒ j ∈ {n − 1, n − 2, ..., 1, 0} ⇒ n comparações

⇒ 2 + 3 + 4 + ... + n comparações (PA)
```

$\Rightarrow S_n = \frac{n(a_1+a_n)}{2} = \frac{n(2+n)}{2} = \frac{n^2+2n}{2}$

```
// analisando j--:
  // quando i = 1 ⇒ 1 execução
  // quando i = 2 ⇒ 2 execuções
  // quando i = 3 ⇒ 3 execuções
  // ...
  // quando i = n-2 ⇒ n-2 execuções
  // quando i = n-1 ⇒ n-1 execuções

⇒ 1 + 2 + ... + n-2 + n-1 execuções (PA)
```

$\Rightarrow S_n = \frac{n(a_1+a_n)}{2} = \frac{n(1+n-1)}{2} = \frac{n^2}{2}$

- Quantas vezes são executadas as seguintes instruções?

| | |
|---|---|
| int i = 1 | *1* |
| i < tamanho | *tamanho* |
| i++ | *tamanho - 1* |
| int j = i | *tamanho - 1* |
| j > 0 && v[j - 1] > v[j] | *(tamanho²+2 tamanho)/2* |
| j-- | *tamanho²/2* |
| int temp = v[j - 1] | *tamanho²/2* |
| v[j - 1] | *tamanho²/2* |
| v[j] = temp | *tamanho²/2* |

- Qual a complexidade de tempo da função **insertionSort** no pior caso e no melhor caso?
  *Pior caso: $O(n^3)$; melhor caso: $\Omega(n^2)$*

c. selection-sort

```
void troca(int* v, int i, int j) {
    int temp = v[i];
    v[i] = v[j];
    v[j] = temp;
} // custo c. O(1), Ω(1)

void selectionSort(int* v, int tamanho) {
    for (int i = 0; i < (tamanho - 1); i++) {
        int iMenor = i;
        for (int j = i+1; j < tamanho; j++) {
            if (v[j] < v[iMenor])
                iMenor = j;
        }
        troca(v, i, iMenor);
    }
}

// int i = 0 ⇒ 1 execução
// i < (n-1) ⇒ n execuções
// i++ ⇒ n-1 execuções
// int iMenor = i  ⇒ n-1 execuções
// int j = i+1 ⇒ n-1 execuções
// analisando j < n:
  // quando i=0 ⇒ j ∈ {1, 2, ..., n − 1, n} ⇒ n execuções
  // quando i=1 ⇒ j ∈ {2, 3, ..., n − 1, n} ⇒ n-1 execuções
  // quando i=2 ⇒ j ∈ {3, 4, ..., n − 1, n} ⇒ n-2 execuções
  // quando i=n-2 ⇒ j ∈ {n − 1, n} ⇒ 2 execuções
  // ⇒ 2, 3, 4, …, n-1, n (execuções)
  // ⇒ s_n = n(a_1+a_n)/2 = n(2+n)/2 = (n²+2n)/2 execuções
  // Total: parcelaForDeFora x parcelaForDeDentro
  // Total: (n − 1) · (n²+2n)/2 = (n³+2n²−n²−2n)/2 = (n³+n²−2n)/2
// analisando j++, if (v[j] < v[iMenor]) e iMenor = j:
  // quando i=0 ⇒ n-1 execuções
  // quando i=1 ⇒ n-2 execuções
  // quando i=2 ⇒ n-3 execuções
  // quando i=n-2 ⇒ 1 execução
  // ⇒ 1, 2, 3, …, n-2, n-1 (execuções)
  // ⇒ s_n = n(a_1+a_n)/2 = n(1+n−1)/2 = n²/2 execuções
  // Total: parcelaForDeFora x parcelaForDeDentro
  // Total: (n − 1) · n²/2 = (n³−n²)/2 execuções cada
// analisando troca(v, i, iMenor):
  // Total: custoDaFunção x quantidadeDeRepDoForDeFora
  // Total: c · (n − 1) = cn − c = O(n)
```

- Quantas vezes são executadas as seguintes instruções?

| | |
|---|---|
| int i = 0 | *1* |
| i < (tamanho - 1) | *n* |
| i++ | *n-1* |
| int iMenor = i; | *n-1* |
| int j = i+1 | *n-1* |
| j < tamanho | $(n^3+n^2-2n)/2$ |
| j++ | $(n^3-n^2)/2$ |
| if (v[j] < v[iMenor]) | $(n^3-n^2)/2$ |
| iMenor = j | $(n^3-n^2)/2$ |
| troca(v, i, iMenor) | *cn-c* |

- Qual a complexidade de tempo da função **selectionSort** no pior caso e no melhor caso?
  $O(n^3)$ e $\Omega(n^3)$.
- Qual a complexidade de tempo da função **troca** no pior caso e no melhor caso?
  $O(1)$ e $\Omega(1)$.

d. bubble-sort

```
void bubbleSort(int* v, int n) {
   for (int varredura = 0; varredura < n - 1; varredura++) {
      bool trocou = false;
      for (int i = 0; i < n - varredura - 1; i++) {
         if (v[i] > v[i + 1]) {
            troca(v, i, i+1);
            trocou = true;
         }
      }
      if (trocou == false)
         return;
   }
}
// analisando i<n-varredura -1:
   // quando v=0 ⇒ i ∈ {0, 1, 2, ..., n − 2, n − 1} ⇒ n execuções
   // quando v=1 ⇒ i ∈ {0, 1, 2, ..., n − 3, n − 2} ⇒ n-1 execuções
   // ...
   // quando v=n-2 ⇒ i ∈ {0, 1} ⇒ 2 execuções
   // ⇒ 2, 3, …, n-1, n (execuções)
```

```
  // ⇒ s_n = n(a_1+a_n)/2 = n(2+n)/2 = (n²+2n)/2 execuções
  // Total = execuçõesDoForDeFora x execuçõesDoForDeDentro
  // Total = (n − 1) · (n²+2n)/2 = (n³+2n²−n²−2n)/2 = (n³+n²−2n)/2
// analisando dentro do for:
  // quando v=0 ⇒ n-1 execuções
  // quando v=1 ⇒ n-2 execuções
  // ...
  // quando v=n-2 ⇒ i ∈ {0} ⇒ 1 execução
  // ⇒ 1, 2, 3, …, n-2, n-1 (execuções)
  // ⇒ s_n = n(a_1+a_n)/2 = n(1+n−1)/2 = n²/2 execuções
  // Total: parcelaForDeFora x parcelaForDeDentro
  // Total: (n − 1) · n²/2 = (n³−n²)/2 execuções cada
```

- Quantas vezes são executadas as seguintes instruções?

| | |
|---|---|
| int varredura = 0 | *1* |
| varredura < n - 1 | *n* |
| varredura++ | *n-1* |
| bool trocou = false | *n-1* |
| int i = 0 | *n-1* |
| i < n - varredura - 1 | $(n^3+n^2-2n)/2$ |
| i++ | $(n^3-n^2)/2$ |
| if (v[i] > v[i + 1]) | $(n^3-n^2)/2$ |
| troca(v, i, i+1) | $(n^3-n^2)/2$ |
| trocou = true; | $(n^3-n^2)/2$ |
| if (trocou == false) | *n-1* |
| return | *1* |

- Qual a complexidade de tempo da função **bubbleSort** no pior caso e no melhor caso?
  $O(n^3)$ e $\Omega(n^3)$.

e. foo

```
void foo(int n) {
   int c = 0;
   for (int i = n; i >= 1; i = i/2) {
      for (int j = 0; j < i; j++) {
         c++;
      }
   }
   return c;
}
// analisando i>=1
   // quando i=n ⇒ i ∈ {n, n/2, n/4, ..., 1}
   // S_n = (a_1·(q^n−1))/(q−1) = (1·((1/2^n)−1))/((1/2)−1) = ((1−2^n)/2^n)/(−1/2) = (1−2^n)/2^n · −1/2 = (2^n−1)/2^(n+1) = 2^n/2^(n+1) − 1/2^(n+1)
   // S_n = 2^(n−n−2) − 2^(−n−1) = 2^(−2) − 2^(−n−1)
```

- Quantas vezes são executadas as seguintes instruções?

| | |
|---|---|
| int c = 0 | *1* |
| int i = n | *1* |
| i >= 1 | |
| i = i/2 | |
| j < i | |
| j++ | |
| c++ | |
| return c | |

- Qual a complexidade de tempo da função **foo** no pior caso e no melhor caso?