Complexidade de Algoritmos

Mariana Kolberg

Análise da complexidade pessimista

Plano

Uma hierarquia de abstrações:

Tempo de execução [s] $\longrightarrow$ Número de operações $\longrightarrow$ Operações no caso pessimista $\longrightarrow$ Complexidade pessimista assintótica

Análise de Complexidade Pessimista - ACP

- A complexidade pessimista é o critério mais utilizado para verificar a eficiência de algoritmos
- Lembrando que

$$C_p^{=}[a](n) = \max\{desemp[a](d) \in R_+ \mid tam(d) = n\}$$

$$C_p^{\leq}[a](n) = \max\{desemp[a](d) \in R_+ \mid tam(d) \leq n\}$$

- Iremos estudar a complexidade das seguintes estruturas:
  - Atribuição : **v← e**
  - Seqüência : **S;T**
  - Condicional : **se b então S senão T (ou se b então S)**
  - Iteração definida : **para i de j até m faça S**
  - Iteração indefinida : **enquanto b faça S**

Partes Conjuntivas e Disjuntivas

- A complexidade pessimista de um algoritmo com partes conjuntivas é limitada por

$$\text{Máx}\,(c_P^{\leq}[a_1], c_P^{\leq}[a_2]) \leq c_P^{\leq}[a_0] \leq c_P^{\leq}[a_1] + c_P^{\leq}[a_2].$$

- Ou seja, tem ordem

$$\Theta(c_P^{\leq}[a_1] + c_P^{\leq}[a_2]).$$

- A complexidade pessimista de um algoritmo com partes disjuntivas é limitada por

$$\max(c_p^{\leq}[a_1](n), c_p^{\leq}[a_2](n)) \geq c_p^{\leq}[a_0](n) \geq \min(c_p^{\leq}[a_1](n), c_p^{\leq}[a_2](n))$$

Análise da Complexidade Pessimista

- Considere uma operação de **atribuição** v←e
  - Sua complexidade pessimista tem as seguintes cotas

$$\text{Máx}(c_P[e], c_P[\leftarrow_e]) \leq c_P[v \leftarrow e] \leq c_P[e] + c_P[\leftarrow_e]$$

  - A ordem da complexidade pessimista da atribuição é

$$c_p[v \leftarrow e] = O(c_p[e] + c_p[\leftarrow_e])$$

- Considere uma operação de **seqüência** S;T
  - A complexidade pessimista da seqüência tem cotas

$$\max(c_p[S](n), c_p[T](s(n))) \leq c_p[S;T](n) \leq c_p[S](n) + c_p[T](s(n))$$

  - A ordem da complexidade é

$$c_p[S;T](n) = O(c_p[S](n) + c_p[T](s(n)))$$

- Considere uma operação **condicional**
  - A complexidade pessimista da seqüência tem cotas

$$MiAO(c_P[S], c_P[T]) + c_p[b] \leq c_p[\text{se b então S senão T fim-se}] \leq c_p[b] + MxAO(c_p[S], c_p[T])$$

  - A ordem da complexidade é

$$c_P[\underline{\text{se}}\ b\ \underline{\text{então}}\ S\ \underline{\text{senão}}\ T] = O(c_P[b] + MxAO(c_P[S], c_P[T]))$$

ACP – Iteração Definida

- Considere a iteração **para i de j até m faça S** sobre uma entrada **d**

a) para i de 1 até 20 faça m ← m+1

A complexidade é O(20.1)=O(1)

b) para i de 1 até n faça s ← s+R[i]

A complexidade é O(n.1)=O(n)

c) para i de 1 até (n-1) faça Troca(A[i],A[i+1])

A complexidade é O((n-1).1) =O(n).

ACP – Iteração Definida

- Para a iteração **para i de j até m faça S** sobre uma entrada d faça

  $j=j(d)$; $m=m(d)$ e $N(d)=m-j+1$.

  S é executado N(d) vezes da seguinte forma:

  S é executado com i=j sobre a entrada d produzindo $S(d)$;

  S é executado com i:=j+1 sobre a entrada S(d) produzindo $S(S(d))=S^2(d)$;

  ...

  S é executado com i:=m sobre a entrada $S^{N(d)-1}(d)$ resultando em $S^{N(d)}(d)$.

- O desempenho é calculado levando em consideração a soma dos desempenhos de S para cada entrada **d, S(d), S^2(d),…, $S^{N(d)-1}$(d)**
- Logo, desemp[para i de j até m faça S](d) é dado pelo somatório:

$$\sum_{i=j(d)}^{m(d)} desemp[\ S\ ](S^{(i-j(d))}(d))$$

- Onde j(d) é o valor inicial da iteração, e i vai mudando a cada iteração

ACP – Iteração Definida

Iteração definida com preservação (assintótica) de tamanho,

- A complexidade de para i de j até m faça S tem cotas

$$c_p[S](n) \leq c_p[\text{para i de j até m faça S}](n) \leq N^*(n).c_p[S](n)$$

- Portanto, a ordem da complexidade pessimista é dada por

$$c_p[\text{para i de j até m faça S}] = O(N^*(n).c_p[S](n))$$

Onde $N(n) = \max\{\ N(d) \mid tam(d) \leq n\}$ (consideramos o n máximo de iterações)

$$N(d) = m(d) - j(d) + 1$$

$$N^*(n) = \max\{N(n), 0\}$$

ACP – Iteração Definida

Qual é a complexidade do trecho **para i de 1 até p faça Busca(A[i],v)**?

Onde **Busca** procura um valor na lista **v** com complexidade O(**log m**).

ACP – Iteração Definida

Qual é a complexidade do trecho **para i de 1 até p faça Busca(A[i],v)**?

Onde **Busca** procura um valor na lista **v** com complexidade O(**log m**).

A complexidade do algoritmo é O(p log m).

Considerando n=max(p,m) temos O(nlogn).

ACP – Iteração Definida

Considere a iteração definida **para i de j até m faça S** onde S preserva o tamanho da entrada **d**, i. e. tam(S(d))=tam(d) e C_p [S](n) é um polinômio de grau k, i. e., C_p [S](n)=O(n^k).

a) Neste caso, temos

c_P[para **i** de **j** até **m** faça S] = O(N^* (n).n^k)

b) No caso em que N(n) não depende de n teremos

c_P[para **i** de **j** até **m** faça S] = O(n^k)

c) No caso em que N(n) depende de n temos que avaliar N^* (n).

ACP – Iteração Definida

- A complexidade de **para i de j até m faça S** tem cotas

Superior: $\sum_{i=j(n)}^{m(n)} c_P\, [\, S\,](s^{(i-j(n))}(n))$

onde: $\mathbf{m}(n) := \text{Máx}\, \{\, \mathbf{m}(d) \,/\, \mathit{tam}(d) \leq n\}$

$s(n) := \text{Máx}\, \{\, \mathit{tam}(S(d)) \,/\, \mathit{tam}(d) \leq n\,\}$

$\mathbf{j}(n) := \text{mín}\, \{\mathbf{j}(d) \,/\, \mathit{tam}(d) \leq n\,\}$

Inferior : $\text{Máx}\, \{\, c_P\, [\, S\,](s^{(i-j(n))}(n)) \,/\, i = j(n), \ldots, m(n)\,\}$

A ordem da complexidade pessimista de **para i de j até m faça S**

$[\, \underline{\text{para}}\ i\ \underline{\text{de}}\ j\ \underline{\text{até}}\ m\ \underline{\text{faça}}\ S\,](n) = O(\sum_{i=j(n)}^{m(n)} c_P\, [\, S\,](s^{(i-j(n))}(n))\,)$

ACP – Iteração Definida

Considere a seqüência $(n\text{-}i)^k$ para i variando de 1 até n. Mostre que

$$\sum_{i=1}^{n}(n-i)^k = O(n^{k+1})$$

ACP – Iteração Definida

Considere a seqüência (n-i)^k para i variando de 1 até n. Mostre que

$$\sum_{i=1}^{n}(n-i)^k = O(n^{k+1})$$

Assumindo que $(n-i)^k \leq n^k$

ACP – Iteração Definida

Considere a seqüência $(n\text{-}i)^k$ para i variando de 1 até n. Mostre que

$$\sum_{i=1}^{n}(n-i)^k = O(n^{k+1})$$

Assumindo que $(n-i)^k \leq n^k$

temos $\displaystyle\sum_{i=1}^{n}(n-i)^k \leq \sum_{i=1}^{n}(n)^k = (n)^{k+1}$

Portanto

$$\sum_{i=1}^{n}(n-i)^k = O(n^{k+1})$$

ACP – Iteração Definida

- Considere o seguinte trecho **para i de 1 até n faça S**, onde Cp [S] é limitada por polinômio de grau k. Sabemos que tam(S(d)) = tam(d) - 1.
  - Como representar essa diferença de tamanho a cada iteração?
  - Qual é a complexidade do trecho ?

ACP – Iteração Definida

Considere o seguinte trecho **para i de 1 até n faça S**, onde Cp [S] é limitada por polinômio de grau k. Sabemos que tam(S(d)) = tam(d) – 1 (a cada iteração, o tamanho diminui em uma unidade). Qual é a complexidade do trecho ?

Neste caso, temos

Tam(d)-1=tam(d)-i

$$c_p[\text{para i de 1 até n faça S}](n) = O(\sum_{i=0}^{n-1}(n-i)^k)$$

ACP – Iteração Definida

Considere o seguinte trecho **para i de 1 até n faça S**, onde Cp [S] é limitada por polinômio de grau k. Sabemos que tam(S(d)) = tam(d) - 1. Qual é a complexidade do trecho ?

Neste caso, temos

$$c_p[\text{para i de 1 até n faça S}](n) = O(\sum_{i=0}^{n-1}(n-i)^k)$$

Pode-se mostrar que

$$\sum_{i=0}^{n-1}(n-i)^k = O(n^{k+1})$$

ACP – Iteração Definida

Considere o seguinte trecho **para i de 1 até n faça S**, onde Cp [S] é limitada por polinômio de grau k. Sabemos que tam(S(d)) = tam(d) - 1. Qual é a complexidade do trecho ?

Neste caso, temos

$$c_p[\text{para i de 1 até n faça S}](n) = O(\sum_{i=0}^{n-1}(n-i)^k)$$

Pode-se mostrar que

$$\sum_{i=0}^{n-1}(n-i)^k = O(n^{k+1})$$

Portanto, temos

$$c_P\ [\ \underline{\text{para}}\ \text{i}\ \underline{\text{de}}\ 1\ \underline{\text{até}}\ \text{n}\ \underline{\text{faça}}\ \text{S}\ ](n) = O(\ n^{k+1}\ )$$

ACP – Iteração Indefinida

As iterações indefinidas comumente usada são **enquanto** e **repita**.

a) i←0;

enquanto i<10 **faça** i←i+1;

A complexidade tem ordem constante e igual a O(1);

b) i←0;

enquanto i<n faça i←i+1;A[i]←0 **fim-enquanto**

A complexidade tem ordem linear igual a O(n);

c) t←1; i ←n;

enquanto i > 1 faça t←t+Q[i,i]; i←i-1 **fim-enquanto**

A complexidade tem ordem linear igual a O(n)

ACP – Iteração Indefinida

- A execução da iteração indefinida **enquanto b faça S** sobre a entrada d, itera a execução de S, sucessivamente como segue:
  - Se a entrada d satisfaz a condição **b** então executa-se S sobre **d** dando **S(d)**
  - Se S(d) satisfaz a condição **b** então executa-se S sobre **S(d)** dando $\mathbf{S^2(d)}$; etc.
  - Quando a condição b deixa de ser verdadeira sobre $\mathbf{S^{H(d)}(d)}$, a execução termina com $\mathbf{S^{H(d)}\ (d)}$ como saída.
- Um problema a ser resolvido é determinar H(d). Este valor faz com que a **iteração indefinida** passe a ser **definida** a tornando mais fácil de ser resolvida.

$$H(d) = \min\{i \in \mathbf{N} | \neg \mathbf{b}(\mathbf{S^i}(\mathbf{d}))\}$$

  - Menor valor para o qual a condição não é satisfeita

ACP – Iteração Indefinida

Este tipo de iteração pode ser descrito recursivamente como

$$\text{se } b \text{ então } (S; \text{enquanto } b \text{ faça } S) \text{ fim-se}$$

O desempenho desta estrutura pode ser descrito da seguinte maneira.

Caso b(d) seja falso:

$$desemp[\text{ enquanto b faça S }](d) = aval[b](d)$$

Caso b(d) seja verdadeiro

$$desemp[\text{ enquanto b faça S }](d) = aval[b](d) +$$

$$+ desemp[\text{ S }](d) + desemp[\text{ enquanto b faça S }](S(d))$$

ACP – Iteração Indefinida

Assim, o desempenho é dado por

$$\textit{desemp}[\ \underline{\text{enqto}}\ b\ \underline{\text{faça}}\ S\ ](d) = \textit{aval}\ [\ b\ ](S^{H(d)}(d)) + \\ + \sum_{i=0}^{H(d)-1} [\textit{aval}[b](S^{i}(d)) + \textit{desemp}[S](S^{i}(d))]$$

Como a complexidade das avaliações é dominada pela complexidade de S, temos:

$$\textit{desemp}[\ \underline{\text{enqto}}\ b\ \underline{\text{faça}}\ S\ ](d) = \sum_{i=0}^{H(d)-1} \textit{desemp}[\ S\ ](S^{i}(d)).$$

ACP – Iteração Indefinida

Considere a seguinte iteração indefinida

$$\text{enquanto } (A[i] = 0 \land 1 \leq i \leq n) \text{faça } A[i] \leftarrow A[i] + 1; i \leftarrow i + 1 \text{ fim-enquanto}$$

No pior caso **i** é inicializado com 1 e incrementado até n, gerando **n iterações.**

Como as instruções no corpo do enquanto têm complexidade constante.

A complexidade complexidade da iteração acima é igual a

n.1=O(n)

ACP – Iteração Indefinida

Considere a seguinte iteração

enquanto $1 \leq i \leq n$ faça Buscas$(A[i], B[1 \ldots i])$; $i \leftarrow i + 1$; fim-enquanto

Qual é complexidade do trecho sabendo que Buscas é linear no tamanho do vetor B?

ACP – Iteração Indefinida

Considere a seguinte iteração

enquanto $1 \leq i \leq n$ faça Buscas$(A[i], B[1 \ldots i])$; $i \leftarrow i + 1$; fim-enquanto

Qual é complexidade do trecho sabendo que Buscas é linear no tamanho do vetor B?

$$O(\sum_{k=1}^{n} k) = O(\frac{n(n+1)}{2}) = O(n^2)$$

ACP – Iteração Indefinida

A forma complexidade pessimista da iteração indefinida, tem cotas:

$$\text{inferior}: \; c_p[b](s^{h(n)}(n))$$

$$\text{superior}: c_p[b](s^{h(n)})(n) + \sum_{i=0}^{h(n)-1} [c_p[b](s^i(n)) + c_p[S](s^i(n))]$$

$$h(n) = \max\{H(d) \in N \mid tam(d) \leq n\}$$

A ordem de complexidade pessimista de enquanto b faça S é dada por

$$c_P\,[\,\underline{\text{enqto}}\text{ b }\underline{\text{faça}}\text{ S }](n) = O(\,c_P\,[\text{ b }](s^{h(n)})(n) + \\ + \sum_{i=0}^{h(n)-1} [c_P\,[\text{ b }](s^i(n)) + c_P\,[\text{ S }](s^i(n))\,]\,).$$

ACP – Iteração Indefinida

Considere a iteração indefinida **enqto b faça S** onde: c_P [S] é um polinômio de grau k; e h é no máximo linear, i. e., $h(n) \leq n$.

a) Se S preserva tamanho, temos $s(n) = n$. Logo, qual é a complexidade do algoritmo ?

a) Se S faz o tamanho decrescer de uma unidade, temos $s(n) = n - 1$. Qual é a complexidade do algoritmo ?

ACP – Iteração Indefinida

Considere a iteração indefinida **enqto b faça S** onde: c_P [S] é um polinômio de grau k; e h é no máximo linear, i. e., h(n) ≤ n.

a) Se S preserva tamanho, temos s(n) = n. Logo, qual é a complexidade do algoritmo ?

$$c_P\,[\,\underline{\text{enqto}}\ b\ \underline{\text{faça}}\ S\,](n) = O\left(\sum_{i=1}^{n} n^k\right) = O(n^{k+1})$$

a) Se S faz o tamanho decrescer de uma unidade, temos s(n) = n - 1. Qual é a complexidade do algoritmo ?

ACP – Iteração Indefinida

Considere a iteração indefinida **enqto b faça S** onde: c_P [S] é um polinômio de grau k; e h é no máximo linear, i. e., $h(n) \leq n$.

a) Se S preserva tamanho, temos $s(n) = n$. Logo, qual é a complexidade do algoritmo ?

$$c_P [\ \underline{\text{enqto}}\ b\ \underline{\text{faça}}\ S\ ](n) = O\left(\sum_{i=1}^{n} n^k\right) = O(n^{k+1})$$

a) Se S faz o tamanho decrescer de uma unidade, temos $s(n) = n - 1$. Qual é a complexidade do algoritmo ?

$$c_P [\ \underline{\text{enqto}}\ b\ \underline{\text{faça}}\ S\ ] = O\left(\sum_{i=0}^{n-1} (n - i)^k\right). = O(n^{k+1})$$

ACP – Exercício

```
Procedimento Loto( Rslt : V, Apst : M )                        {teste da loteria esportiva}
        {Entrada: vetor Rslt : V (resultado de teste), matriz Apst : M (apostador)
     Saída: vetor Ptos : vetor [ 1..n ] de IN (número de pontos feitos por apostador).}
  sorte V := vetor [ 1..13 ] de { 1..3 } ; M := matriz [ 1..n, 0..13 ] de Info ;
1. i ← 1 ;                                              {inicializa número de apostadores}
2. enqto i ≤ n faça                                         {examina apostador: de 2 a 8}
3.    Ptos[ i ] ← 0 ;                         {inicializa número de pontos do apostador i}
4.    para j de 1 até 13 faça                                {examina palpites: de 4 a 6}
5.       se Apst[ i,j ] = Rslt[ j ] então Ptos[ i ] ← Ptos[ i ] + 1     .
6.    fim-para ;                                {4: pontos do apostador i contabilizados}
7.    i ← i + 1 ;                                                        {novo apostador}
8. fim-enqto ;                             {2: contabilizados os pontos de todos apostadores}
9. para i de 1 até n faça                                   {lista ganhadores: de 9 a 11}
10.  se Ptos[ i ] = 13 então escreva( Apst[ i,0 ], "Ganhador, parabéns" )
11. fim-para ;                                                 {9: ganhadores listados}
12. retorne-saída( Ptos ) ;                                             {dá saída Ptos}
13. fim-Procedimento .                          {fim do algoritmo Loto: loteria esportiva}
```

ACP – Exercício

```
1. i ← 1 ;
2. enqto i ≤ n faça
3.    Ptos[ i ] ← 0 ;
4.    para j de 1 até 13 faça
5.       se Apst[ i,j ] = Rslt[ j ] então Ptos[ i ] ← Ptos[ i ] + 1     .
6.    fim-para ;
7.    i ← i + 1 ;
8. fim-enqto ;
9. para i de 1 até n faça
10.   se Ptos[ i ] = 13 então escreva( Apst[ i,0 ], "Ganhador, parabéns" )
11. fim-para ;
12. retorne-saída( Ptos ) ;
13. fim-Procedimento .
```

$$c_p[Loto] = O(c_p[\text{linha } 1] + c_p[\text{linha } 2 \ldots \text{linha } 8] + c_p[\text{linha } 9 \ldots \text{linha } 11])$$

$$c_p[\text{linha } 1] = O(1)$$

$$c_p[\text{linha } 2 \ldots \text{linha } 8] = O(\sum_{i=1}^{n} c_p[\text{linha } 3 \ldots \text{linha } 7])$$

$$c_p[\text{linha } 3 \ldots \text{linha } 7] = O(c_p[\text{linha } 3] + \sum_{j=1}^{13} c_p[\text{linha } 4 \ldots \text{linha } 6] + c_p[\text{linha } 7])$$

$$c_p[\text{linha } 3 \ldots \text{linha } 7] = O(1 + 13.1 + 1) = O(1)$$

$$c_p[\text{linha } 2 \ldots \text{linha } 8] = O(\sum_{i=1}^{n} 1) = O(n)$$

ACP – Exercício

```
1. i ← 1 ;
2. enqto i ≤ n faça
3.    Ptos[ i ] ← 0 ;
4.    para j de 1 até 13 faça
5.       se Apst[ i,j ] = Rslt[ j ] então Ptos[ i ] ← Ptos[ i ] + 1     .
6.    fim-para ;
7.    i ← i + 1 ;
8. fim-enqto ;
9. para i de 1 até n faça
10.   se Ptos[ i ] = 13 então escreva( Apst[ i,0 ], "Ganhador, parabéns" )
11. fim-para ;
12. retorne-saída( Ptos ) ;
13. fim-Procedimento .
```

$$c_p[\text{linha } 9 \ldots \text{linha } 11] = O(\sum_{i=1}^{n} c_p[\text{linha } 10])$$

$$c_p[\text{linha } 10] = O(c_p[PTos[i] = 13] + c_p[escreva...]) = O(1+1) = O(1)$$

$$c_p[\text{linha } 9 \ldots \text{linha } 11] = O(\sum_{i=1}^{n} 1) = O(n)$$

$$c_p[Loto] = O(c_p[\text{linha } 1] + c_p[\text{linha } 2 \ldots \text{linha } 8] + c_p[\text{linha } 9 \ldots \text{linha } 11])$$

$$c_p[Loto] = O(1 + n + n) = O(n)$$

ACP – Exercício

```
Função Max_&_min( Tab : D ) → Par[ Elmord ]        {para máximo e mínimo de tabela}
                                                    {Entrada: tabela Tab : D.
                   Saída: valores Max, min : Elmord (máximo e mínimo da entrada).}
  sorte D := vetor [ p..q ] de Elmord ;
1. Max ← Tab[ p ] ;                                          {candidato a máximo}
2. min ← Tab[ p ] ;                                          {candidato a mínimo}
3. para i de p + 1 até q faça                           {varre tabela: de 3 a 6}
4.    se Tab[ i ] > Max então Max ← Tab[ i ] ;                    {atualiza Max}
5.    se Tab[ i ] < min então min ← Tab[ i ] ;                    {atualiza min}
6. fim-para ;                                             {3: i de p + 1 até q}
7. retorne-saída( Max , min ) ;                           {dá saída Max e min}
8. fim-Função .               {fim do algoritmo Max_&_min: máximo e mínimo de tabela}
```

Considere n=q-p+1

ACP – Exercício

```
1. Max ← Tab[ p ] ;
2. min ← Tab[ p ] ;
3. para i de p + 1 até q faça
4.    se Tab[ i ] > Max então Max ← Tab[ i ] ;
5.    se Tab[ i ] < min então min ← Tab[ i ] ;
6. fim-para ;
7. retorne-saída( Max , min ) ;
8. fim-Função .
```

Considere n=q-p+1

$$c_p[Max - Min] = O(c_p[\text{linha 1}] + c_p[\text{linha 2}] + c_p[\text{linha 3} \ldots \text{linha 6}] + c_p[\text{linha 7}])$$

$$c_p[\text{linha 1}] = c_p[\text{linha 2}] = c_p[\text{linha 7}] = O(1)$$

$$c_p[\text{linha 3} \ldots \text{linha 6}] = O(\sum_{i=p+1}^{q} (c_p[\text{linha 4}] + c_p[\text{linha 5}]))$$

$$c_p[\text{linha 4}] = O(c_p[Tab[i] > Max] + c_p[Max \leftarrow Tab[i]]) = O(1)$$

$$c_p[\text{linha 5}] = O(c_p[Tab[i] < Min] + c_p[Min \leftarrow Tab[i]]) = O(1)$$

$$c_p[\text{linha 3} \ldots \text{linha 6}] = O(\sum_{i=p+1}^{q} (1+1)) = O((n-1).2) = O(n)$$

$$c_p[Max - Min] = O(1+1+1+n) = O(n)$$

ACP – Exercício

```
Procedimento Classif_Max( A : V )          {classifica vetor em ordem ascendente}
                                                          {Entrada: vetor A : V.
          Saída: vetor A : V (o vetor de entrada classificado em ordem ascendente).}
   sorte V := vetor [ 1..n ] de Elmord ;
   var {auxiliar} Temp : Elmord {temporária (para troca)} ;
1. para i de 1 até n -1 faça                                           {de 1 a 10}
2.    para j de 1 até n - i faça                                        {de 2 a 9}
3.       se A[ j ] > A[ j + 1 ]                                  {testa inversão}
4.          então                                     {troca A[ j ] com A[ j + 1 ]}
5.             Temp ← A[ j ] ;
6.             A[ j ] ← A[ j + 1 ] ;
7.             A[ j + 1 ] ← Temp ;
8.       fim-se ;                                      {3: A[ j ] ← A[ j + 1 ]?}
9.    fim-para ;                                            {2: j de 1 até n - i}
10. fim-para ;                                               {1: i  de 1 até n}
11. retorne-saída( A ) ;                                           {dá saída A}
12. fim-Procedimento   .          {fim de Classif_Max: classificação ascendente}
```

ACP – Exercício

```
1. para i de 1 até n -1 faça
2.    para j de 1 até n - i faça
3.       se A[ j ] > A[ j + 1 ]
4.          então
5.             Temp ← A[ j ] ;
6.             A[ j ] ← A[ j + 1 ] ;
7.             A[ j + 1 ] ← Temp ;
8.       fim-se ;
9.    fim-para ;
10. fim-para ;
11. retorne-saída( A ) ;
12. fim-Procedimento   .
```

$$c_p[Classi - Max] = O(c_p[\text{linha } 1 \ldots \text{linha } 10] + c_p[\text{linha } 11])$$

$$c_p[\text{linha } 11] = O(1)$$

$$c_p[\text{linha } 1 \ldots \text{linha } 10] = O(\sum_{i=1}^{n-1} c_p[\text{linha } 2 \ldots \text{linha } 9])$$

$$c_p[\text{linha } 2 \ldots \text{linha } 9] = O(\sum_{j=1}^{n-i} c_p[\text{linha } 3 \ldots \text{linha } 8])$$

$$c_p[\text{linha } 3 \ldots \text{linha } 8] = O(c_p[A[j] > A[j+1]] + c_p[\text{linha } 5 \ldots \text{linha } 7])$$

$$c_p[A[j] > A[j+1]] = O(1)$$

ACP – Exercício

```
1. para i de 1 até n -1 faça
2.    para j de 1 até n - i faça
3.       se A[ j ] > A[ j + 1 ]
4.          então
5.             Temp ← A[ j ] ;
6.             A[ j ] ← A[ j + 1 ] ;
7.             A[ j + 1 ] ← Temp ;
8.       fim-se ;
9.    fim-para ;
10. fim-para ;
11. retorne-saída( A ) ;
12. fim-Procedimento .
```

$$c_p[\text{linha } 5 \ldots \text{linha } 7] = O(1)$$

$$c_p[\text{linha } 3 \ldots \text{linha } 8] = O(1+1) = O(1)$$

$$c_p[\text{linha } 2 \ldots \text{linha } 9] = O(\sum_{j=1}^{n-i} 1) = O(n-i-1+1) = O(n-i)$$

$$c_p[\text{linha } 1 \ldots \text{linha } 10] = O(\sum_{i=1}^{n-1} (n-i)) = O(\frac{(n-1)n}{2}) = O(n^2)$$

$$c_p[Classi - Max] = O(c_p[\text{linha } 1 \ldots \text{linha } 10] + c_p[\text{linha } 11])$$

$$c_p[Classi - Max] = O(n^2+1) = O(n^2)$$

ACP – Inserção Direta

ORDENAÇÃO POR INSERÇÃO DIRETA (INGL. STRAIGHT INSERTION SORT)

Entrada Uma seqüência $a_1, \ldots, a_n$ de números inteiros.

Saída Uma seqüência $a_{\pi(1)}, \ldots, a_{\pi(n)}$ de números inteiros tal que π é uma permutação de $[1, n]$ e para $i < j$ temos $a_{\pi(i)} \leq a_{\pi(j)}$.

```
for i:=2 to n do
  { inv: a_1, ..., a_{i-1} ordenado }
  { coloca item i na posição correta }
  c:=a_i
  j:=i;
  while c < a_{j-1} and j > 1 do
    a_j:=a_{j-1}
    j:=j-1
  end while
  a_j:=c
end for
```

ACP – Inserção Direta

ORDENAÇÃO POR INSERÇÃO DIRETA (INGL. STRAIGHT INSERTION SORT)

Entrada Uma seqüência $a_1, \ldots, a_n$ de números inteiros.

Saída Uma seqüência $a_{\pi(1)}, \ldots, a_{\pi(n)}$ de números inteiros tal que π é uma permutação de $[1, n]$ e para $i < j$ temos $a_{\pi(i)} \leq a_{\pi(j)}$.

```
for i:=2 to n do
  { inv: a_1,...,a_{i-1} ordenado }
  { coloca item i na posição correta }
  c:=a_i
  j:=i;
  while c < a_{j-1} and j > 1 do
    a_j:=a_{j-1}
    j:=j-1
  end while
  a_j:=c
end for
```

$$c_p[SI](n) \leq \sum_{2 \leq i \leq n} \sum_{2 < j \leq i} O(1) = O(n^2)$$