Algoritmos de Procura e de Ordenação III

09/10/2024

Sumário

- Recap
- Bubble Sort – Ordenação por troca sequencial
- Insertion Sort – Ordenação por inserção
- Exercícios / Tarefas 
- Sugestão de leitura

Recapitulação

Selection Sort – Lembram-se do exemplo ?

0	1	2	3	4
7	**2**	**6**	**4**	**3**

7	**2**	**6**	**4**	**3**
3	**2**	**6**	**4**	**7**
3	**2**	**4**	**6**	**7**
3	**2**	**4**	**6**	**7**
2	**3**	**4**	**6**	**7**

5 elementos

4 passos

- Nº de comparações = 4 + 3 + 2 + 1
- Nº de trocas = 1 + 1 + 0 + 1

Selection Sort – Ordem de Complexidade

- Nº fixo de Comparações entre elementos do array :

$$C(n) \approx \frac{n^2}{2} \Rightarrow \boldsymbol{O(n^2)}$$

- Nº de Trocas entre elementos do array:

$$W_T(n) = n - 1 \Rightarrow \boldsymbol{O(n)}$$

$$A_T(n) \approx n - \ln n \Rightarrow \boldsymbol{O(n)}$$

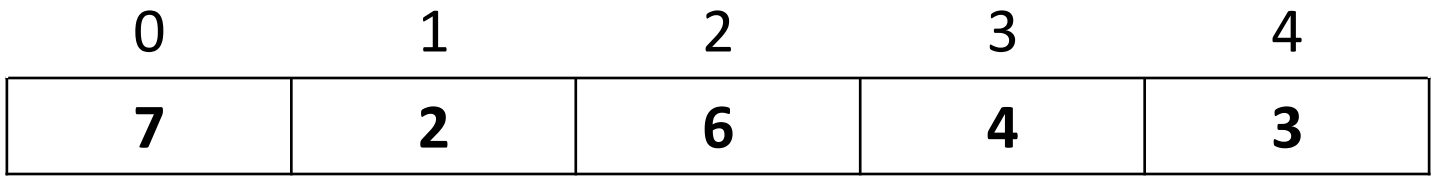
$$B_T(n) = 0$$

Bubble Sort
– Ordenação por Troca Sequencial

Bubble Sort – Estratégia

- Percorrer o array da esquerda para a direita
- Trocar elementos adjacentes, se estiverem fora de ordem
  - Quantas comparações ?
- A última ocorrência do maior elemento fica na sua posição final
- Repetir o processo para os restantes elementos
  - Parar logo que possível !
- Algoritmo in-place
- Shaker Sort : alternar o sentido : esquerda-direita / direita-esquerda

Exemplo



Exemplo – 1ª Iteração

0	1	2	3	4
7	**2**	**6**	**4**	**3**

7	**2**	**6**	**4**	**3**

1ª iteração

Exemplo – Avançar e trocar, se necessário

0	1	2	3	4
7	**2**	**6**	**4**	**3**

1ª iteração

7	**2**	**6**	**4**	**3**
2	**7**	**6**	**4**	**3**

Exemplo – Avançar e trocar, se necessário

0	1	2	3	4
7	**2**	**6**	**4**	**3**

1ª iteração

7	**2**	**6**	**4**	**3**
2	**7**	**6**	**4**	**3**
2	**6**	**7**	**4**	**3**

Exemplo – Avançar e trocar, se necessário

0	1	2	3	4
7	**2**	**6**	**4**	**3**

1ª iteração

7	**2**	**6**	**4**	**3**
2	**7**	**6**	**4**	**3**
2	**6**	**7**	**4**	**3**
2	**6**	**4**	**7**	**3**

Exemplo – Posição final

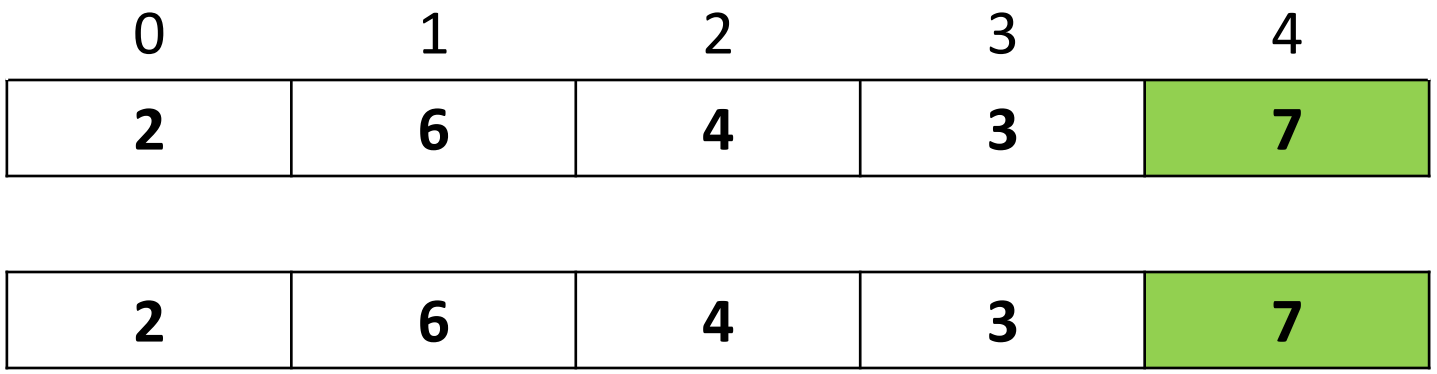
0	1	2	3	4
7	**2**	**6**	**4**	**3**

1ª iteração

7	**2**	**6**	**4**	**3**
2	**7**	**6**	**4**	**3**
2	**6**	**7**	**4**	**3**
2	**6**	**4**	**7**	**3**
2	**6**	**4**	**3**	**7**

- 4 comparações + 4 trocas

Exemplo – 2ª Iteração



2ª iteração

Exemplo – Avançar e trocar, se necessário

0	1	2	3	4
2	**6**	**4**	**3**	**7**

2ª iteração

2	**6**	**4**	**3**	**7**
2	**6**	**4**	**3**	**7**

Exemplo – Avançar e trocar, se necessário

0	1	2	3	4
2	**6**	**4**	**3**	**7**

2ª iteração

2	**6**	**4**	**3**	**7**
2	**6**	**4**	**3**	**7**
2	**4**	**6**	**3**	**7**

Exemplo – Posição final

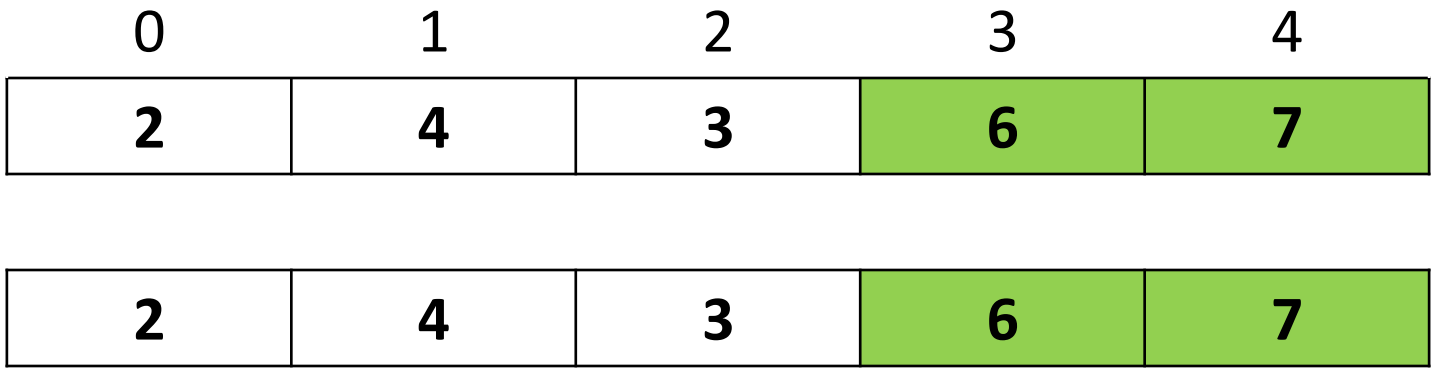
0	1	2	3	4
2	**6**	**4**	**3**	**7**

2ª iteração

2	**6**	**4**	**3**	**7**
2	**6**	**4**	**3**	**7**
2	**4**	**6**	**3**	**7**
2	**4**	**3**	**6**	**7**

- 3 comparações + 2 trocas

Exemplo – 3ª Iteração



3ª iteração

Exemplo – Avançar e trocar, se necessário

0	1	2	3	4
2	**4**	**3**	**6**	**7**

3ª iteração

2	**4**	**3**	**6**	**7**
2	**4**	**3**	**6**	**7**

Exemplo – Posição final

0	1	2	3	4
2	**4**	**3**	**6**	**7**

3ª iteração

2	**4**	**3**	**6**	**7**
2	**4**	**3**	**6**	**7**
2	**3**	**4**	**6**	**7**

- 2 comparações + 1 troca

Exemplo – Posição final – Array ordenado

0	1	2	3	4
2	**3**	**4**	**6**	**7**

4ª iteração

2	**3**	**4**	**6**	**7**
2	**3**	**4**	**6**	**7**

- 1 comparação + 0 trocas

- TOTAL de comparações = 4 + 3 + 2 + 1
- TOTAL de trocas = 4 + 2 + 1 + 0

Bubble Sort

```
void bubbleSort( int a[], int n ) {
        int k = n; int stop = 0;
        while( stop == 0 ) {
            stop = 1; k--;
            for( int i = 0; i < k; i++ )
                if( a[i] > a[i + 1] )  {
                    swap( &a[i], &a[i + 1] );
                    stop = 0;
                }
        }
}
```

Nº de Comparações – Melhor Caso e Pior Caso

- Melhor Caso ? - Array ordenado
- $B_c(n) = n - 1$ **O(n)**
- Pior Caso ? - Array pela ordem inversa, sem elementos repetidos
- $W_c(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$ **O(n²)**

Nº de Trocas – Melhor Caso e Pior Caso

- Melhor Caso ? - Array ordenado
- $B_t(n) = 0$ **O(1)**
- Pior Caso ? - Array pela ordem inversa, sem elementos repetidos
  - 1 troca para cada comparação
- $W_{t(n)} = W_c(n) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$ **O(n²)**

Nº de Comparações – Caso Médio

- Casos possíveis ?

Nº de iterações do ciclo while	Nº de comparações realizadas : **C(j)**	Probabilidade
1	(n – 1)	
2	(n – 1) + (n – 2)	
…	…	
j	(n – 1) + (n – 2) + … + (n – j)	
…	…	
n – 2	(n – 1) + (n – 2) + … + 2	
n – 1	(n – 1) + (n – 2) + … + 2 + 1	

- $C(j) = \sum_{i=1}^{j}(n-i) = \frac{j}{2}[(n-1)+(n-j)] = \frac{j}{2}[(2n-1)-j]$

Nº de Comparações – Caso Médio

- Probabilidade ? Simplificação…

Nº de iterações do ciclo while	Nº de comparações realizadas : C(j)	Probabilidade
1	(n – 1)	1 / (n – 1)
2	(n – 1) + (n – 2)	1 / (n – 1)
…	…	…
j	(n – 1) + (n – 2) + … + (n – j)	1 / (n – 1)
…	…	…
n – 2	(n – 1) + (n – 2) + … + 2	1 / (n – 1)
n – 1	(n – 1) + (n – 2) + … + 2 + 1	1 / (n – 1)

- Habitualmente, para arrays aleatórios, o número de iterações do ciclo while é próximo do seu número máximo

Nº de Comparações – Caso Médio

$$A_c(n) = \sum_{k=1}^{n-1} \frac{1}{n-1} C(k) = \cdots = \frac{1}{2(n-1)} \sum_{k=1}^{n-1} [(2n-1)k - k^2]$$

Expressão auxiliar: $\sum_{k=1}^{n} k^2 = \frac{1}{6} n(n+1)(2n+1)$

$$\boldsymbol{A_c(n) = \frac{1}{3} n^2 - \frac{1}{6} n}$$

O(n²)

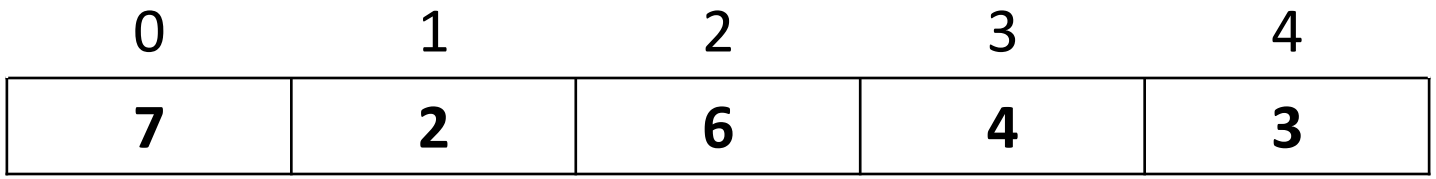
- Façam o desenvolvimento e confirmem o resultado

Insertion Sort
– Ordenação por Inserção

Insertion Sort – Estratégia

- O elemento a[0] constitui um subconjunto de um só elemento
- Inserir ordenadamente o elemento a[1] nesse subconjunto
  - 1 comparação + 0 ou 1 troca de posição
- Temos agora um subconjunto ordenado com dois elementos
- Repetir o processo, um a um, para os restantes elementos do array
  - Casos possíveis ? Quantas comparações ? Quantos deslocamentos ?
- Algoritmo in-place
- Variante: começar na outra extremidade do array

Exemplo



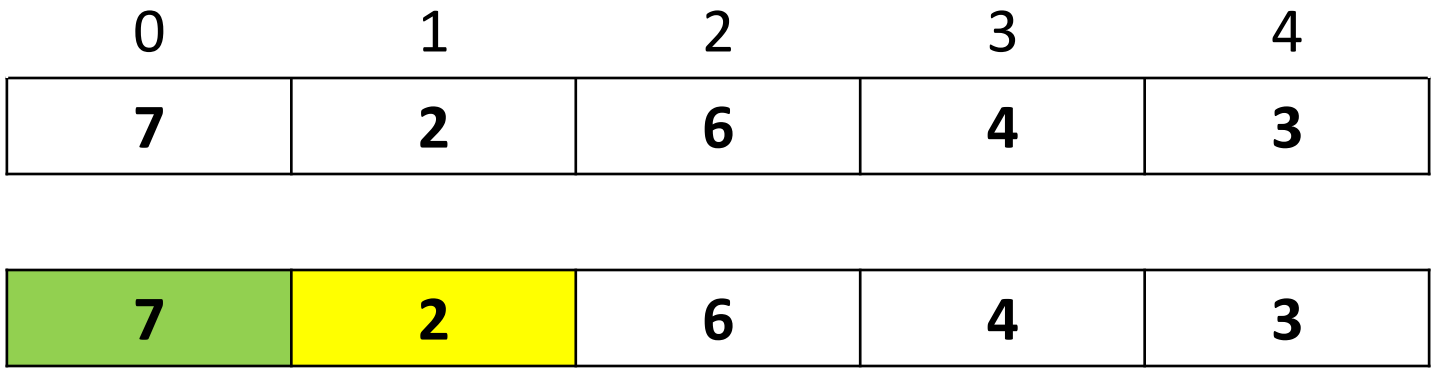
Exemplo – 1ª Iteração

0	1	2	3	4
7	**2**	**6**	**4**	**3**

7	**2**	**6**	**4**	**3**

1ª iteração

Exemplo – Inserir ordenadamente



1ª iteração

Exemplo – Posição final

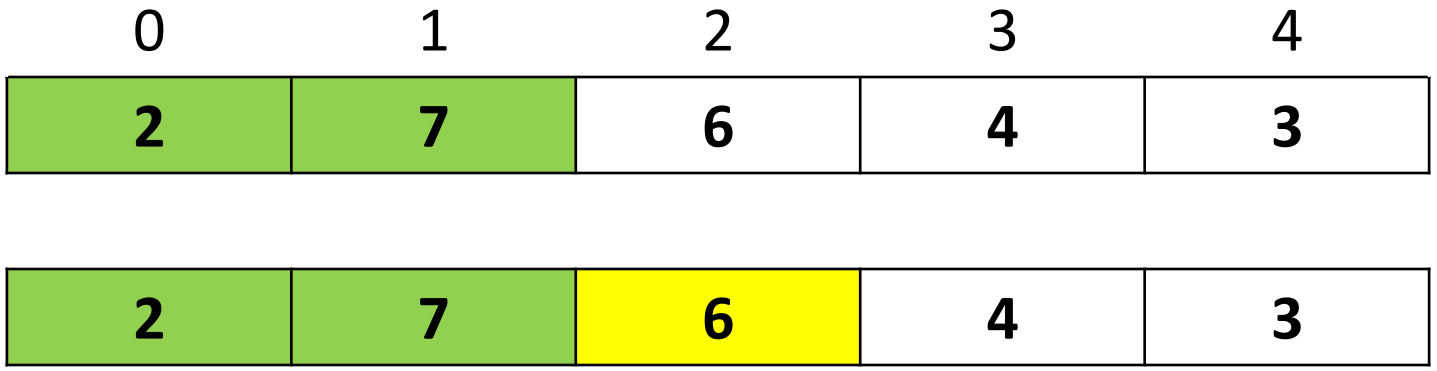
0	1	2	3	4
7	**2**	**6**	**4**	**3**

1ª iteração

7	**2**	**6**	**4**	**3**
2	**7**	**6**	**4**	**3**

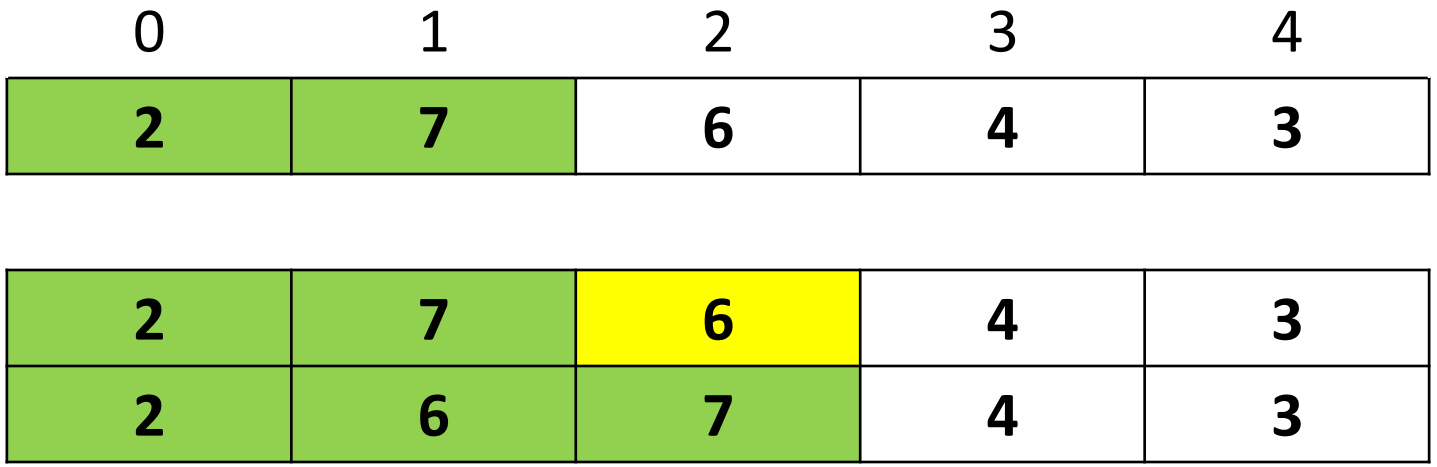
- 1 comparação + 1 deslocamento

Exemplo – Inserir ordenadamente



2ª iteração

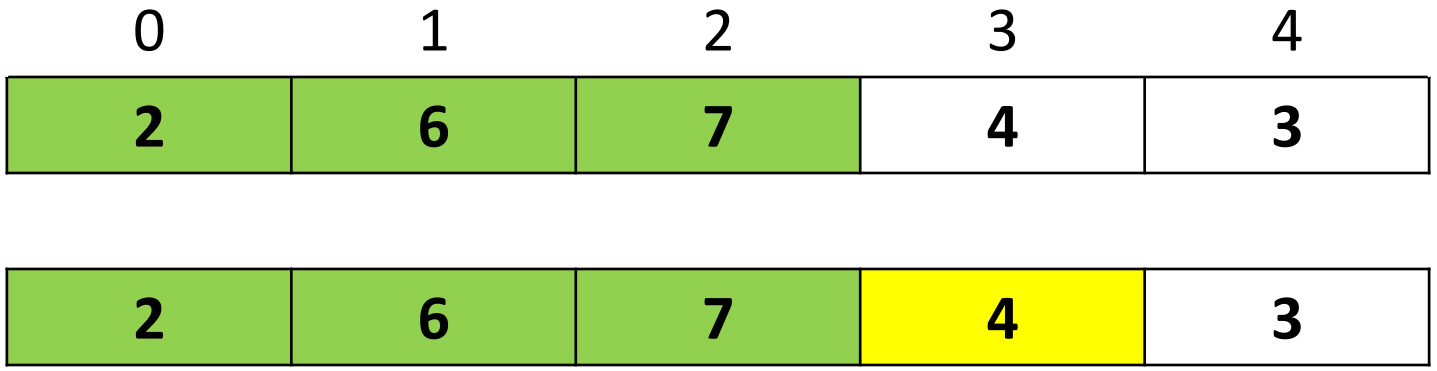
Exemplo – Posição final



2ª iteração

- 2 comparações + 1 deslocamento

Exemplo – Inserir ordenadamente



3ª iteração

Exemplo – Continuar a deslocar

0	1	2	3	4
2	**6**	**7**	**4**	**3**

3ª iteração

2	**6**	**7**	**4**	**3**
2	**6**	**4**	**7**	**3**

Exemplo – Posição final

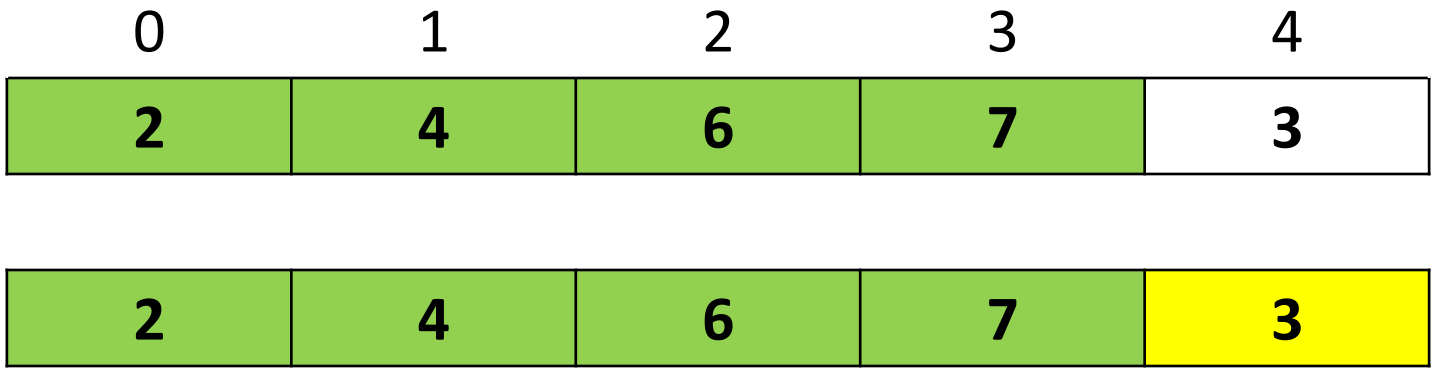
0	1	2	3	4
2	**6**	**7**	**4**	**3**

3ª iteração

2	**6**	**7**	**4**	**3**
2	**6**	**4**	**7**	**3**
2	**4**	**6**	**7**	**3**

- 3 comparações + 2 deslocamentos

Exemplo – Inserir ordenadamente



4ª iteração

Exemplo – Continuar a deslocar

0	1	2	3	4
2	**4**	**6**	**7**	**3**

4ª iteração

2	**4**	**6**	**7**	**3**
2	**4**	**6**	**3**	**7**

Exemplo – Continuar a deslocar

0	1	2	3	4
2	**4**	**6**	**7**	**3**

4ª iteração

2	**4**	**6**	**7**	**3**
2	**4**	**6**	**3**	**7**
2	**4**	**3**	**6**	**7**

Exemplo – Posição final

0	1	2	3	4
2	**4**	**6**	**7**	**3**

4ª iteração

2	**4**	**6**	**7**	**3**
2	**4**	**6**	**3**	**7**
2	**4**	**3**	**6**	**7**
2	**3**	**4**	**6**	**7**

4 comparações + 3 deslocamentos

Função Auxiliar – Inserção Ordenada

```
void insertElement( int sorted[], int n, int elem ) {
        // Array sorted está ordenado
        // Há espaço para acrescentar mais um elemento
        int i;
        for( i = n – 1; (i >= 0) && (elem < sorted[i]); i-- )
                sorted[i  + 1] = sorted[i]
        sorted[i + 1] = elem;
}
```

- Deslocamentos para a direita, para abrir espaço e inserir

Comparações – Melhor Caso e Pior Caso

- Melhor Caso
- B(n) = 1
- elem >= sorted[n – 1]

- Pior Caso
- W(n) = n
- elem < sorted[0] OU sorted[0] <= elem < sorted[1]

Comparações – Caso Médio

Casos possíveis	Nº de comparações	Probabilidade
elem < sorted[0]	n	1 / (n + 1)
sorted[0] <= elem < sorted[1]	n	1 / (n + 1)
sorted[1] <= elem < sorted[2]	n – 1	1 / (n + 1)
…	…	…
sorted[i] <= elem < sorted[i+1]	n – i	1 / (n + 1)
…	…	…
sorted[n-2] <= elem < sorted[n-1]	2	1 / (n + 1)
sorted[n-1] <= elem	1	1 / (n + 1)

$$A_c(n) = \frac{1}{n+1}[1 + 2 + \cdots + n + n] = \frac{n}{2} + \frac{n}{n+1} \approx \frac{n}{2} + 1$$

Insertion Sort

```
void insertionSort( int a[], int n ) {
        for( int  i = 1; i < n; i++ )
                if( a[i] < a[i – 1] )
                        insertElement( a, i, a[i] );
}
```

- Deslocamentos (i.e., atribuições) são efetuados pela função auxiliar
- Contar também as comparações feitas pela função auxiliar !!

Comparações – Melhor Caso e Pior Caso

- Melhor Caso
- $B_c(n) = n - 1$ **O(n)**
- Array ordenado : A função auxiliar nunca é chamada

- Pior Caso
- $W_c(n) = \sum_{i=1}^{n-1}(1 + i) = \frac{n-1}{2} \times (n + 2)$ **O(n²)**
- A função auxiliar é sempre chamada !!
- E tem sempre o comportamento de pior caso !!

Comparações – Caso Médio

- Análise simplificada !
- Considera-se que em cada iteração a função auxiliar tem sempre o comportamento do caso médio (i/2 + 1)

$$A_c(n) \approx \sum_{i=1}^{n-1} \left[1 + \left(\frac{i}{2} + 1\right)\right] = \left[2(n-1) + \frac{n(n-1)}{4}\right]$$

$$A_c(n) \approx \frac{n^2}{4} + \frac{7n}{4}$$

- Comparar com o pior caso !

Exercícios / Tarefas

Tarefa 1 – Bubble Sort

- Encontrar configurações de um array com n elementos que, para o algoritmo **Bubble Sort**, correspondam:
- Ao **melhor caso** para as **comparações**
- Ao **pior caso** para as **comparações**
- Ao **melhor caso** para as **trocas**
- Ao **pior caso** para as **trocas**

- **Alguns dos casos anteriores ocorrem em simultâneo ?**

Exercício 1 – Escolha múltipla

Considere o seguinte *array* de 6 elementos, que se pretende ordenar usando o algoritmo ***Bubble Sort***.

0	1	2	3	4	5
6	5	4	3	2	1

a) São efetuadas **15 comparações** entre elementos do *array*, para que seja ordenado por **ordem crescente**.

b) São efetuadas **15 trocas** entre elementos do *array*, para que seja ordenado por **ordem crescente**.

c) Ambas estão corretas.

d) Nenhuma está correta.

Exercício 2 – Escolha múltipla

Considere o seguinte *array* de 6 elementos, que se pretende ordenar usando o algoritmo ***Bubble Sort***.

0	1	2	3	4	5
6	5	2	3	4	1

a) São efetuadas **12 trocas** entre elementos do *array*, se este for ordenado por **ordem crescente**.

b) São efetuadas **3 trocas** entre elementos do *array*, se este for ordenado por **ordem decrescente**.

c) Ambas estão corretas.

d) Nenhuma está correta.

Tarefa 2 – Bubble Sort – Trocas – Caso Médio

- **Bubble Sort :** Efetuar a análise do caso médio para o número de trocas efetuadas

- **Possível cenário :**
- Igualmente provável terminar após qualquer uma das iterações do ciclo externo (while)
- Em cada iteração do ciclo externo (while) fazem-se, em média, 50% do nº de trocas possíveis

Tarefa 3 – Insertion Sort

- Encontrar configurações de um array com n elementos que, para o algoritmo **Insertion Sort**, correspondam:
- Ao **melhor caso** para as **comparações**
- Ao **pior caso** para as **comparações**
- Ao **melhor caso** para os **deslocamentos**
- Ao **pior caso** para os **deslocamentos**

- **Alguns dos casos anteriores ocorrem em simultâneo ?**

Exercício 3 – Escolha múltipla

Considere o seguinte *array* de 6 elementos, que se pretende ordenar usando o algoritmo ***Insertion Sort***.

0	1	2	3	4	5
6	5	4	4	2	1

a) São efetuadas **15 comparações** entre elementos do *array*, para que seja ordenado por **ordem crescente**.

b) São efetuadas **25 atribuições** entre elementos do *array*, para que seja ordenado por **ordem decrescente**.

c) Ambas estão corretas.

d) Nenhuma está correta.

Tarefa 4 – Insertion Sort – Caso Médio

- **Função auxiliar de inserção ordenada :** Efetuar a análise do caso médio para o número de deslocamentos efetuados, i.e., das atribuições efetuadas envolvendo elementos do array

- **Possível cenário :**
- Igualmente provável terminar após qualquer uma das iterações

Tarefa 5 – Insertion Sort – Caso Médio

- **Insertion Sort :** Efetuar a análise para o caso médio dos deslocamentos, i.e., das atribuições efetuadas envolvendo elementos do array

- **Possível cenário simplificado :**
- Em cada iteração fazem-se, em média, 50% do nº de deslocamentos possíveis

Tarefa – Overview – Algoritmos de Ordenação

- Construir uma **tabela** que agrupe as características dos algoritmos analisados, relativamente à sua ordem de complexidade

- E uma outra **tabela** que mostre como evolui o número de comparações efetuadas, para sucessivos valores de n
- n = 100, 1000, 10000, 100000, 1000000, …

- Qual é o maior array que consegue processar, em tempo útil, no seu computador ?

Sugestão de leitura

Sugestão de leitura

- J. J. McConnell, Analysis of Algorithms, 1st Edition, 2001
  - Capítulo 3: secções 3.1 e 3.2