

Algoritmos de Procura e de Ordenação III

04/10/2023

Sumário

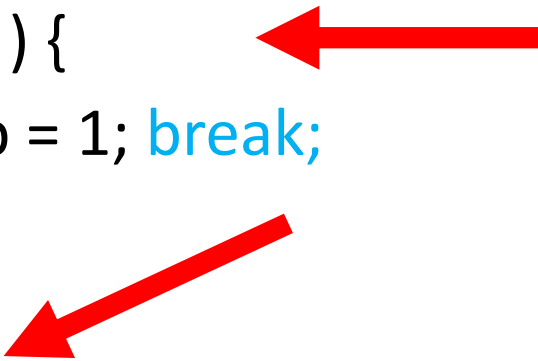
- Recap
- Selection Sort – Ordenação por seleção – **Conclusão**
- Bubble Sort – Ordenação por troca sequencial
- Insertion Sort – Ordenação por inserção
- Sugestão de leitura

Let's
RECAP

Recapitulação

Procura sequencial – Array ordenado

```
int search( int a[], int n, int x ) {  
    int stop = 0; int i;  
    for( i=0; i<n; i++ ) {  
        if( x <= a[i] ) {  
            stop = 1; break;  
        }  
    }  
    if( stop && x == a[i] ) return i;  
    return -1;  
}
```



Comparações:

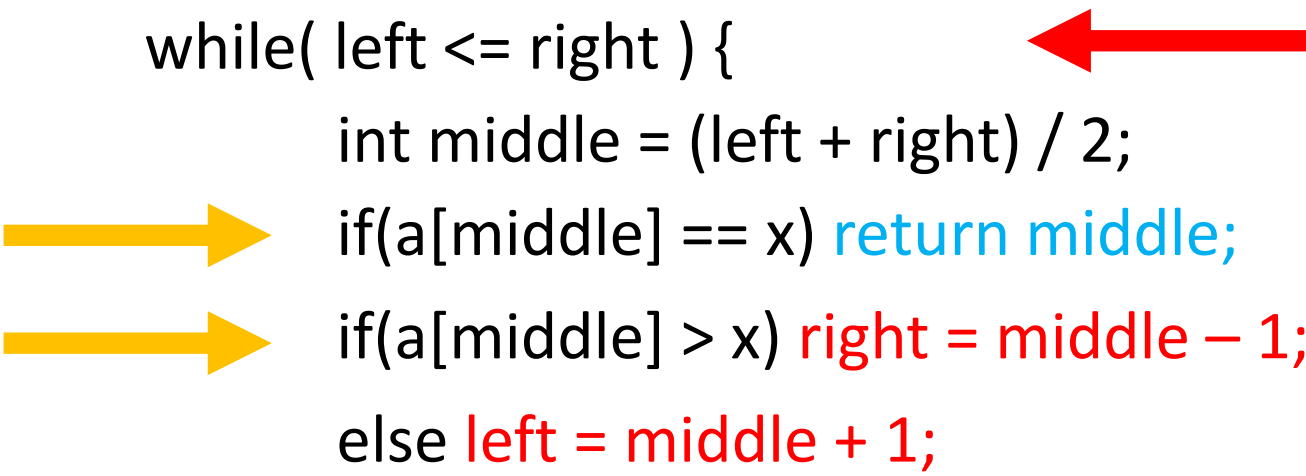
$$B(n) = 2$$

$$W(n) = n + 1$$

$$A(n) \approx n / 2$$

Procura binária – Array ordenado

```
int binSearch( int a[], int n, int x ) {  
    int left = 0; int right = n - 1;  
    while( left <= right ) {  
        int middle = (left + right) / 2;  
        if(a[middle] == x) return middle;  
        if(a[middle] > x) right = middle - 1;  
        else left = middle + 1;  
    }  
    return -1;  
}
```



A diagram illustrating the binary search algorithm's loop. A red arrow points from the right towards the 'while' loop condition. Two yellow arrows point from the left towards the 'if(a[middle] > x)' and 'else' branches, highlighting the updates to the 'right' and 'left' pointers respectively.

Iterações:

$$B(n) = 1$$

$$W(n) = 1 + \lfloor \log_2 n \rfloor$$

$$A(n) \approx W(n) - \frac{1}{2}$$

Desempenho – Nº de comparações

Nº de elementos	Procura Sequencial		Procura Binária	
	A(n)	W(n)	A(n)	W(n)
$2^9 - 1 = 511$	256	512	17	18
$2^{10} - 1 = 1023$	512	1024	19	20
$2^{14} - 1 = 16383$	8192	16384	27	28
$2^{17} - 1 = 131071$	65536	131072	33	34
$2^{20} - 1 = 1048575$	524288	1048576	39	40
	$O(n)$	$O(n)$	$O(\log n)$	$O(\log n)$

- **P. Binária** : nº de comparações $\approx 2 \times$ nº de iterações
- Caso médio : valores aproximados
- Valores procurados podem não estar no array (**Cenário 2**)

Selection Sort

– Ordenação por Seleção

Ideia

- Procurar a **última ocorrência do maior** elemento
 - Quantas **comparações** ?
- Colocá-lo na última posição, se necessário, efetuando uma **troca**
- **Repetir** o processo para os restantes elementos
 - Quantas **comparações** ?
- Algoritmo **in-place**
- **Variante:** procurar a primeira ocorrência do menor elemento

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

- N^o de **comparações** = $4 + 3 + 2 + 1$
- N^o de **trocas** = $1 + 1 + 0 + 1$

Selection Sort

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

Nº de Comparações

- Número **fixo** de comparações ! --- Algoritmo “pouco inteligente”
- Mesmo que o array já esteja ordenado, continuamos a comparar !!

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

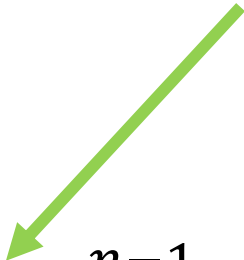
$$\mathbf{O(n^2)}$$

Nº de Trocas – Melhor Caso e Pior Caso

- Melhor Caso ?
- $Bt(n) = 0$ - Quando ?
- Pior Caso ?
- $Wt(n) = n - 1$ - Quando ? $O(n)$
- Um array pela ordem inversa é uma configuração de pior caso ?

Nº de Trocas – Caso Médio

- $p(I_j)$ é a probabilidade de o elemento $a[j]$ estar na **posição correta**
- Simplificação : **Equiprobabilidade** : $p(I_j) = 1 / (j + 1)$
- $(1 - p(I_j))$ é a probabilidade de ser necessária **uma troca** para o elemento $a[j]$ ficar na posição correta
- $(n - 1)$ **passos**


$$A_t(n) = \sum_{j=1}^{n-1} (1 - p(I_j)) \times \mathbf{1} = \sum_{j=1}^{n-1} 1 - \sum_{j=1}^{n-1} p(I_j)$$

Nº de Trocas – Caso Médio

$$A_t(n) = \sum_{j=1}^{n-1} (1 - p(I_j)) \times \mathbf{1} = \sum_{j=1}^{n-1} 1 - \sum_{j=1}^{n-1} p(I_j)$$

$$A_t(n) = n - 1 - \sum_{j=1}^{n-1} \frac{1}{j+1} = n - \left\{ 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} \right\} = n - H_n$$

$$\mathbf{A_t(n) = n - H_n \approx n - \ln n} \quad \mathbf{O(n)}$$

Bubble Sort

– Ordenação por Troca Sequencial

Ideia

- 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

0	1	2	3	4
7	2	6	4	3

Exemplo

0	1	2	3	4
7	2	6	4	3

7	2	6	4	3
----------	----------	----------	----------	----------

1ª iteração

Exemplo

0	1	2	3	4
7	2	6	4	3

7	2	6	4	3
2	7	6	4	3

1ª iteração

Exemplo

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

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

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

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

2ª iteração

Exemplo

0	1	2	3	4
2	6	4	3	7

2	6	4	3	7
2	6	4	3	7

2ª iteração

Exemplo

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

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

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

3ª iteração

Exemplo

0	1	2	3	4
2	4	3	6	7

2	4	3	6	7
2	4	3	6	7

3ª iteração

Exemplo

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

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

4ª iteração

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

Tarefa 1

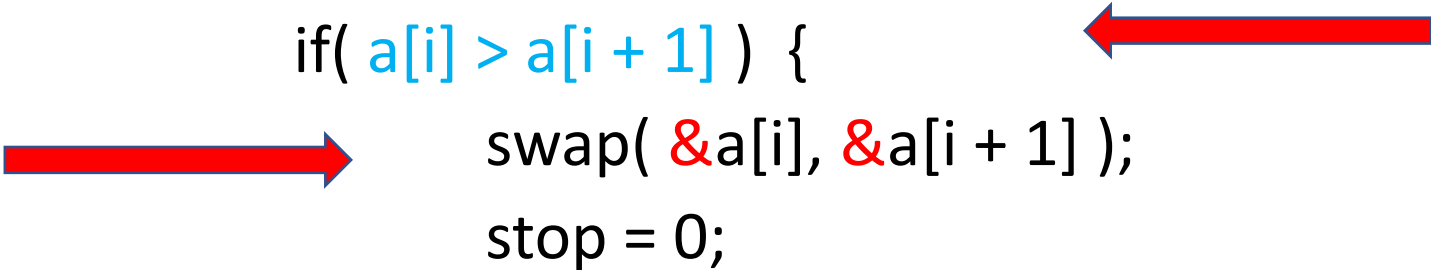
- Organizar **configurações do array** que correspondam:
- Ao **melhor caso** para as **comparações**
- Ao **pior caso** para as **comparações**
- Ao **melhor caso** para as **trocas**
- Ao **pior caso** para as **trocas**
- Alguns dos casos anteriores ocorrem em **simultâneo** ?

Nº de operações ?

- Nº de comparações = ?
- Melhor caso : $Bc(n) = n - 1$ - Quando ? $O(n)$
- Pior Caso : $Wc(n) = (n - 1) + \dots + 1 = n \times (n - 1) / 2$ $O(n^2)$
- Nº de trocas = ?
- Melhor caso : $Bt(n) = 0$ - Quando ? $O(1)$
- Pior caso : $Wt(n) = Wc(n)$ - Quando ? $O(n^2)$

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

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

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

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

Nº de Comparações – Caso Médio

- Casos possíveis ?

Nº de iterações do ciclo while	Nº de comparações realizadas	Probabilidade
1	$(n - 1)$	
2	$(n - 1) + (n - 2)$	
...	...	
j	$(n - 1) + (n - 2) + \dots + (n - j)$	
...	...	
$n - 2$	$(n - 1) + (n - 2) + \dots + 2$	
$n - 1$	$(n - 1) + (n - 2) + \dots + 2 + 1$	

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

Nº de Comparações – Caso Médio

- Probabilidade ? Simplificação...

Nº de iterações do ciclo while	Nº de comparações realizadas	Probabilidade
1	$(n - 1)$	$1 / (n - 1)$
2	$(n - 1) + (n - 2)$	$1 / (n - 1)$
...
j	$(n - 1) + (n - 2) + \dots + (n - j)$	$1 / (n - 1)$
...
$n - 2$	$(n - 1) + (n - 2) + \dots + 2$	$1 / (n - 1)$
$n - 1$	$(n - 1) + (n - 2) + \dots + 2 + 1$	$1 / (n - 1)$

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

Nº de Comparações – Caso Médio

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

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

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

$O(n^2)$

- Façam o desenvolvimento e confirmem o resultado

Tarefa 2

- Efetuar a análise para o **caso médio das trocas**
- 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

Insertion Sort

– Ordenação por Inserção

Ideia

- 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

0	1	2	3	4
7	2	6	4	3

Exemplo

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

1ª iteração

Exemplo

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

1ª iteração

Exemplo

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

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

2ª iteração

Exemplo

0	1	2	3	4
2	7	6	4	3

2ª iteração

2	7	6	4	3
2	6	7	4	3

- 2 comparações + 1 deslocamento

Exemplo

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

3ª iteração

Exemplo

0	1	2	3	4
2	6	7	4	3

3ª iteração

2	6	7	4	3
2	6	4	7	3

Exemplo

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

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

4ª iteração

Exemplo

0	1	2	3	4
2	4	6	7	3

4ª iteração

2	4	6	7	3
2	4	6	3	7

Exemplo

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

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

Tarefa 3

- Organizar **configurações do array** que 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** ?

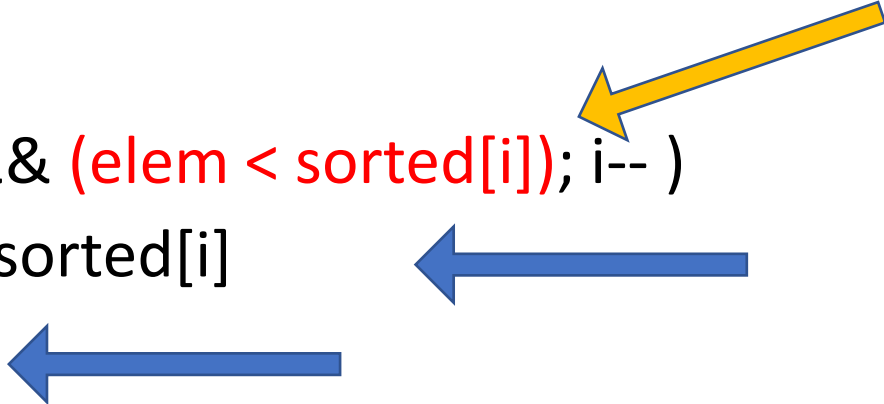
Nº de operações ?

- Nº de comparações = ?
- Melhor caso : $Bc(n) = n - 1$ - Quando ? $O(n)$
- Pior Caso : $Wc(n) = 1 + \dots + (n - 1) = n \times (n - 1) / 2$ $O(n^2)$
- Nº de deslocamentos = ?
- Melhor caso : $Bt(n) = 0$ - Quando ?
- Pior caso : $Wt(n) = Wc(n)$ - Quando ? $O(n^2)$

Como efetuar os deslocamentos de modo eficiente ?

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
- $B(n) = 1$
- $\text{elem} \geq \text{sorted}[n - 1]$

- Pior Caso
- $W(n) = n$
- $\text{elem} < \text{sorted}[0]$ OU $\text{sorted}[0] \leq \text{elem} < \text{sorted}[1]$

Comparações – Caso Médio

Casos possíveis	Nº de comparações	Probabilidade
$\text{elem} < \text{sorted}[0]$	n	$1 / (n + 1)$
$\text{sorted}[0] \leq \text{elem} < \text{sorted}[1]$	n	$1 / (n + 1)$
$\text{sorted}[1] \leq \text{elem} < \text{sorted}[2]$	$n - 1$	$1 / (n + 1)$
...
$\text{sorted}[i] \leq \text{elem} < \text{sorted}[i+1]$	$n - i$	$1 / (n + 1)$
...
$\text{sorted}[n-2] \leq \text{elem} < \text{sorted}[n-1]$	2	$1 / (n + 1)$
$\text{sorted}[n-1] \leq \text{elem}$	1	$1 / (n + 1)$


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

Tarefa 4

- Efetuar a análise para o **caso médio dos deslocamentos**, i.e., das **atribuições** efetuadas envolvendo **elementos do array**
- Possível **cenário** :
- Igualmente provável **terminar após qualquer uma das iterações**

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 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

- $W_c(n) = \sum_{i=1}^{n-1} (1 + i) = \frac{n-1}{2} \times (n + 2)$

$O(n^2)$

- 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**

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

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

- Comparar com o pior caso !

Tarefa 5

- Efetuar a análise para o **caso médio dos deslocamentos**, i.e., das **atribuições** efetuadas envolvendo **elementos do array**
- Possível **cenário** :
- Em cada **iteração** fazem-se, em média, **50% do nº de deslocamentos possíveis**

Tarefa 6

- Construir uma **tabela** que agrupe as **características dos algoritmos** anteriores
- E **outra tabela** que mostre como evolui o **número de comparações efetuadas**, para sucessivos valores de n
- $n = 100, 1000, 10000, 100000, 1000000, \dots$

Sugestão de leitura

Sugestão de leitura

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