Algoritmos de Procura e de Ordenação II

02/10/2023

Sumário

- Recap
- Binary Search – Procura binária – Conclusão
- Algoritmos de Ordenação
- Selection Sort – Ordenação por seleção
- Sugestão de leitura

Recapitulação

Procura sequencial – Array não ordenado

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

Comparações:

$B(n) = 1$

$W(n) = n$

$A(n) = ?$

Caso Médio – $p = Prob(x \in a[0..(n-1)])$

$$A(n,p) = \frac{p \times (n+1)}{2} + (1-p) \times n$$

- Se p = 1, então A(n) = (n + 1) / 2 ≈ n / 2
- Se p = 50%, então A(n) = (n + 1) / 4 + n / 2 ≈ 3 x n / 4
- Se p = 25%, então A(n) = (n + 1) / 8 + 3 x n / 4 ≈ 7 x n / 8

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:

$B(n) = 2$

$W(n) = n + 1$

$A(n) = ?$

\# Ordem da conjunção !!

Caso Médio – Equiprobabilidade

Casos possíveis		Nº de comparações	Probabilidade
Sucesso 0	É o 1º elemento	2	1 / (2 n +1)
Sucesso 1	É o 2º elemento	3	1 / (2 n +1)
…	…	…	…
Sucesso (n – 1)	É o último elemento	n + 1	1 / (2 n +1)
Insucesso 0	Menor do que o 1º	2	1 / (2 n +1)
Insucesso 1	Entre o 1º e o 2º	3	1 / (2 n +1)
…	…	…	…
Insucesso (n – 1)	Entre o penúltimo e o último	n + 1	1 / (2 n +1)
Insucesso n	Maior do que o último	n	1 / (2 n +1)

- $A(n) \approx \frac{n}{2}$ --- Melhor do que caso idêntico com array não ordenado !!

Binary Search – Array ordenado

Procura binária – Nº de iterações do ciclo ?

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

Melhor Caso

- O 1º elemento consultado é o valor procurado !!

- 1 comparação
- 1 iteração do ciclo while

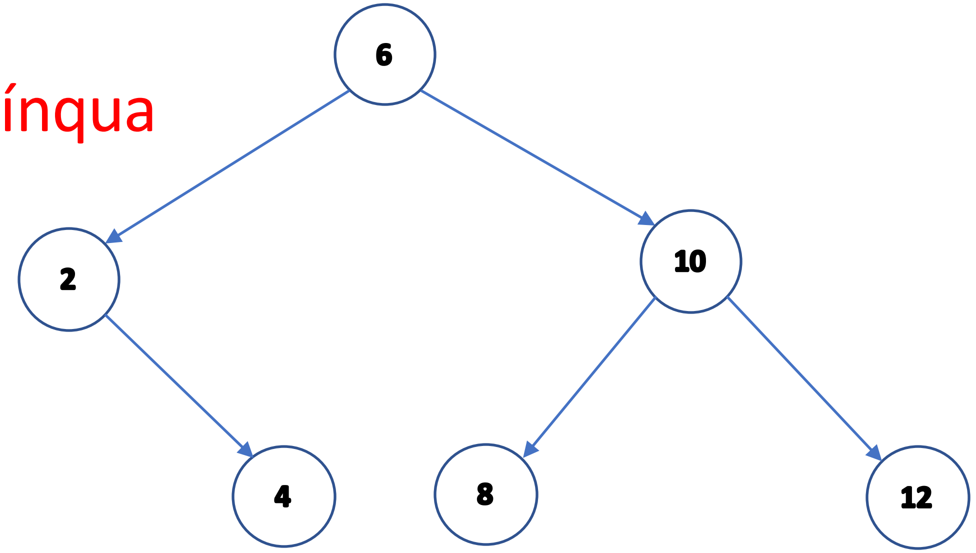
- $B(n) = 1$

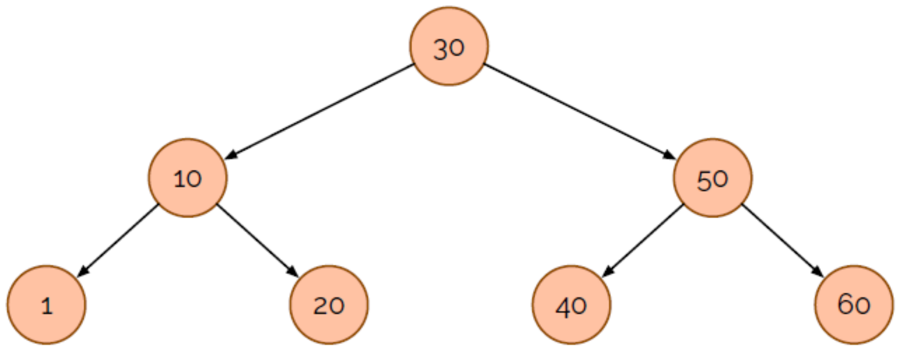
- Muito pouco habitual…

Pior Caso – Valor procurado pertence ao array

- Percorrer a árvore até atingir a folha mais longínqua
- Em geral, qual é a forma da árvore ?
- E qual é a sua altura ?

- Array com 1 elemento : ?
- Array com 2 elementos : ?
- Array com 3 elementos : ?
- Array com 4 elementos : ?
- …
- Array com n elementos : ?

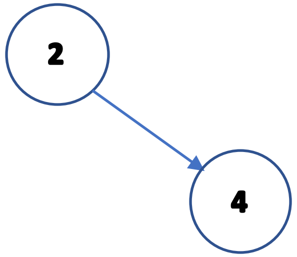




Pior Caso – Situações mais simples

- n = 1 1 iteração
- n = 2

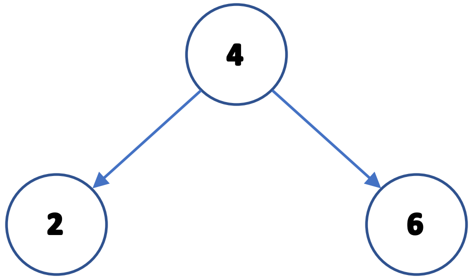
0	1
2	**4**



2 iterações

- n = 3

0	1	2
2	**4**	**6**

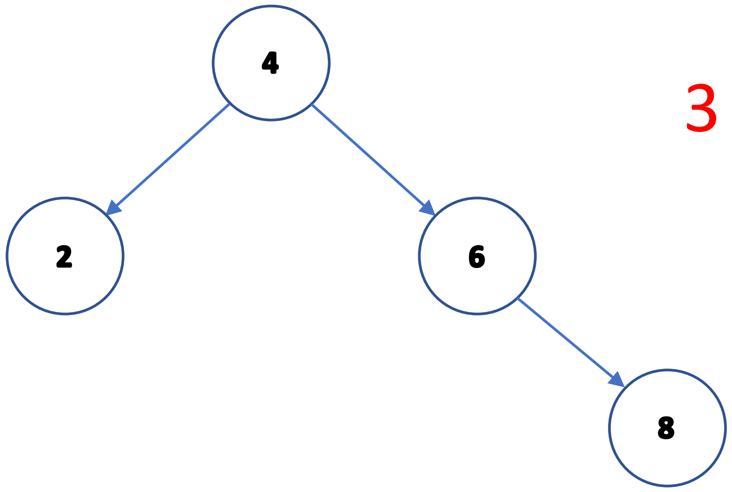


2 iterações

Pior Caso – Situações mais simples

- n = 4

0	1	2	3
2	**4**	**6**	**8**

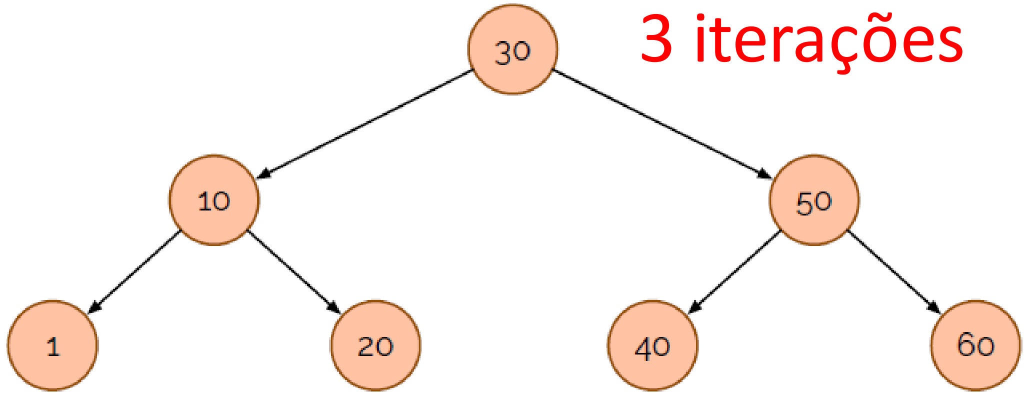


3 iterações

- ...

- n = 7

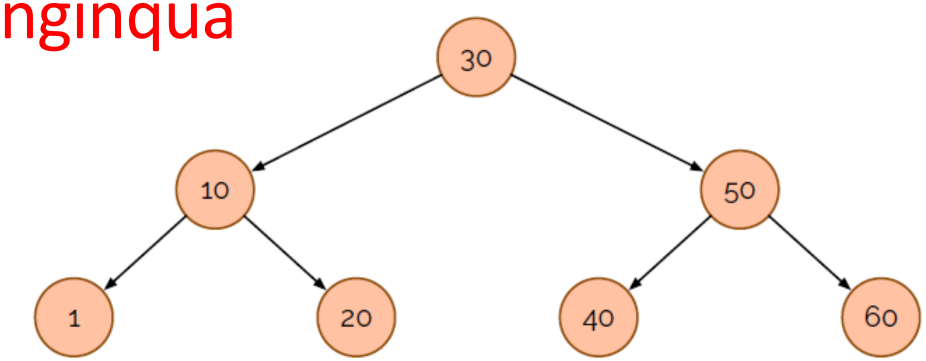
0	1	2	3	4	5	6
1	**10**	**20**	**30**	**40**	**50**	**60**



3 iterações

Pior Caso – Valor procurado pertence ao array

- Percorrer a árvore até atingir a folha mais longínqua



- Array com 1 elemento : 1 iteração
- Array com 2 elementos : 2 iterações
- Array com 3 elementos : 2 iterações
- Array com 4 elementos : 3 iterações
- …
- Array com 7 elementos : 3 iterações
- Array com 8 elementos : 4 iterações

Array com n elementos : ?

Pior Caso

- Há alguma diferença se o valor procurado não pertencer ao array ?
- Ou o nº de iterações é o mesmo, sempre que procurarmos um valor que não pertence ao array ?

- No pior caso, o nº de iterações é determinado pela altura da árvore binária associada

$$W(n) = 1 + \lfloor \log_2 n \rfloor = 1 + \text{floor}(\log n)$$

$$\boldsymbol{W(n) \in O(\log n)}$$

Caso Médio – Valor procurado pertence ao array

- Cenário experimental

- Procurar uma vez cada um dos valores registados no array
- Qual o nº médio de iterações realizadas ?

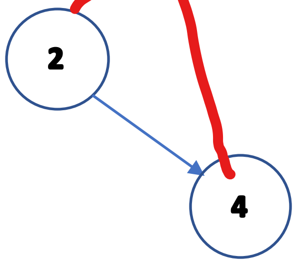
- Análise formal

- Equiprobabilidade
- Casos particulares : array com $2^k - 1$ elementos

Caso Médio – Situações mais simples

- $n = 2$

0	1
2	**4**

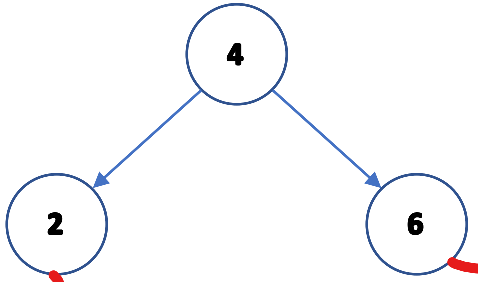


$(1 + 2) / 2 = 1.5$

1.5 iterações

- $n = 3$

0	1	2
2	**4**	**6**



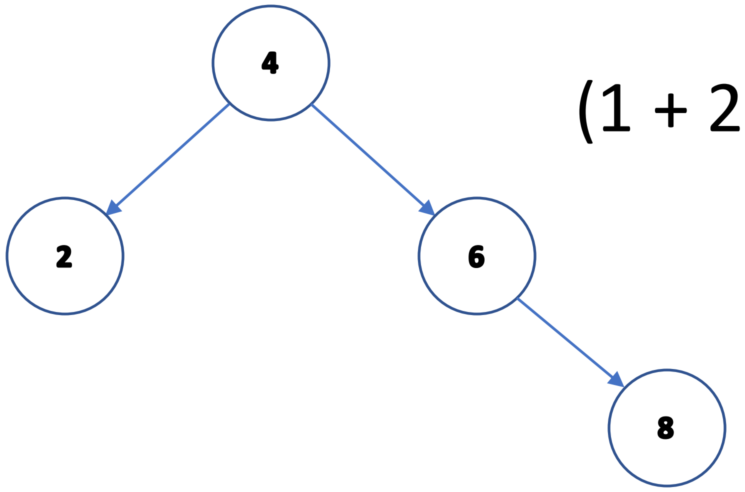
$(1 + 2 + 2) / 3 = 1.67$

1.67 iterações

Caso Médio – Situações mais simples

- n = 4

0	1	2	3
2	**4**	**6**	**8**



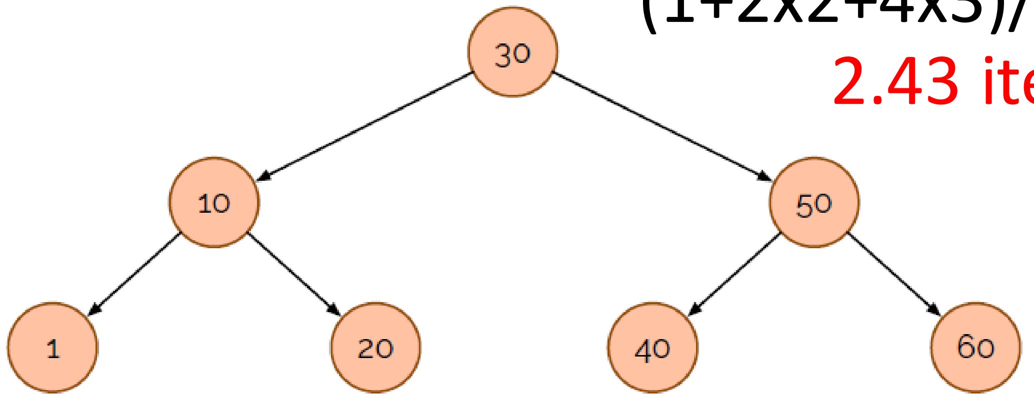
(1 + 2 + 2 + 3) / 4 = 2

2 iterações

- …

- n = 7

0	1	2	3	4	5	6
1	**10**	**20**	**30**	**40**	**50**	**60**



(1+2x2+4x3)/7=2.43

2.43 iterações

Caso Médio – Generalização

- Valor procurado pertence ao array – Equiprobabilidade
- Caso particular : **$n = 2^k - 1$, $k = \log_2(n + 1)$**
- Representação em árvore binária : árvore tem **k níveis**

Índice do nível	Nº de nós	Nº de iterações
0	1	1
1	2	2
2	4	3
3	8	4
…	…	…

Caso Médio – Generalização

- Valor procurado pertence ao array – Equiprobabilidade
- Caso particular : **n = 2^{k} – 1, k = log$_2$(n + 1)**
- Nº de níveis da árvore binária = k

$$A(n) = \sum_{i=0}^{k-1} \frac{1}{n} \times 2^i \times (i+1) = \frac{1}{n}\left[\sum_{i=0}^{k-1} i \times 2^i + \sum_{i=0}^{k-1} 2^i\right]$$

Caso Médio – Generalização

- Expressões auxiliares

$$\sum_{i=0}^{k-1} 2^i = 2^k - 1 \qquad\qquad \sum_{i=0}^{k-1} i \times 2^i = 2^k(k - 2) + 2$$

$$\boldsymbol{A(n) = \frac{1}{n}\left[2^k(k - 1) + 1\right] = \frac{k \times 2^k - \left(2^k - 1\right)}{n} = \frac{k \times (n + 1)}{n} - 1}$$

$$\boldsymbol{A(n) = k + \frac{k}{n} - 1 = k - \left(1 - \frac{k}{n}\right) = \log_2(n + 1) - \left(1 - \frac{\log_2(n + 1)}{n}\right)}$$

Comparação – Situações mais simples

- n = 1 1 iteração
- n = 3 1.67 iterações
- n = 7 2.43 iterações

- A(1) = log 2 – 1 + (log 2)/1 = 1
- A(3) = log 4 – 1 + (log 4)/3 = 1 + 2/3 **OK!**
- A(7) = log 8 – 1 + (log 8)/7 = 2 + 3/7

Comparação – Caso Médio vs Pior Caso

- Caso particular : $n = 2^k - 1$

- $W(3) = 1 + \text{floor}(\log 3) = 2$
- $A(3) = \log 4 - 1 + (\log 4)/3 = 1 + 2/3$ $\qquad$ $A(3) = W(3) - 1/3$

- $W(7) = 1 + \text{floor}(\log 7) = 3$
- $A(7) = \log 8 - 1 + (\log 8)/7 = 2 + 3/7$ $\qquad$ $A(7) = W(7) - 4/7$

- $W(n) = 1 + \text{floor}(\log n) = \log(n + 1)$
- $A(n) = \log(n + 1) - 1 + \log(n + 1)/n = W(n) - 1 + \log(n + 1)/n \approx W(n) - 1$

E se puder não pertencer ao array ?

Casos possíveis		Nº de iterações	Probabilidade
Sucesso 0	É o 1º elemento	?	1 / (2 n +1)
Sucesso 1	É o 2º elemento	?	1 / (2 n +1)
…	…	…	…
Sucesso (n – 1)	É o último elemento	?	1 / (2 n +1)
Insucesso 0	Menor do que o 1º	Nº de níveis da árvore	1 / (2 n +1)
Insucesso 1	Entre o 1º e o 2º	Nº de níveis da árvore	1 / (2 n +1)
…	…	…	…
Insucesso (n – 1)	Entre o penúltimo e o último	Nº de níveis da árvore	1 / (2 n +1)
Insucesso n	Maior do que o último	Nº de níveis da árvore	1 / (2 n +1)

Caso Médio – Generalização

- Caso particular : **n = 2^k – 1, k = log$_2$(n + 1)**
- Nº de níveis da árvore binária = k

$$A(n) = \frac{1}{2n+1}\left[\sum_{i=0}^{k-1} 2^i \times (i+1) + (n+1) \times k\right]$$

$$\mathbf{A(n)} = \frac{1}{2n+1}[k \times (n+1) - n + (n+1) \times k]$$

$$A(n) = \frac{1}{2n+1}[2k(n+1) - n] \approx k - \frac{1}{2} \approx \log_2(n+1) - \frac{1}{2}$$

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 ≈ 2 x nº de iterações
- Caso médio : valores aproximados
- Valores procurados podem não estar no array (Cenário 2)

Resumo

- Estratégia "diminuir-para-reinar" – "decrease-and-conquer"
- Representar as possíveis situações com o auxílio de uma árvore binária
- Pior caso: percurso da raiz até à folha mais longínqua
- $W(n) = 1 + \lfloor \log_2 n \rfloor$ Nº de níveis da árvore binária
- Caso particular : **$n = 2^k - 1$, $k = \log_2(n + 1)$**
- Cenário 1 : $A(n) \approx W(n) - 1$
- Cenário 2 : $A(n) \approx W(n) - \frac{1}{2}$ $\boldsymbol{O(\log n)}$
- Para um n qualquer, obtemos expressões "semelhantes"

Algoritmos de Ordenação

Algoritmos de Ordenação

- Para quê ordenar os elementos de um conjunto/multi-conjunto ?
- Para facilitar operações de procura, intersecção, reunião, etc.

- Para simplificar, vamos considerar arrays de números inteiros

- MAS, em geral, cada elemento do array é um registo com uma chave de ordenação
- Função auxiliar para a comparação de duas chaves: -1, 0, 1

Critérios de ordem habituais

- Ordem crescente : a[i-1] < a[i]

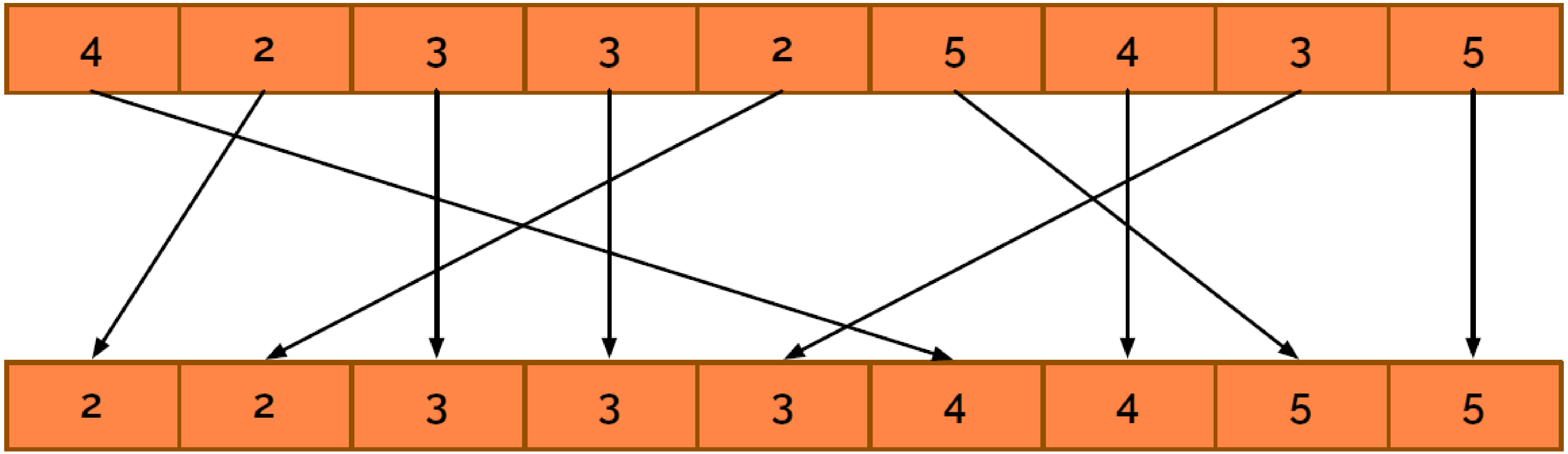
0	1	2	3	4	5
2	**4**	**6**	**8**	**10**	**12**

- Ordem não-decrescente : a[i-1] <= a[i]

0	1	2	3	4	5
2	**4**	**4**	**8**	**10**	**10**

Algoritmo de ordenação estável

- Preservar a ordem relativa inicial de elementos com o mesmo valor



Arrays auxiliares de indexação

- Evitar a troca de elementos/registos que ocupem muitos bytes

0	1	2	3	4	5
99999	**33333**	**22222**	**88888**	**55555**	**11111**

- Trocar apenas os correspondentes índices no array auxiliar

0	1	2	3	4	5
0	**1**	**2**	**3**	**4**	**5**

5	**2**	**1**	**4**	**3**	**0**

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

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**

- 4 comparações

Exemplo

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

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

- 4 comparações
- 1 troca

Exemplo

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

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

- 4 + 3 comparações
- 1 troca

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**

- 4 + 3 comparações
- 1 + 1 trocas

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**

- 4 + 3 + 2 comparações
- 1 + 1 trocas

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**

- 4 + 3 + 2 comparações
- 1 + 1 + 0 trocas

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**

- 4 + 3 + 2 + 1 comparações
- 1 + 1 + 0 trocas

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**

- 1 + 1 + 0 + 1 trocas

Terminado !!

Nº de operações efetuadas ?

- Nº de comparações = ?
- (n – 1) + (n – 2) + … + 2 + 1 = n x (n – 1) / 2 **$O(n^2)$**

- Nº de trocas = ?
- Melhor caso : Bt(n) = 0 - Quando ?
- Pior caso : Wt(n) = n – 1 - Quando ? **$O(n)$**

Trocar

```
void swap( int* a, int* b )          // Call-by-pointer
{
        int temp = *a;
        *a = *b;
        *b = temp;
}
```

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 !!

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

O(n²)

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

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

Nº de Trocas – Caso Médio

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

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

$$\boldsymbol{A_t(n) = n - H_n \approx n - \ln n}$$

O(n)

Sugestão de leitura

Sugestão de leitura

- J. J. McConnell, Analysis of Algorithms, 1st Edition, 2001
  - Capítulo 2: secção 2.2