

# ALGORITMOS E ESTRUTURAS DE DADOS

algoritmos de ordenamento

2010-2011

Carlos Lisboa Bento

## Algoritmos de ordenamento

conceitos

Objectivo do ordenamento

### Ficheiro

Uma operação frequente sobre um ficheiro é a pesquisa

Pesquisa: localização de um registo num ficheiro (aceder ao registo)

Antunes, João A.	R. P. António Viera, 23	720456
Baptista, Vitor C.	R. Carlos Seixas, 9, 6º	705423
Melo, Eurico R.	Quinta Nova, L12, B	346512
Pereira, Maria A.	Lrg. da Portagem, 12, 4º	20345
Silva, José A.	R. das Padeira, 23, 3º	816524

Exemplos de conjuntos ordenados:

- Uma pauta de exame
- Uma lista telefónica
- Um dicionário

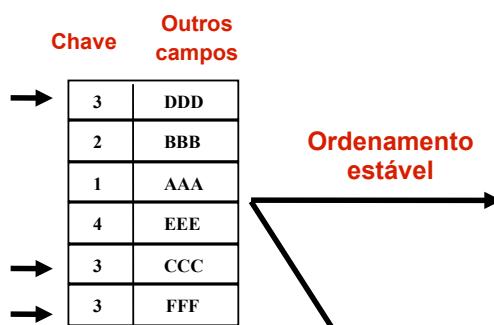
Objectivo do ordenamento

Facilitar a pesquisa

# Algoritmos de ordenamento

conceitos

Pode haver vários registos com a mesma chave



Ordenamento estável

Chave	Outros campos
1	AAA
2	BBB
3	DDD
3	CCC
3	FFF
4	EEE

Ordenamento não estável

Chave	Outros campos
1	AAA
2	BBB
3	CCC
3	DDD
3	FFF
4	EEE

3

Ordenamento estável: se para todos os registos i e j tais que K(i) = K(j), se r(i) preceder r(j) no ficheiro original, então r(i) precede r(j) no ficheiro ordenado.

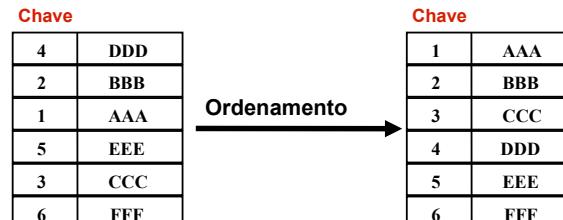
# Algoritmos de ordenamento

conceitos

Ordenamento de registos/endereço

Ordenamento por registos:

o ordenamento tem lugar nos próprios registos do ficheiro

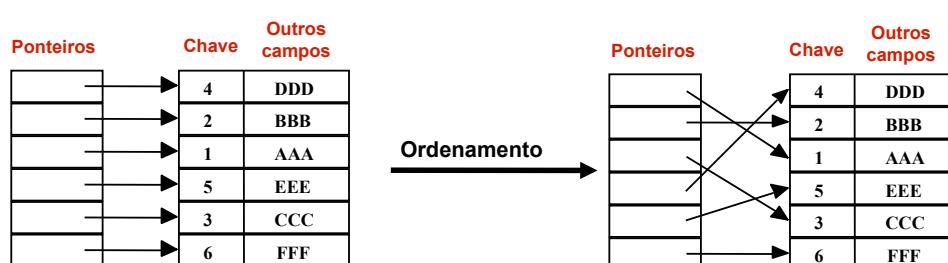


Ordenamento

Particularmente útil quando o tamanho dos registos é grande

Ordenamento por endereço:

o ordenamento é feito numa tabela de ponteiros



Ordenamento

4

# Algoritmos de ordenamento

conceitos

Eficiência de um método de ordenamento

Critérios:

- **Tempo necessário à escrita do programa;**
- **Tempo de ordenamento (execução do programa);**
- **Espaço em memória ocupado pelo programa.**

- Análise do algoritmo
- Projecto do programa
- Escrita
- Validação

- Número de comparações
- Número de atribuições

Tempo de ordenamento

Frequentemente considerado como o critério mais importante

Determinação do tempo médio de ordenamento de um dado algoritmo

Método analítico

Método experimental

# Algoritmos de ordenamento

conceitos

Determin. analítica do tempo médio de ordenamento num fich de dim n

Método analítico

Através da análise matemática do algoritmo de ordenamento em estudo, incluindo a análise de diversos casos:

- o melhor caso;
- o pior caso;
- caso médio

- Número de comparações
- Número de atribuições

Desvantagens:

- A análise é, por vezes, muito complexa;
- Nem sempre é possível determinar um valor médio através do estudo de casos extremos;
- É muito difícil lidar com a dependência do tempo de ordenamento relativamente à sequência original de dados.

# Algoritmos de ordenamento

conceitos

Determin. experimental do tempo médio de ordenamento num fich. de dim. n

Método experimental

Correr o programa para um número significativo de ficheiros de várias dimensões e medir o tempo de execução do ordenamento para cada caso.



Estimativa do tempo médio de ordenamento de um ficheiro de tamanho n (genérico)

Desvantagem:

- Pode ser difícil generalizar os resultados experimentais devido à dependência do tempo de ordenamento relativamente à sequência original de dados;

# Algoritmos de ordenamento

conceitos

Escolha de um método de ordenamento

- A escolha de um método de ordenamento é, em geral, uma decisão de **compromisso**: não há um método de ordenamento universalmente superior a todos os outros.
- O método mais **adequado** depende da **situação concreta**.
- É importante **ter em conta**:
  - A **dimensão do ficheiro** a ordenar (nº de registo, tamanho dos registo);
  - **Número de vezes** que o programa vai ser **utilizado**.

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

Algoritmos Elementares

- Inserção
- Selecção
- Troca (bubble sort)

*Estudados no 1º ano*

Algoritmos Eficientes

- Shell
- Rápido (quicksort)
- Mergesort
- Heap
- Radix Sort

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

9

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

Ordenamento por Inserção

Conceitos

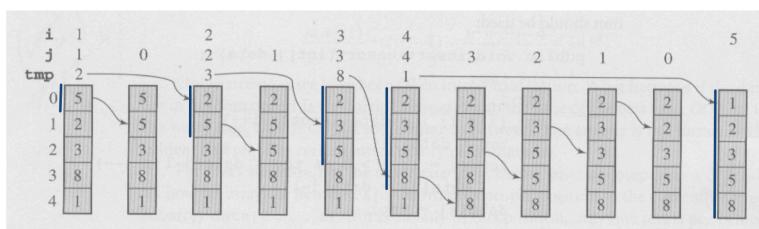
Método

Num ordenamento por inserção os registos são inseridos na posição correcta de uma sequência ordenada já existente.

Algoritmos Elementares

- Inserção
- Selecção
- Troca (bubble sort)

*Revisão*



Data Structures and Algorithms in JAVA, Adam Drozdek

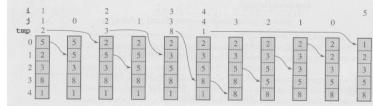
# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Inserção

Implementação



### Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

```
insertionsort(data[]) {  
    for (i = 1; i < data.length; i++)  
        tmp = data[i];  
        move all elements data[j] greater than tmp by one position;  
        place tmp in its proper position;
```

Data Structures and Algorithms in JAVA, Adam Drozdek

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Inserção

Implementação

### Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

```
public void insertionsort(Object[] data) {  
    Comparable tmp;  
    int i, j;  
    for (i = 1; i < data.length; i++) {  
        tmp = (Comparable)data[i];  
        for (j = i; j > 0 && tmp.compareTo(data[j-1]) < 0; j--)  
            data[j] = data[j-1];  
        data[j] = tmp;  
    }  
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Inserção

Complexidade

Considerações Preliminares

- Só movimenta elementos no array quando necessário.
- Quando o array está ordenado poucas atribuições – só  $data[i] > tmp > data[i]$
- Elementos que podem já estar na posição correcta podem voltar a ser deslocados – ex. 2 e 3
- Quando um elemento necessida de ser repositcionado todos os elementos maiores têm de ser reposicionados.
- Dado que um elemento que já está na posição correcta pode ser temporariamente “desalojado” resulta em movimentos redundantes relativamente ao resultado final.

```
public void insertionsort(Object[] data) {  
    Comparable tmp;  
    for (i = 1; i < data.length; i++) {  
        tmp = (Comparable) data[i];  
        for (j = i; j > 0 && tmp.compareTo(data[j-1]) < 0; j--)  
            data[j] = data[j-1];  
        data[j] = tmp;  
    }  
}
```

# Algoritmos de ordenamento

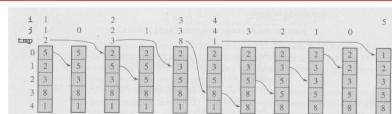
conceitos

Métodos de Ordenamento

## Ordenamento por Inserção

Complexidade

Comparações realizadas



MELHOR CASO – array previamente ordenado

Uma só comparação para cada posição  $\rightarrow n-1$  comparações  $\rightarrow O(n)$

PIOR CASO – array em ordem inversa

Para cada ciclo externo i ocorrem i comparações  $\rightarrow n$ . total de comparações

$$\sum_{i=1}^{n-1} i = 1 + 2 + \dots + (n-1) = \frac{n(n-1)}{2} \rightarrow O(n^2)$$

CASO MÉDIO – array aleatoriamente ordenado

$$\sum_{i=1}^{n-1} \frac{i+1}{2} = \frac{1}{2} \sum_{i=1}^{n-1} i + \frac{1}{2} \sum_{i=1}^{n-1} 1 = \frac{\frac{1}{2} n(n-1)}{2} + \frac{1}{2} (n-1) = \frac{n^2 + n - 2}{4} \rightarrow O(n^2)$$

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Inserção

Complexidade

Atribuições realizadas

```
public void insertionsort(Object[] data) {
    Comparable tmp;
    int i, j;
    for (i = 1; i < data.length; i++) {
        tmp = (Comparable)data[i];
        for (j = i; j > 0 && tmp.compareTo(data[j-1]) < 0; j--)
            data[j] = data[j-1];
        data[j] = tmp;
    }
}
```

MELHOR CASO – array previamente ordenado

Para cada comparação temos as atribuições  $data[i] = tmp = data[i]$   $\rightarrow 2(n-1)$  atribuições  $\rightarrow O(n)$

PIOR CASO – array em ordem inversa

Para cada ciclo externo  $i$  ocorrem  $i$  atribuições + as referentes a  $tmp$  no ciclo externo  $\rightarrow n.$  total de comparações

$$\sum_{i=1}^{n-1} i = 1 + 2 + \dots + (n-1) = \frac{n(n-1)}{2} = \frac{n(n-1)}{2} + 2(n-1) = \frac{n^2 + 3n - 4}{2} \rightarrow O(n^2)$$

CASO MÉDIO – array aleatoriamente ordenado

$$\sum_{i=1}^{n-1} \frac{i+1}{2} + 2 = \frac{1}{2} \sum_{i=1}^{n-1} i + \sum_{i=1}^{n-1} \frac{3}{2} = \frac{\frac{1}{2}n(n-1)}{2} + \frac{3}{2}(n-1) = \frac{n^2 + 5n - 6}{4} \rightarrow O(n^2)$$

© Dep. De Engenharia Informática - Univ. de Coimbra

Programação e Algoritmos III

8 - 17

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

15

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento

Conceitos

Métodos de Ordenamento

Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Algoritmos Eficientes

- Shell
- Heap
- Rápido (quicksort)
- Mergesort
- Radix

Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

© Dep. De Engenharia Informática - Univ. de Coimbra

Programação e Algoritmos III

8 - 18

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

16

# Algoritmos de ordenamento

conceitos

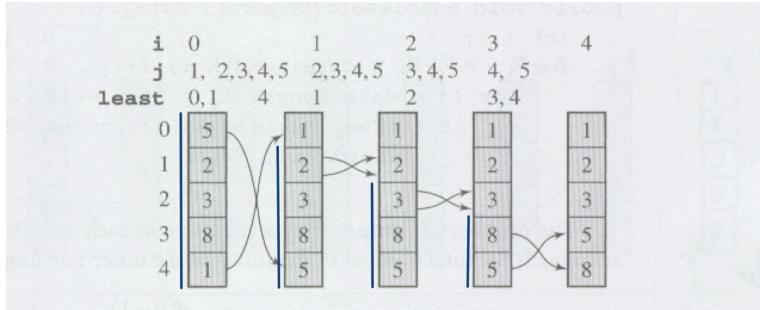
Métodos de Ordenamento

## Ordenamento por Selecção

Conceitos

Método

Num ordenamento por selecção os elementos sucessivos são seleccionados da sequência original e colocados na sua posição correcta.



Data Structures and Algorithms in JAVA, Adam Drozdek

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

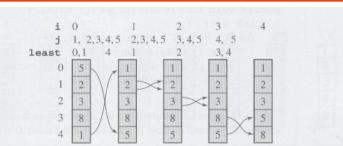
## Ordenamento por Selecção

Implementação

Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão



```
selectionsort(data[])
    for(i = 0; i < data.length-1; i++)
        select the smallest element among data[i],...,data[data.length-1];
        swap it with data[i];
```

Data Structures and Algorithms in JAVA, Adam Drozdek

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Selecção

Implementação

### Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

```
public void selectionsort(Object[] data) {  
    int i,j,least;  
    for (i = 0; i < data.length-1; i++) {  
        for (j = i+1, least = i; j < data.length; j++)  
            if (((Comparable)data[j]).compareTo(data[least]) < 0)  
                least = j;  
        swap(data,least,i);  
    }  
}
```

```
void swap(Object[] a, int e1, int e2) {  
    Object tmp = a[e1]; a[e1] = a[e2]; a[e2] = tmp;  
}
```

*Data Structures and Algorithms in JAVA, Adam Drozdek*

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Selecção

Complexidade

### Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

Comparações realizadas

```
public void selectionsort(Object[] data) {  
    int i,j,least;  
    for (i = 0; i < data.length-1; i++) {  
        for (j = i+1, least = i; j < data.length; j++)  
            if (((Comparable)data[j]).compareTo(data[least]) < 0)  
                least = j;  
        swap(data,least,i);  
    }  
}
```

CASO GERAL – temos um ciclo dentro de outro ciclo em que o ciclo exterior com o contador i tem n-1 iterações e o ciclo interior tem (n-1) - i iterações

$$\sum_{i=0}^{n-2} (n-1-i) = (n-1) + \dots + 1 = \frac{n(n-1)}{2} \rightarrow O(n^2)$$

Atribuições realizadas

PIOR CASO – array com o maior elemento está na primeira posição e o resto do array está ordenado.

Temos

3 (n-1)  $\Rightarrow O(n)$

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento

Métodos de Ordenamento

Conceitos

Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Algoritmos Eficientes

- Shell
- Heap
- Rápido (quicksort)
- Mergesort
- Radix

© Dep. De Engenharia Informática - Univ. de Coimbra

Programação e Algoritmos III

8 - 23

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

21

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Troca

Conceitos

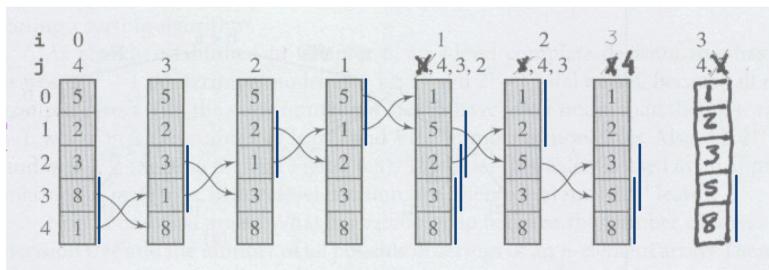
Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

Método

- 1) Percorrer o ficheiro várias vezes, sequencialmente;
- 2) Em cada passagem compara-se cada elemento com o seguinte ( $x[i]$  com  $x[i+1]$ ) e troca-se estes dois elementos se não estiverem na ordem correcta.



Data Structures and Algorithms in JAVA, Adam Drozdek

© Dep. De Engenharia Informática - Univ. de Coimbra

Programação e Algoritmos III

8 - 24

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

22

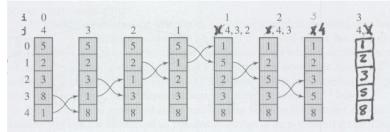
# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Troca

Implementação



### Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

```
bubblesort(data[])
    for (i = 0; i < data.length-1; i++)
        for (j = data.length-1; j > i; --j)
            swap elements in position j and j-1 if they are out of order;
```

Data Structures and Algorithms in JAVA, Adam Drozdek

© Dep. De Engenharia Informática - Univ. de Coimbra

Programação e Algoritmos III

8 - 25

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

23

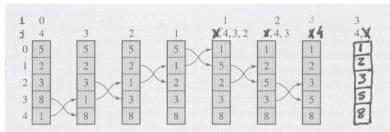
# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Troca

Implementação



### Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

```
public void bubblesort(Object[] data) {
    int i,j;
    for (i = 0; i < data.length-1; i++)
        for (j = data.length-1; j > i; --j)
            if (((Comparable)data[j]).compareTo(data[j-1]) < 0)
                swap(data,j,j-1);
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

© Dep. De Engenharia Informática - Univ. de Coimbra

Programação e Algoritmos III

8 - 26

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

24

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

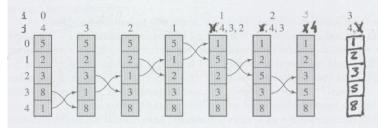
## Ordenamento por Troca

Complexidade

Considerações Preliminares

- O algoritmo “avalia” em cada momento unicamente a ordem relativa a dois elementos do array.
- Elementos que a certa altura já estão correctamente posicionados podem ser recolocados.
- Para mover um elemento de um extremo do array para outro executa trocas com todos os elementos do array.

```
public void bubblesort(Object[] data) {  
    int i,j;  
    for (i = 0; i < data.length-1; i++)  
        for (j = data.length-1; j > i; --j)  
            if (((Comparable)data[j]).compareTo(data[j-1]) < 0)  
                swap(data, j, j-1);  
}
```



Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Troca

Complexidade

Algoritmos Elementares

- Inserção
- Seleção
- Troca (bubble sort)

Revisão

Comparações realizadas

CASO GERAL – temos um ciclo dentro de outro ciclo em que o ciclo exterior com o contador i tem n-1 iterações e o ciclo interior tem (n-1) - i iterações

$$\sum_{i=0}^{n-2} (n-1-i) = (n-1) + \dots + 1 = \frac{n(n-1)}{2} \rightarrow O(n^2)$$

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

## Ordenamento por Troca

Complexidade

Atribuições realizadas

```
public void bubblesort(Object[] data) {  
    int i,j;  
    for (i = 0; i < data.length-1; i++)  
        for (j = data.length-1; j > i; --j)  
            if (((Comparable)data[j]).compareTo(data[j-1]) < 0)  
                swap(data,j,j-1);  
}
```

MELHOR CASO – array previamente ordenado

Não há troca de elementos → não há atribuições realizadas

PRIOR CASO – array por ordem inversa. Temos um ciclo dentro de outro ciclo em que o ciclo exterior com o contador i tem n-1 iterações e o ciclo interior tem (n-1) - i iterações

$$\sum_{i=0}^{n-2} 3(n-1-i) = 3(n-1) + \dots + 3 = 3 \frac{n(n-1)}{2} \rightarrow O(n^2)$$

CASO MÉDIO – array aleatoriamente ordenado

$$3 \sum_{i=0}^{n-2} \frac{n-i-1}{2} = \frac{3}{2} \sum_{i=0}^{n-2} (n-1) - \frac{3}{2} \sum_{i=0}^{n-2} i = 3 \frac{(n-1)^2}{2} - 3 \frac{(n-1)(n-2)}{4} = 3 \frac{n(n-1)}{4} \rightarrow O(n^2)$$

© Dep. De Engenharia Informática - Univ. de Coimbra

Programação e Algoritmos III

8 - 29

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

27

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

Inserção

Seleção

Troca

<http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html>

<http://www.sorting-algorithms.com/>

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

28

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

Algoritmos Elementares

- Inserção
- Selecção
- Troca (bubble sort)

*Estudados no 1º ano*

Algoritmos Eficientes

- Shell
- Mergesort
- Rápido (quicksort)
- Heap

# Algoritmos de ordenamento

## conceitos

Shell Sort

Algoritmos Eficientes de Ordenamento

O ordenamento por INSERÇÃO tem complexidade  $O(n)$  se a sequência já estiver ordenada.

Ideia inicial – deverá ser vantajoso primeiro ordenar partes da sequência e depois ordenar a sequência global numa fase que esta já está no seu todo quase ordenada.

# Algoritmos de ordenamento

conceitos

## Shell Sort

1959 Donald Shell

A ideia é evitar movimentações de grandes quantidades de dados. Como?

Começando por comparar elementos que estão bastante distantes entre si e gradualmente ir comparando elementos mais próximos entre si numa sequência de saltos dada por  $h_n, \dots, h_2, h_1$   
- **sequência de incrementos**.

O engenho deste método de ordenamento está centrado na forma como a sequência inicial é dividida em várias subsequências.

$$h = h_n, \dots, 1$$

$h_k$ -sorted  $a[i] \leq a[i+h_k]$  e temos k subsequências

# Algoritmos de ordenamento

conceitos

## Shell Sort 5-3-1

data before 5-sort	10	8	6	20	4	3	22	1	0	15	16
Five subarrays before sorting	10	—	—	—	—	3	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	6	—	—	—	—	—	—	1	—	—	—
	20	—	—	—	—	—	—	—	0	—	—
	4	—	—	—	—	—	—	—	—	15	—

# Algoritmos de ordenamento

conceitos

Shell Sort 5-3-1

data before 5-sort	10	8	6	20	4	3	22	1	0	15	16
Five subarrays before sorting	10	—	—	—	—	3	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	6	—	—	—	—	—	—	1	—	—	—
	20	—	—	—	—	—	—	—	0	—	—
	4	—	—	—	—	—	—	—	—	15	—
Five subarrays after sorting	3	—	—	—	—	10	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	1	—	—	—	—	—	—	6	—	—	—
	0	—	—	—	—	—	—	—	20	—	—
	4	—	—	—	—	—	—	—	—	15	—

# Algoritmos de ordenamento

conceitos

Shell Sort 5-3-1

data before 5-sort	10	8	6	20	4	3	22	1	0	15	16
Five subarrays before sorting	10	—	—	—	—	3	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	6	—	—	—	—	—	—	1	—	—	—
	20	—	—	—	—	—	—	—	0	—	—
	4	—	—	—	—	—	—	—	—	15	—
Five subarrays after sorting	3	—	—	—	—	10	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	1	—	—	—	—	—	—	6	—	—	—
	0	—	—	—	—	—	—	—	20	—	—
	4	—	—	—	—	—	—	—	—	15	—
data after 5-sort and	3	8	1	0	4	10	22	6	20	15	16

# Algoritmos de ordenamento

conceitos

Shell Sort 5-3-1

<b>data before 5-sort</b>	10	8	6	20	4	3	22	1	0	15	16
<b>Five subarrays before sorting</b>	10	—	—	—	—	3	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	6	—	—	—	—	—	—	1	—	—	—
	20	—	—	—	—	—	—	—	0	—	—
	4	—	—	—	—	—	—	—	—	15	—
<b>Five subarrays after sorting</b>	3	—	—	—	—	10	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	1	—	—	—	—	—	—	6	—	—	—
	0	—	—	—	—	—	—	—	20	—	—
	4	—	—	—	—	—	—	—	—	15	—
<b>data after 5-sort and before 3-sort</b>	3	8	1	0	4	10	22	6	20	15	16
<b>Three subarrays before sorting</b>	3	—	—	0	—	—	22	—	—	15	—
	8	—	—	—	4	—	—	6	—	—	16
	1	—	—	—	10	—	—	—	20	—	—

# Algoritmos de ordenamento

conceitos

Shell Sort 5-3-1

<b>data before 5-sort</b>	10	8	6	20	4	3	22	1	0	15	16
<b>Five subarrays before sorting</b>	10	—	—	—	—	3	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	6	—	—	—	—	—	—	1	—	—	—
	20	—	—	—	—	—	—	—	0	—	—
	4	—	—	—	—	—	—	—	—	15	—
<b>Five subarrays after sorting</b>	3	—	—	—	—	10	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	1	—	—	—	—	—	—	6	—	—	—
	0	—	—	—	—	—	—	—	20	—	—
	4	—	—	—	—	—	—	—	—	15	—
<b>data after 5-sort and before 3-sort</b>	3	8	1	0	4	10	22	6	20	15	16
<b>Three subarrays before sorting</b>	3	—	—	0	—	—	22	—	—	15	—
	8	—	—	—	4	—	—	6	—	—	16
	1	—	—	—	10	—	—	—	20	—	—
<b>Three subarrays after sorting</b>	0	—	—	3	—	—	15	—	—	22	—
	4	—	—	6	—	—	8	—	—	—	16
	1	—	—	10	—	—	20	—	—	—	—

# Algoritmos de ordenamento

conceitos

Shell Sort 5-3-1

<b>data before 5-sort</b>	10	8	6	20	4	3	22	1	0	15	16
<b>Five subarrays before sorting</b>	10	—	—	—	—	3	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	6	—	—	—	—	—	—	1	—	—	—
	20	—	—	—	—	—	—	—	0	—	—
	4	—	—	—	—	—	—	—	—	15	—
<b>Five subarrays after sorting</b>	3	—	—	—	—	10	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	1	—	—	—	—	—	—	6	—	—	—
	0	—	—	—	—	—	—	—	20	—	—
	4	—	—	—	—	—	—	—	—	15	—
<b>data after 5-sort and before 3-sort</b>	3	8	1	0	4	10	22	6	20	15	16
<b>Three subarrays before sorting</b>	3	—	—	0	—	—	22	—	—	15	—
	8	—	—	—	4	—	—	6	—	—	16
	1	—	—	—	—	10	—	—	20	—	—
<b>Three subarrays after sorting</b>	0	—	—	3	—	—	15	—	—	22	—
	4	—	—	6	—	—	8	—	—	16	—
	1	—	—	—	10	—	—	20	—	—	—
<b>data after 3-sort and before 1-sort</b>	0	4	1	3	6	10	15	8	20	22	16
<b>data after 1-sort</b>	0	1	3	4	6	8	10	15	16	20	22

# Algoritmos de ordenamento

conceitos

Shell Sort 5-3-1

<b>data before 5-sort</b>	10	8	6	20	4	3	22	1	0	15	16
<b>Five subarrays before sorting</b>	10	—	—	—	—	3	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	6	—	—	—	—	—	—	1	—	—	—
	20	—	—	—	—	—	—	—	0	—	—
	4	—	—	—	—	—	—	—	—	15	—
<b>Five subarrays after sorting</b>	3	—	—	—	—	10	—	—	—	—	16
	8	—	—	—	—	—	22	—	—	—	—
	1	—	—	—	—	—	—	6	—	—	—
	0	—	—	—	—	—	—	—	20	—	—
	4	—	—	—	—	—	—	—	—	15	—
<b>data after 5-sort and before 3-sort</b>	3	8	1	0	4	10	22	6	20	15	16
<b>Three subarrays before sorting</b>	3	—	—	0	—	—	22	—	—	15	—
	8	—	—	—	4	—	—	6	—	—	16
	1	—	—	—	—	10	—	—	20	—	—
<b>Three subarrays after sorting</b>	0	—	—	3	—	—	15	—	—	22	—
	4	—	—	6	—	—	8	—	—	16	—
	1	—	—	—	10	—	—	20	—	—	—
<b>data after 3-sort and before 1-sort</b>	0	4	1	3	6	10	15	8	20	22	16
<b>data after 1-sort</b>	0	1	3	4	6	8	10	15	16	20	22

# Algoritmos de ordenamento

implementação

## Shell Sort

```
void Shellsort (Object[] data) {  
    int i, j, k, h, hCnt, increments[] = new int[20];  
    Comparable tmp;  
    // create an appropriate number of increments h  
    for (h = 1, i = 0; h < data.length; i++) {  
        increments[i] = h;  
        h = 3*h + 1;  
    }  
    // loop on the number of different increments h  
    for (i--; i >= 0; i--) {  
        h = increments[i];  
        // loop on the number of subarrays h-sorted in ith pass  
        for (hCnt = h; hCnt < 2*h; hCnt++) {  
            // insertion sort for subarray containing every hth element  
            // of array data  
            for (j = hCnt; j < data.length; ) {  
                tmp = (Comparable)data[j];  
                k = j;  
                while (k-h >= 0 && tmp.compareTo(data[k-h]) < 0) {  
                    data[k] = data[k-h];  
                    k -= h;  
                }  
                data[k] = tmp;  
                j += h;  
            }  
        }  
    }  
}
```

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

39

# Algoritmos de ordenamento

conceitos

## Shell Sort – O problema da escolha da sequência h

- Qualquer sequência decrescente com  $h_1=1$  pode ser utilizada.
- Sequências com um maior número de elementos resultam no melhoramento de desempenho.
- No entanto, Donald Knuth provou que, com unicamente dois elementos na sequência,  $h_2 = (16n/\pi)^{1/3}$  e  $h_1=1$  temos  $O(n^{5/3})$  em vez de  $O(n^2)!!$
- Empiricamente obtêm-se bons resultados com

$$h_1 = 1 \quad h_{i+1} = 3 h_i + 1$$

terminando com  $h_t$  em que  $h_{t+2} \geq n$

- Outra sequência que empiricamente resulta em bons resultados é a que se obtém pela sequenciação de h baseado no valor 2.2 (dividindo sucessivamente h por 2.2, com  $h_n=a/2$ , sendo a o numero de elementos a ordenar)
- São de evitar sequências do tipo 1, 2, 4, 8 ... ou 1, 3, 6, 9 pois perde-se o efeito de entrelaçamento heterogéneo na formação dos agrupamentos de dados.

© DEI Carlos Lisboa Bento

ALGORITMOS E ESTRUTURAS DE DADOS

40

# Algoritmos de ordenamento

conceitos

Shell Sort

## Resultados empíricos

N	Insertion Sort	Shellsort		
		Shell's	Odd Gaps Only	Dividing by 2.2
1,000	122	11	11	9
2,000	483	26	21	23
4,000	1,936	61	59	54
8,000	7,950	153	141	114
16,000	32,560	358	322	269
32,000	131,911	869	752	575
64,000	520,000	2,091	1,705	1,249

## Resultados analíticos

Não disponíveis.

# Algoritmos de ordenamento

conceitos

Shell Sort

## Resultados empíricos

	10,000			20,000		
	Ascending	Random	Descending	Ascending	Random	Descending
insertionSort	.05	15.82	26.92	.06	1.2	1 m 14.83 4.7
selectionSort	46.86	51.46	59.91	3 m 13.5	4.1	3 m 44.92 4.4
bubbleSort	43.06	1 m 11.24	1 m 13.05	2 m 58.51	4.1	4 m 47.81 4.0
ShellSort	.11	.22	.11	.22	2.0	.49 2.2
heapSort	.38	.39	.28	.72	1.9	.88 2.3
mergesort	.27	.33	.22	.50	1.8	.66 2.0
quicksort	.11	.17	.11	.16	1.5	.44 2.6
quicksort2	.06	.16	.11	.11	1.8	.38 2.4
radixSort	1.20	1.05	1.16	2.31	1.9	1.59 1.5
bitRadixSort	2.97	1.92	2.47	6.54	2.2	4.50 2.7
radixSort2	.11	.11	.11	.21	1.9	.28 2.0
bitRadixSort2	.33	.17	.27	.14	1.3	.44 2.2

# Algoritmos de ordenamento

implementação

Shell Sort na Web

<http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html>

<http://www.sorting-algorithms.com/>

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

Algoritmos Elementares

- Inserção
- Selecção
- Troca (bubble sort)

*Estudados no 1º ano*

Algoritmos Eficientes

- Shell
- Rápido (quicksort)
- Mergesort
- Heap

# Algoritmos de ordenamento

conceitos

Merge Sort

Método:

John von Neumann

Ideia – tornar o processo de partição tão simples quanto possível e gerando subsequências de igual tamanho.

Concentrar o ordenamento no processo de fusão das subsequências.

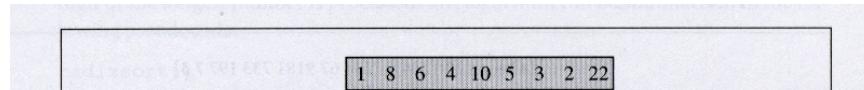
```
mergesort(data)
    if data have at least two elements
        mergesort(left half of data);
        mergesort(right half of data);
        merge(both halves into a sorted list);
```

Data Structures and Algorithms in JAVA, Adam Drozdek

# Algoritmos de ordenamento

conceitos

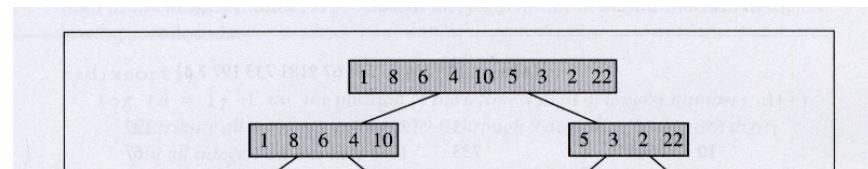
Merge Sort



# Algoritmos de ordenamento

conceitos

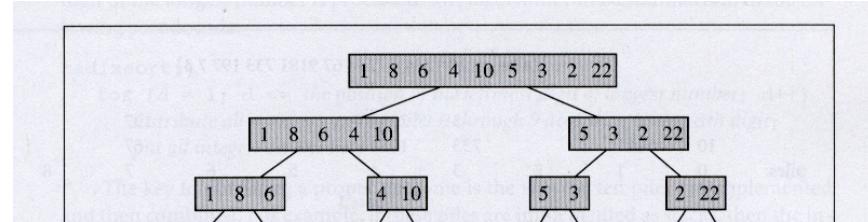
Merge Sort



# Algoritmos de ordenamento

conceitos

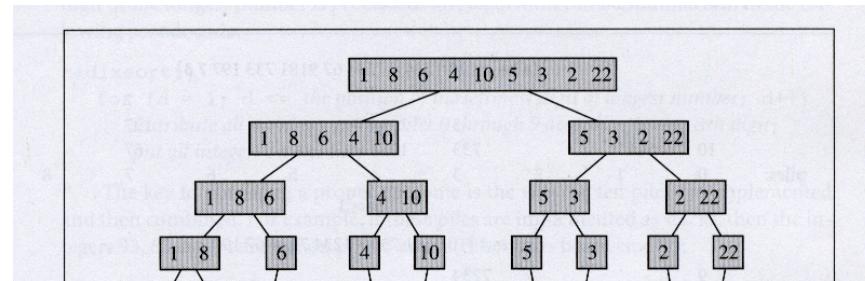
Merge Sort



# Algoritmos de ordenamento

conceitos

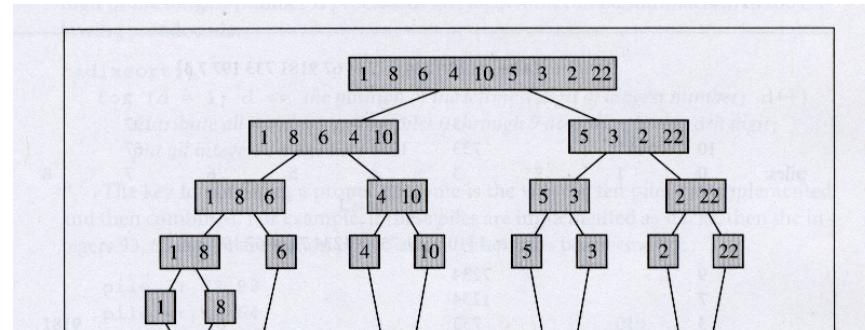
Merge Sort



# Algoritmos de ordenamento

conceitos

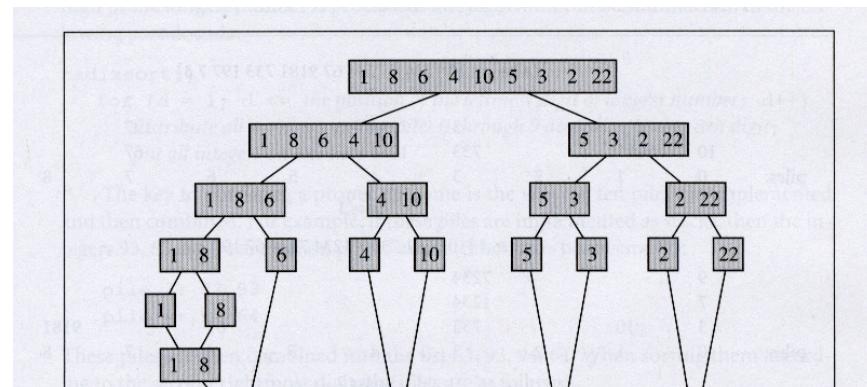
Merge Sort



# Algoritmos de ordenamento

conceitos

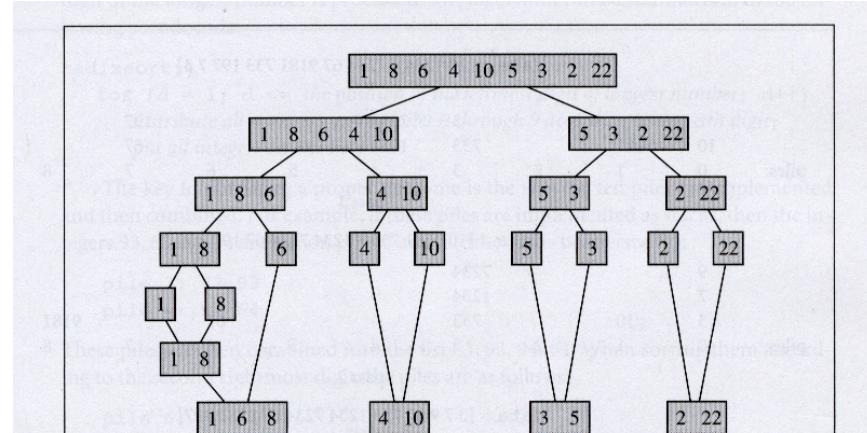
Merge Sort



# Algoritmos de ordenamento

conceitos

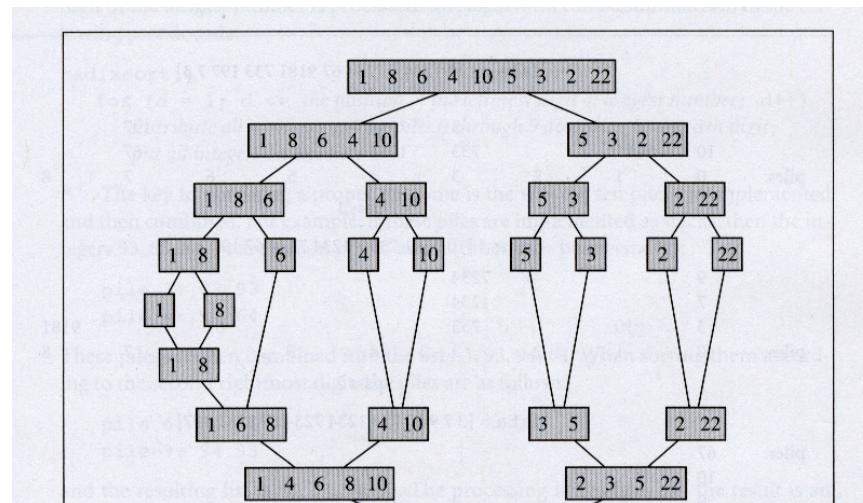
Merge Sort



# Algoritmos de ordenamento

conceitos

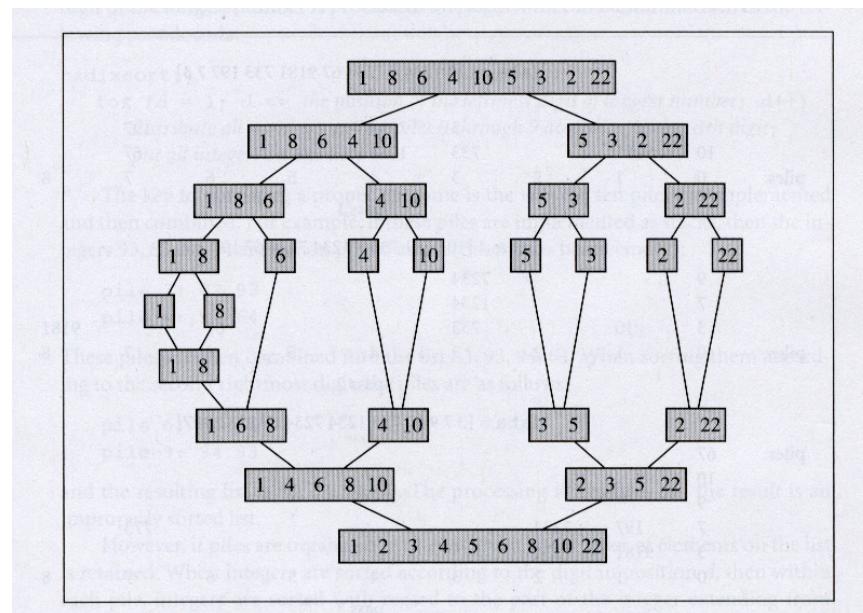
Merge Sort



# Algoritmos de ordenamento

conceitos

Merge Sort



# Algoritmos de ordenamento

conceitos

Merge Sort

## Comparações realizadas

**PIOR CASO** – quando o último elemento de uma subsequência precede unicamente o último elemento da outra sequência. Ex.: [1 6 10 12] e [5 9 11 13]

→  $O(n \log n)$

## Atribuições realizadas

**PIOR CASO**

→  $O(n \log n)$

... mas temos um problema sério

O espaço adicional para armazenamento das subsequências a juntar é incomportável para sequências longas de ordenamento.

# Algoritmos de ordenamento

implementação

Merge Sort na Web

<http://www.sorting-algorithms.com/>

<http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html>

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

Algoritmos Elementares

- Inserção
- Selecção
- Troca (bubble sort)

*Estudados no 1º ano*

Algoritmos Eficientes

- Shell
- Rápido (quicksort)
- Mergesort
- Heap

# Algoritmos de ordenamento

conceitos

Quick Sort

O algoritmo Quicksort(S) é recursivo.

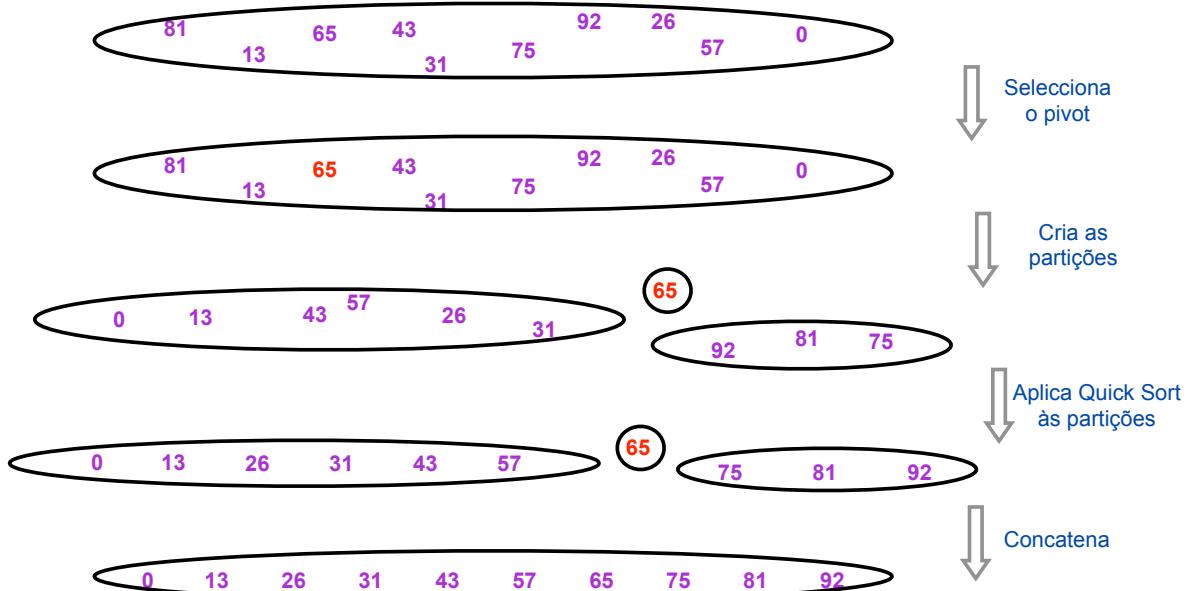
Tendo um conjunto S de elementos para ordenar

- 1) Se  $\#S=0$  ou  $\#S=1$  devolve S.
- 2) Escolhe um elemento  $v$  de S que é designado por *pivot*.
- 3) Particiona  $S-\{v\}$  (todos os elementos de S menos o *pivot*) em dois conjuntos disjuntos  
 $L = \{ x \in S-\{v\} \mid x \leq v \}$  e  
 $R = \{ x \in S-\{v\} \mid x \geq v \}$
- 4) Devolve Quicksort( $L$ ), seguido de  $v$ , seguido de Quicksort( $R$ ).

# Algoritmos de ordenamento

conceitos

Quick Sort



# Algoritmos de ordenamento

conceitos

Quick Sort

Algumas questões:

- Que elemento escolher para pivot ?
- Como construir as partições L e R ?
- O que fazer com elementos iguais ao pivot ?

# Algoritmos de ordenamento

conceitos

## Quick Sort

A considerar:

- Os subproblemas não são necessariamente de igual tamanho o que pode conduzir à degradação de performances.
- O melhor caso ocorre quando o processo de partição gera dois subconjuntos de igual tamanho. Neste caso o tempo de execução é  $O(N \log N)$ .
- O pior caso ocorre quando o pivot é o menor elemento, neste caso o suconjunto L é vazio e o subconjunto R contém todos os elementos menos o pivot. Quando a partição gera repetidamente um subconjunto vazio o tempo de execução é  $O(N^2)$ .
- O caso geral (médio) tem tempo de execução  $O(N \log N)$   
(demonstração em Mark Weiss).

Em conclusão:

- A escolha do pivot é de crucial importância.
- Há que evitar casos degenerados como sejam dados de entrada já ordenados ou em que todos os elementos são iguais.

# Algoritmos de ordenamento

conceitos

## Quick Sort – Escolha do Pivot

Não usar o primeiro elemento da lista como pivot. Porquê?

Uma escolha razoável pode ser o elemento central da lista -  $x[\lfloor \text{low} + \text{high}/2 \rfloor]$ .

Melhor ainda a mediana da partição de comprimento 3 definida pelo primeiro elemento da lista, o elemento central e último elemento.

Ex. [6 1 4 8 9 3 5 2 7 0] -> [6 9 0] -> [0 6 9] -> escolhe o 6

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

Consideremos os índices inicializados com  $i = \text{low}$  e  $j = \text{high}-1$

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

Consideremos os índices inicializados com  $i = \text{low}$  e  $j = \text{high}-1$

**Passo1:**

Troca o pivot com o último elemento.

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

**Consideremos os índices inicializados com  $i = \text{low}$  e  $j = \text{high}-1$**

**Passo1:**

Troca o pivot com o último elemento.

**Passo2:**

Avança  $i$  da esquerda para a direita e  $j$  da direita para a esquerda.

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

**Consideremos os índices inicializados com  $i = \text{low}$  e  $j = \text{high}-1$**

**Passo1:**

Troca o pivot com o último elemento.

**Passo2:**

Avança  $i$  da esquerda para a direita e  $j$  da direita para a esquerda.  
Quando  $i$  encontra um elemento maior que o pivot pára o avanço.

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

**Consideremos os índices inicializados com  $i = \text{low}$  e  $j = \text{high}-1$**

**Passo1:**

Troca o pivot com o último elemento.

**Passo2:**

Avança  $i$  da esquerda para a direita e  $j$  da direita para a esquerda.

Quando  $i$  encontra um elemento maior que o pivot pára o avanço.

Quando  $j$  encontra um elemento menor que o pivot pára o avanço.

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

**Consideremos os índices inicializados com  $i = \text{low}$  e  $j = \text{high}-1$**

**Passo1:**

Troca o pivot com o último elemento.

**Passo2:**

Avança  $i$  da esquerda para a direita e  $j$  da direita para a esquerda.

Quando  $i$  encontra um elemento maior que o pivot pára o avanço.

Quando  $j$  encontra um elemento menor que o pivot pára o avanço.

Se  $i$  e  $j$  ainda não se tiverem cruzado trocam os elementos que referenciam e continuam.

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

**Consideremos os índices inicializados com  $i = \text{low}$  e  $j = \text{high}-1$**

**Passo1:**

Troca o pivot com o último elemento.

**Passo2:**

Avança  $i$  da esquerda para a direita e  $j$  da direita para a esquerda.

Quando  $i$  encontra um elemento maior que o pivot pára o avanço.

Quando  $j$  encontra um elemento menor que o pivot pára o avanço.

Se  $i$  e  $j$  ainda não se tiverem cruzado trocam os elementos que referenciam e continuam.

Enquanto  $i$  e  $j$  não se cruzam continua o processo.

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

**Consideremos os índices inicializados com  $i = \text{low}$  e  $j = \text{high}-1$**

**Passo1:**

Troca o pivot com o último elemento.

**Passo2:**

Avança  $i$  da esquerda para a direita e  $j$  da direita para a esquerda.

Quando  $i$  encontra um elemento maior que o pivot pára o avanço.

Quando  $j$  encontra um elemento menor que o pivot pára o avanço.

Se  $i$  e  $j$  ainda não se tiverem cruzado trocam os elementos que referenciam e continuam.

Enquanto  $i$  e  $j$  não se cruzam continua o processo.

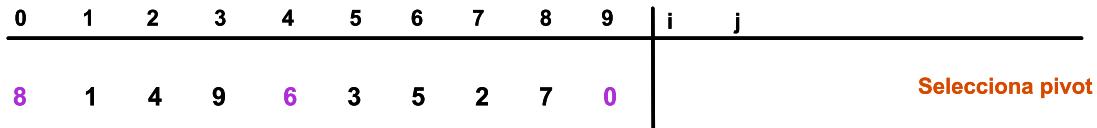
**Passo3:**

Troca o elemento referenciado por  $i$  com o pivot.

# Algoritmos de ordenamento

conceitos

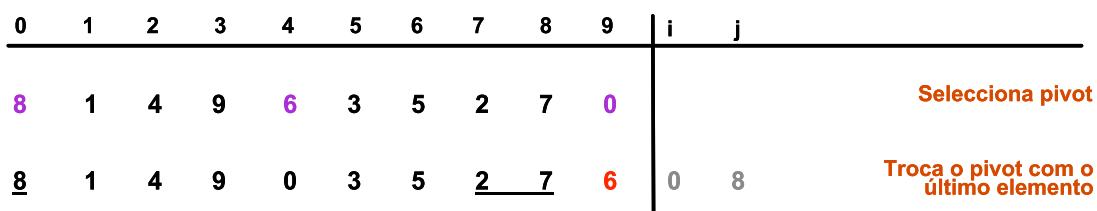
Quick Sort - Rearranjo



# Algoritmos de ordenamento

conceitos

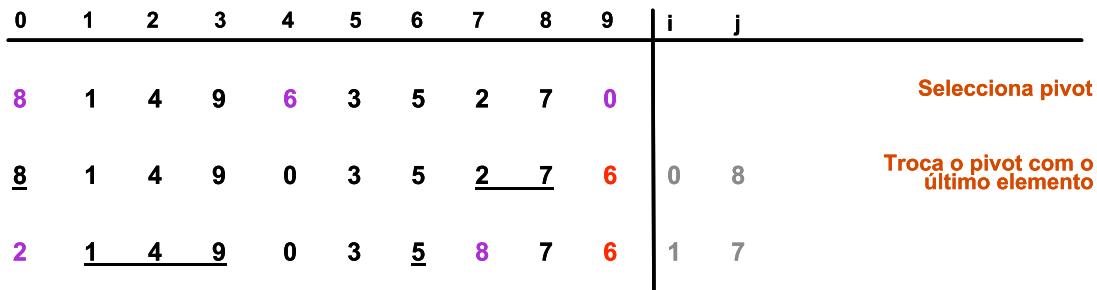
Quick Sort - Rearranjo



# Algoritmos de ordenamento

conceitos

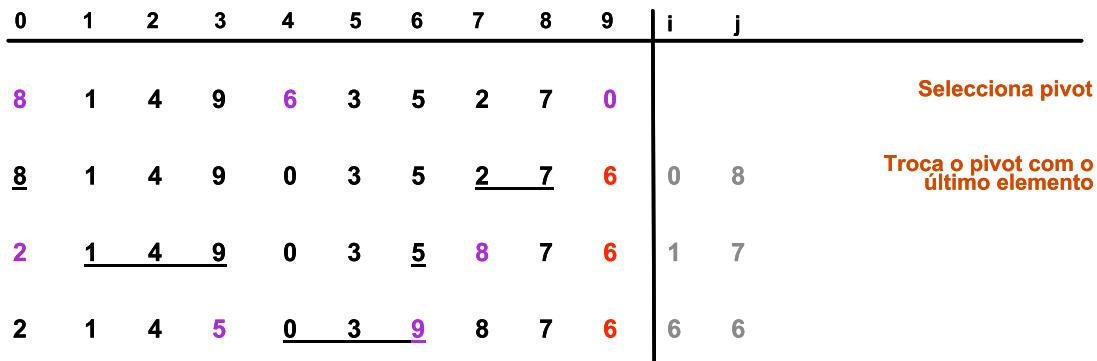
Quick Sort - Rearranjo



# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo



# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

0	1	2	3	4	5	6	7	8	9	i	j
8	1	4	9	6	3	5	2	7	0		
8	1	4	9	0	3	5	<u>2</u>	<u>7</u>	6	0	8
2	<u>1</u>	<u>4</u>	<u>9</u>	0	3	5	8	7	6	1	7
2	1	4	5	<u>0</u>	<u>3</u>	<u>9</u>	8	7	6	6	6
2	1	4	5	0	3	6	8	7	9	6	6

Selecciona pivot

Troca o pivot com o último elemento

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

1. O que fazer quando um elemento é igual ao pivot ?

Deve i parar ou j parar? Se só i parar todos elementos iguais ao pivot vão estar no final no subconjunto à direita do pivot.

# Algoritmos de ordenamento

conceitos

Quick Sort - Rearranjo

## 1. O que fazer quando um elemento é igual ao pivot ?

Deve i parar ou j parar? Se só i parar todos elementos iguais ao pivot vão estar no final no subconjunto à direita do pivot.

**SOLUÇÃO:** param tanto i como j quando encontram elementos iguais ao pivot e trocam os elementos! Parece uma má opção (imaginemos as trocas que têm lugar quando todos os elementos são iguais), mas é aquela que conduz a complexidade  $O(N \log N)$ . Porquê?

# Algoritmos de ordenamento

conceitos

Quick Sort – Rearranjo (algumas questões)

## 1. O que fazer quando um elemento é igual ao pivot ?

Deve i parar ou j parar? Se só i parar todos elementos iguais ao pivot vão estar no final no subconjunto à direita do pivot.

**SOLUÇÃO:** param tanto i como j quando encontram elementos iguais ao pivot e trocam os elementos! Parece uma má opção (imaginemos as trocas que têm lugar quando todos os elementos são iguais), mas é aquela que conduz a complexidade  $O(N \log N)$ . Porquê?

## 2. Melhoria no cálculo da mediana dos três elementos do array

Ordenar os três elementos no array e inicializar i e j não respectivamente a 0 e high-1, mas 1 e high-3. Porquê?

# Algoritmos de ordenamento

conceitos

Quick Sort – Rearranjo (algumas questões)

## 1. O que fazer quando um elemento é igual ao pivot ?

Deve i parar ou j parar? Se só i parar todos elementos iguais ao pivot vão estar no final no subconjunto à direita do pivot.

**SOLUÇÃO:** param tanto i como j quando encontram elementos iguais ao pivot e trocam os elementos! Parece uma má opção (imaginemos as trocas que têm lugar quando todos os elementos são iguais), mas é aquela que conduz a complexidade  $O(N \log N)$ . Porquê?

## 2. Melhoria no cálculo da mediana dos três elementos do array

Ordenar os três elementos no array e inicializar i e j não respectivamente a 0 e high-1, mas 1 e high-3. Porquê?

Ex.:    8 1 4 9 6 3 5 2 7 0                      ordena os três elementos  
      0 1 4 9 6 3 5 2 7 8                      troca o pivot com a posição high-1  
      0 1 4 9 7 3 5 2 6 8                      inicia o rearranjo  
      0 1 4 9 7 3 5 2 6 8

i=1                      j=high-3

# Algoritmos de ordenamento

conceitos

Quick Sort – Rearranjo (algumas questões)

## 3. Pequenos arrays. Vale a pena usar um algoritmo complexo como é o Quick Sort ?

Quando o número de elementos no array é da ordem de 30 usar por exemplo ordenamento por inserção em vez de Quick Sort.

# Algoritmos de ordenamento

conceitos

Quick Sort – Rearranjo (algumas questões)

**3. Pequenos arrays. Vale a pena usar um algoritmo complexo como é o Quick Sort ?**

Quando o número de elementos no array é da ordem de 30 usar por exemplo ordenamento por inserção em vez de Quick Sort.

De notar que dada a natureza recursiva do Quick Sort vão haver muitos arrays de pequena dimensão a serem ordenados.

# Algoritmos de ordenamento

conceitos

Quick Sort – Rearranjo (algumas questões)

**3. Pequenos arrays. Vale a pena usar um algoritmo complexo como é o Quick Sort ?**

Quando o número de elementos no array é da ordem de 30 usar por exemplo ordenamento por inserção em vez de Quick Sort.

De notar que dada a natureza recursiva do Quick Sort vão haver muitos arrays de pequena dimensão a serem ordenados.

Para além disto esta solução evita-nos casos degenerados como sejam calcular a mediana num array com menos de 3 elementos.

# Algoritmos de ordenamento

implementação

## Quick Sort

```
public static void quicksort( Comparable [] a )
{ quicksort( a, 0, a.length - 1 ); }

private static final int CUTOFF = 30;

public static void swapReferences
( Object [] a, int index1, int index2 )
{ Object tmp = a[ index1 ];
  a[ index1 ] = a[ index2 ];
  a[ index2 ] = tmp; }

private static void quicksort
( Comparable [] a, int low, int high )
{
  if( low + CUTOFF > high )
    insertionSort( a, low, high );
  else
  {
    // Sort low, middle, high
    int middle = ( low + high ) / 2;
    if( a[ middle ].lessThan( a[ low ] ) )
      swapReferences( a, low, middle );
    if( a[ high ].lessThan( a[ low ] ) )
      swapReferences( a, low, high );
    if( a[ high ].lessThan( a[ middle ] ) )
      swapReferences( a, middle, high );

    // Place pivot at position high - 1
    swapReferences( a, middle, high - 1 );
    Comparable pivot = a[ high - 1 ];

    // Begin partitioning
    int i, j;
    for( i = low, j = high - 1; ; )
    {
      while( a[ ++i ].lessThan( pivot ) )
        ;
      while( pivot.lessThan( a[ --j ] ) )
        ;
      if( i < j )
        swapReferences( a, i, j );
      else
        break;
    }

    // Restore pivot
    swapReferences( a, i, high - 1 );

    quicksort( a, low, i - 1 ); // Sort small elements
    quicksort( a, i + 1, high ); // Sort large elements
  }
}
```

# Algoritmos de ordenamento

implementação

## Quick Sort

MELHOR CASO – quando o pivot divide a sequência em duas subsequências do mesmo comprimento

Número de comparações vem aproximadamente (assumindo n tendendo para infinito), vai haver no máximo  $\log N$  “níveis” de profundidade de chamadas recursivas.

$$n + 2 \frac{n}{2} + 4 \frac{n}{4} + 8 \frac{n}{8} + \dots + n \frac{n}{n} = n + n + n + \dots + n = (\log n + 1) n \rightarrow O(n \log n)$$

PRIOR CASO – quando o pivot é o maior ou menor elemento da sequência. Número de comparações vem aproximadamente. Vai haver sempre n “níveis” de profundidade de chamadas recursivas.

$$n + n + \dots + n \rightarrow O(n^2)$$

CASO MÉDIO – para sequências aleatoriamente ordenadas

$$O(n \log n)$$

# Algoritmos de ordenamento

implementação

Quick Sort na Web

<http://www.mpi-sb.mpg.de/~podelski/Teaching/Vorlesung6/quick.html>

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

Algoritmos Elementares

- Inserção
- Selecção
- Troca (bubble sort)

*Estudados no 1º ano*

Algoritmos Eficientes

- Shell
- Rápido (quicksort)
- Mergesort
- Heap

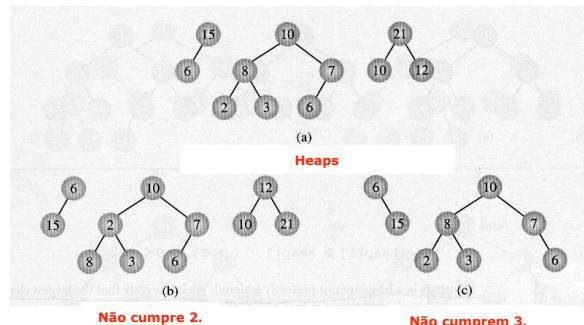
# Algoritmos de ordenamento

conceitos

## HEAPS

### ■ Relembrando...

1. Árvores binárias.
2. Nenhum nó tem valor inferior ao dos seus descendentes (max Heap).
3. A árvore é perfeitamente equilibrada e os nós no último nível ocupam as posições mais à esquerda.

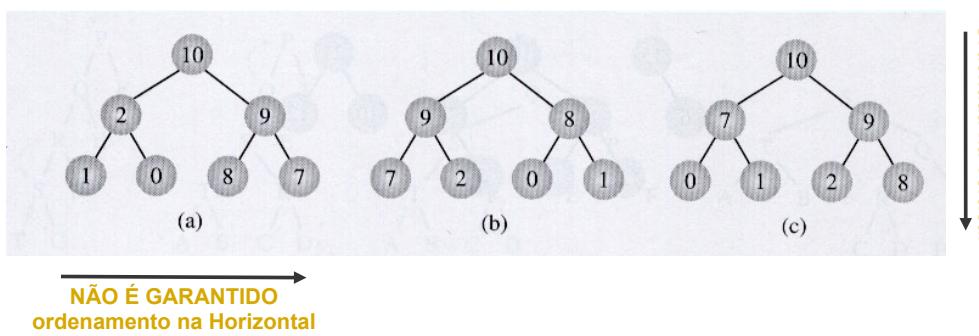


# Algoritmos de ordenamento

conceitos

## HEAPS

### ■ Relembrando...



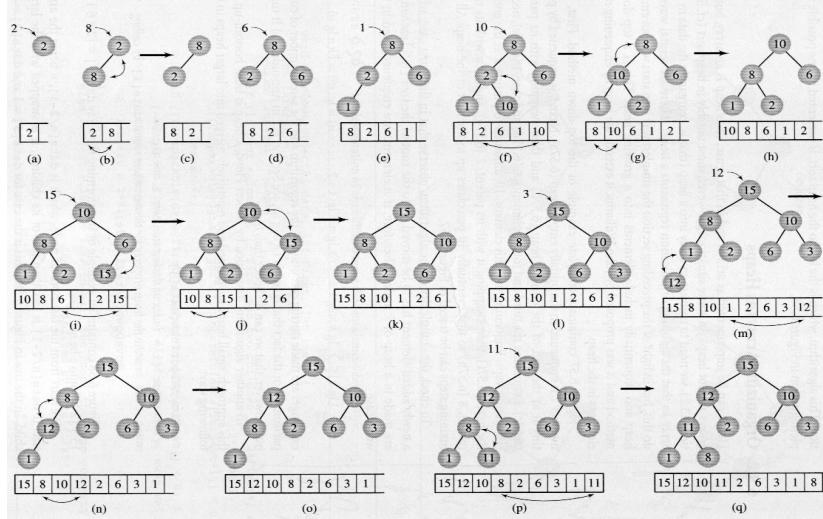
# Algoritmos de ordenamento

conceitos

## HEAPS

### ■ Relembrando...

ARRAYS organizados como HEAPS (abordagem top-down)



Ainda...

❖ o filho esquerdo de um nó está na posição  $2i$

❖ ... o direito na posição  $2i+1$  sendo  $i$  a posição do pai

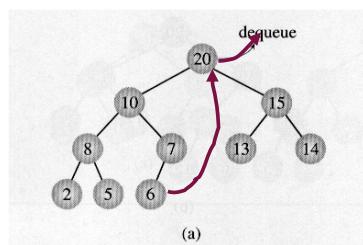
❖ o pai de um nó está na posição  $\lfloor i/2 \rfloor$

# Heaps

conceitos

## HEAPS como LISTAS DE PRIORIDADES

Eliminação numa HEAP definida como LISTA DE PRIORIDADES

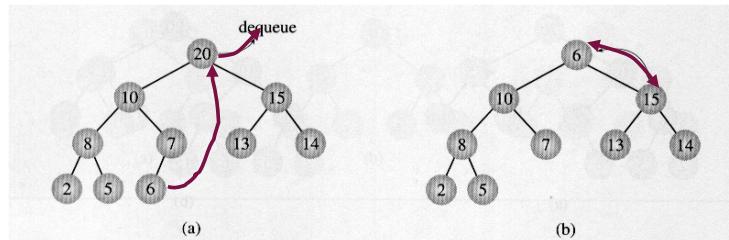


# Heaps

conceitos

## HEAPS como LISTAS DE PRIORIDADES

Eliminação numa HEAP definida como LISTA DE PRIORIDADES

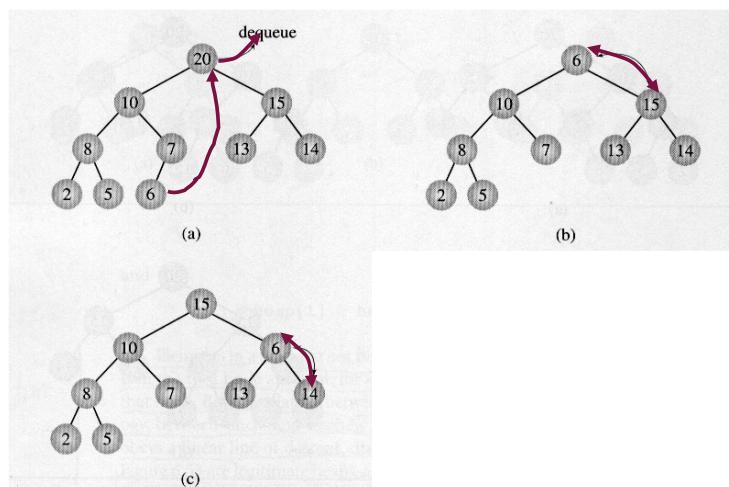


# Heaps

conceitos

## HEAPS como LISTAS DE PRIORIDADES

Eliminação numa HEAP definida como LISTA DE PRIORIDADES

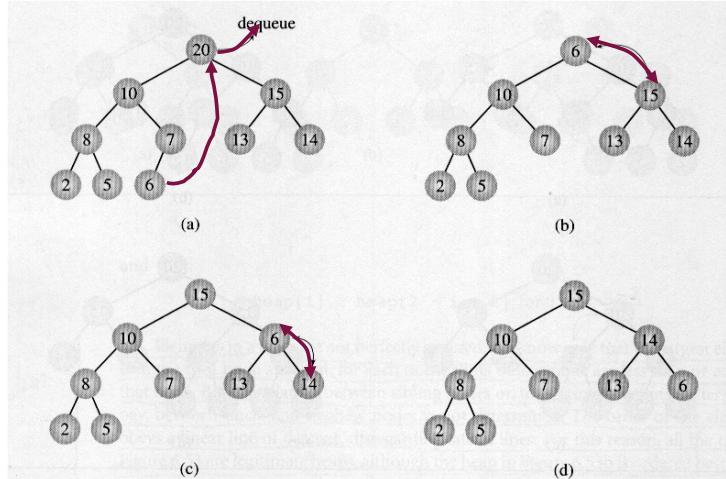


# Heaps

conceitos

## HEAPS como LISTAS DE PRIORIDADES

Eliminação numa HEAP definida como LISTA DE PRIORIDADES



# Algoritmos de ordenamento

conceitos

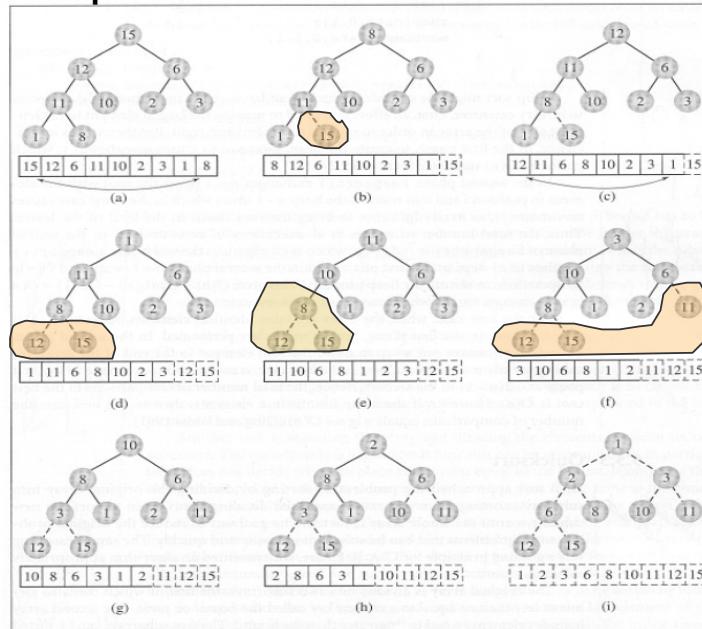
## HEAPSORT

- Numa MaxHeap, o valor que fica na raiz é sempre o maior (ou menor, no caso de uma MinHeap) de todos...
- ...então, e se usarmos o algoritmo da MaxHeap para criar uma Heap e depois tirarmos (dequeue) a raiz sistemáticamente?
  - A sequência de raízes será uma lista ordenada!...
- ...mas onde colocamos os valores das raízes que são tiradas (dequeued)?
  - No final da Heap! Como se estivéssemos a trocar a raiz pela folha mais à direita.

# Algoritmos de ordenamento

conceitos

## HEAPSORT - exemplo



# Algoritmos de ordenamento

Complexidade

## HEAPSORT – Pior caso

- Primeira fase, criação da heap. Número de movimentos: inserção de  $N$  elementos, numa árvore equilibrada.  $O(N \log N)$ . **Nota: existe uma variante que garante  $O(n)$  (algoritmo de Floyd).**
- Segunda fase, extração das raízes e actualizações na heap:
  - A raiz é extraída  $n-1$  vezes (e trocada também pela ultima folha);
  - De cada vez, a árvore tem que ser restruturada (no pior caso,  $\log i$ , sendo  $i$  o numero de nós existentes na heap)
  - O número total de movimentos, no pior caso é então  $\sum_{i=1}^{n-1} \log i$
- Juntando as duas fases, dará  $O(N \log N) + O(N \log N) + n-1$
- ...ou seja, temos uma complexidade de  $O(N \log N)$

# Algoritmos de ordenamento

Complexidade

## HEAPSORT – Melhor caso

- Todos os elementos são iguais.
- Primeira fase:
  - Simples inserção nas folhas. Não há movimentos para cima:  $O(n)$ .
- Segunda fase:
  - $n-1$  movimentos para deslocar a raiz para a folha mais à direita (sem movimentos para baixo):  $O(n)$
- Juntando as duas fases, temos complexidade linear:  $O(n)$

# Algoritmos de ordenamento

implementação

Heap Sort na Web

<http://www.sorting-algorithms.com/>

<http://www.student.seas.gwu.edu/~idsv/idsv.html>

<http://www.bridgeport.edu/~dichter/lilly/heapsorting.htm>

<http://sciris.shu.edu/~borowski/Sorting/SortingDemo.html>

# Algoritmos de ordenamento

conceitos

Métodos de Ordenamento

Algoritmos Elementares

- Inserção
- Selecção
- Troca (bubble sort)

*Estudados no 1º ano*

Algoritmos Eficientes

- Shell
- Rápido (quicksort)
- Mergesort
- Heap

# Algoritmos de ordenamento

... end ;-)

