

MÉTODOS DE ORDENAÇÃO

Introdução à Programação

SI1

Conteúdo

- Conceitos básicos
- Classificação por troca
- Classificação por inserção
- Classificação por seleção

Conceitos Básicos

- **Ordenar:**
 - processo de rearranjar um conjunto de objetos em uma ordem **ascendente** ou **descendente**.
- A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.
 - **Dificuldade de se utilizar um catálogo** telefônico se os nomes das pessoas não estivessem listados em ordem alfabética
- A maioria dos métodos de ordenação é baseada em **comparações dos elementos**

Métodos

- **Métodos simples:**

- Adequados para pequenos arquivos
- Requerem $O(n^2)$ comparações
- Produzem programas pequenos

- **Métodos eficientes:**

- Adequados para arquivos maiores.
- Requerem $O(n \log n)$ comparações.
- Usam menos comparações.
- As comparações são mais complexas nos detalhes

Métodos Simples

- Classificação por **Trocas**
 - Método da Bolha (Bubblesort)
- Classificação por **Inserção**
 - Método da Inserção Direta (Insertionsort)
 - Método dos Incrementos Decrescentes (Shellsort)
- Classificação por **Seleção**
 - Método da Seleção Direta
- Classificação por **Intercalação**
 - Método da Intercalação Simples (MergeSort)

Métodos Eficientes

- Classificação por **Troca**
 - Método de Partição e Troca(Quicksort)
- Classificação por **Seleção**
 - Método de Seleção em Árvores(Heapsort)

CLASSIFICAÇÃO POR TROCAS

Classificação por Trocas

- Processo de classificação que consiste na **comparação** de **pares** de chaves de ordenação, **trocando** os elementos correspondentes se estiverem **fora de ordem**

Classificação por Trocas

- Método da Bolha (**Bubblesort**)
 - Nesse método, o princípio geral da classificação por **trocas** é aplicado a todos os **pares consecutivos** de chaves **não ordenados**
 - Quando não restarem pares **não ordenados**, o vetor estará classificado.

Bubblesort

1. Em cada passo, o **elemento** é comparado com seu **sucessor**.
2. Se o elemento estiver **fora de ordem** a **troca** é realizada.
3. Realizam-se tantos passos quanto forem necessários até que **não ocorram mais trocas**.

Bubblesort – Exemplo

- Vetor inicial **(28 26 30 24 25)**
- Primeira Varredura:
 - (28 26 30 24 25) compara (28,26): **troca**.
 - (26 28 30 24 25) compara (28,30): **não troca**.
 - (26 28 30 24 25) compara (30,24): **troca**.
 - (26 28 24 30 25) compara (30,25): **troca**.
 - (26 28 24 25 **30**) fim da primeira varredura.

Bubblesort - Comentários

- O processo de comparação dos **n-1** pares de chaves é denominado **varredura ou iteração**
- Cada varredura sempre irá posicionar a chave de **maior valor** em sua posição correta, definitiva (**no final do vetor**)
- A cada nova varredura podemos **desconsiderar a última posição** do vetor

Bubblesort – Exemplo

- Vetor inicial **(26 28 24 25 30)**
- **Segunda** Varredura:
 - (26 28 24 25 30) compara (26,28): não troca.
 - (26 28 24 25 30) compara (28,24): troca.
 - (26 24 28 25 30) compara (28,25): troca.
 - (26 24 25 28 30) fim da segunda varredura.

Bubblesort – Exemplo

- Vetor inicial (26 24 25 28 30)
- **Terceira** Varredura:
 - (26 24 25 28 30) compara (26,24): troca.
 - (24 26 25 28 30) compara (26,25): troca.
 - (24 25 26 28 30) fim da terceira varredura.

Bubblesort

Procedimento bubblesort(var lista: vetor [1..n] de inteiro, n:inteiro)

var i, fim, pos: inteiro

troca: logico

chave: inteiro

Inicio

troca=verdadeiro

fim=n-1

pos=1

enquanto troca=verdadeiro faça

troca=falso

para i de 1 ate fim faça

se $v[i] > v[i+1]$ entao

chave = $v[i]$

$v[i]=v[i+1]$

$v[i+1]=$ chave

pos=i

troca=verdadeiro

fimse

fimpara

fim=pos-1

fim enquanto

fim

Bubblesort – Python

```
def bubbleSort(lista,n):  
    troca = True  
    while troca:  
        troca = False  
        for i in range(n-1):  
            if lista[i] > lista[i+1]:  
                chave = lista[i]  
                lista[i] = lista[i+1]  
                lista[i+1] = chave  
                #lista[i],lista[i+1] = lista[i+1],lista[i]  
                troca = True  
    return lista
```

Bubblesort

- A variável **POS** guarda a **posição** onde foi realizada a **última troca** da varredura
- A partir dessa posição, os elementos já se **encontram ordenados** e podem ser **ignorados** na próxima varredura

Bubblesort

Análise

- **Melhor caso** (o vetor já ordenado):
 - Ao final da primeira varredura, o algoritmo verifica que **nenhuma troca** foi realizada e, portanto, o vetor se encontra ordenado
 - Esta primeira e única varredura necessita de **$n-1$ comparações**

Bubblesort

Análise

- **Pior caso** (vetor inversamente ordenado)
 - A cada varredura, **uma chave** será posicionada em seu local definitivo
 - O **total de comparações** necessárias para a ordenação do vetor, nessa caso, será a soma da seguinte progressão aritmética:

$$(n-1)+(n-2)+\dots+2+1 = (n^2-n)/2$$

Bubblesort

Análise

- **Caso médio:**
 - Corresponde a média do desempenho nos casos extremos:
 - $((n-1) + (n^2-n)/2)/2 = (n^2 + n - 2)/4 = O(n^2)$
 - O desempenho médio é da ordem de n^2
 - Este método **não é indicado** para vetores com muito elementos

CLASSIFICAÇÃO POR INSERÇÃO

Métodos Simples

- **Classificação por Trocas**
 - **Método da Bolha (Bubblesort)** 
- **Classificação por Inserção**
 - **Método da Inserção Direta**
 - Método dos Incrementos Decrescentes (Shellsort)
- Classificação por Seleção
 - Método da Seleção Direta
- Classificação por Intercalação
 - Método da Intercalação Simples (MergeSort)

Classificação por Inserção

- Este método consiste em realizar a ordenação pela **inserção** de cada um dos elementos em sua **posição correta**, levando em consideração os elementos **já ordenados**
- Semelhante a organizar **cartas** no baralho

Inserção Direta

- O vetor é dividido em **dois segmentos**: o primeiro contendo os valores **já classificados** e o segundo contendo os elementos ainda **não classificados**
- Inicialmente, o primeiro segmento contém apenas **o primeiro elemento** do vetor e o segundo contém todos os demais elementos

Inserção Direta

- **Método da Inserção Direta**
 1. **Retira-se** o primeiro elemento do vetor não ordenado e coloca-se esse elemento no vetor ordenado na **posição correta**
 2. **Repete-se** o processo até que todos os elementos do vetor não ordenados tenham passado para o vetor ordenado

Inserção Direta - Exemplo

Vetor Inicial (27 12 20 37 19 17 15)

Etapa1: (27 | 12 20 37 19 17 15)

Etapa2: (12 27 | 20 37 19 17 15)

Etapa3: (12 20 27 | 37 19 17 15)

Etapa4: (12 20 27 37 | 19 17 15)

Etapa5: (12 19 20 27 37 | 17 15)

Etapa6: (12 17 19 20 27 37 | 15)

Etapa7: (12 15 17 19 20 27 37)

Inserção Direta

```
procedimento insercao_direta(var v: vetor [1..n] de inteiro, n:inteiro)
  var  i,j: inteiro
        chave: inteiro
início
  para i de 2 até n faça
    chave<--v[i]
    j<-- i - 1
    enquanto (j>0) e (v[j]>chave) faça
      v[j+1]<-- v[j]
      j<-- j - 1
    fim enquanto
    v[j+1]<-- chave
  fim para
fim
```

Inserção Direta Python

```
def insSort(lista):  
    for i in range(1, len(lista)):  
        chave = lista[i]  
        j = i  
        while j > 0 and lista[j - 1] > chave:  
            lista[j] = lista[j - 1]  
            j -= 1  
        lista[j] = chave  
    return lista
```

Inserção Direta

Análise

- **Melhor caso** → o vetor já está ordenado:
 - É necessário pelo menos **uma** comparação para localizar a posição da chave
 - O método efetuará um total de **n-1** iterações para dar o vetor como ordenado.

Inserção Direta

Análise

- **Pior Caso** (vetor inversamente ordenado)
 - Cada elemento a ser inserido será menor que todos os demais já ordenados
 - Todos os elementos terão que ser deslocados uma posição a direita.
 - O **total de comparações** necessárias para a ordenação do vetor, nessa caso, será a soma da seguinte progressão aritmética:

$$(n-1)+(n-2)+\dots+2+1 = (n^2-n)/2$$

Inserção Direta

Análise

- Desempenho Médio (casos normais):
 - Corresponde a média do desempenho nos casos extremos:
 - $((n-1) + (n^2-n)/2)/2 = (n^2 + n - 2)/4 = O(n^2)$
- O desempenho médio é da ordem de n^2 ,
 - é proporcional ao quadrado do número de elementos do vetor
- Método **não indicado** para vetores com muito elementos

Shellsort

- Resultado de trabalho publicado pelo matemático Donald Shell em 1959.
- Ordenação por inserção através **de incrementos**;
- Consiste em passar várias vezes pela lista **dividindo-a em grupos**. Nos grupos menores é aplicado outro método de ordenação (geralmente insertion sort).
- É o algoritmo mais eficiente entre os de baixa complexidade de implementação;

Shellsort

- Algoritmo:
 - Inicialmente a seqüência original é dividida em grupos;
 - Isso pode ser feito dividindo-se o tamanho da seqüência ao meio. O resultado dessa divisão é guardado em uma variável (h , que representa a quantidade de saltos necessários para formar um grupo);
 - Em seguida são aplicadas ordenações (com qualquer outro algoritmo) nos sub-grupos (que são formados saltando-se de “ h em h elementos”);
 - O valor de h vai sendo novamente dividido até que os “saltos” sejam de elemento em elemento;

Shellsort (Simulação)

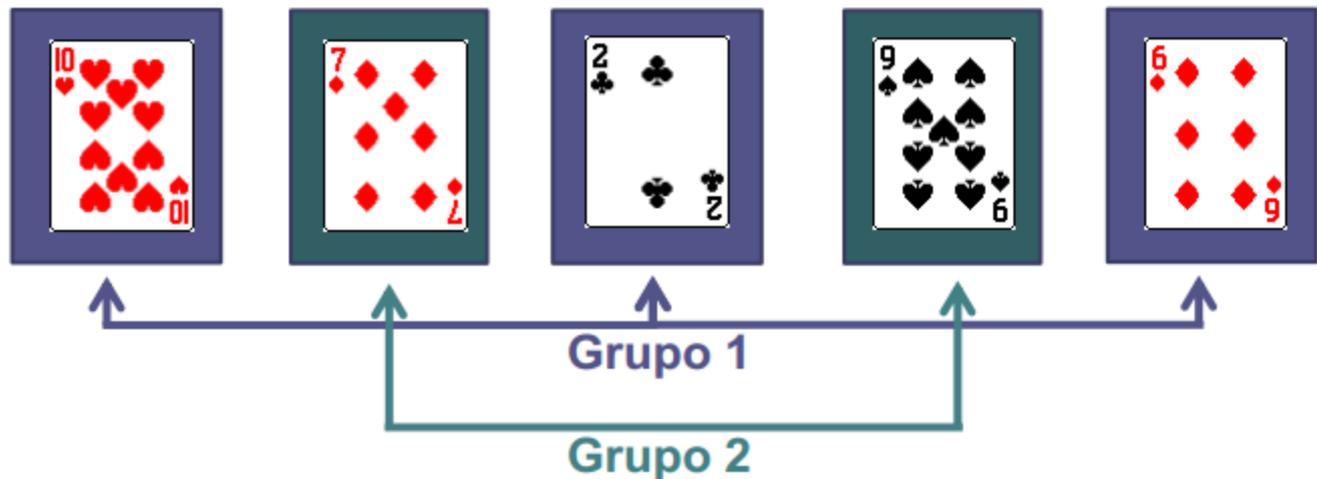


Cálculo do número de saltos (valor inteiro)

$$h = \text{tam} \div 2$$

$$h = 2$$

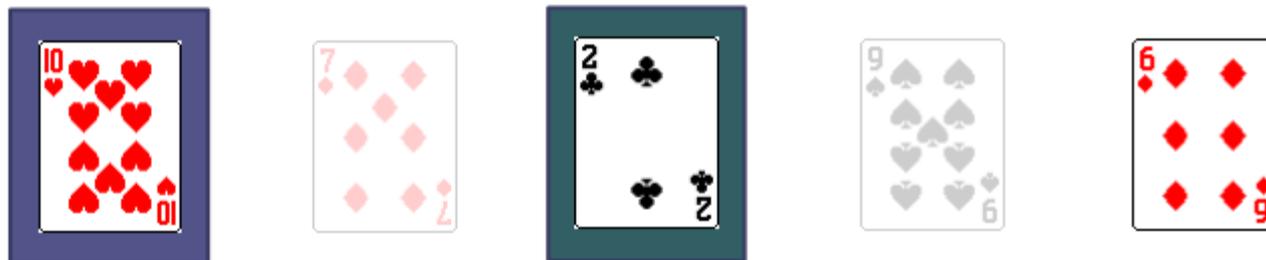
1ª rodada
(elementos
de 2 em 2)



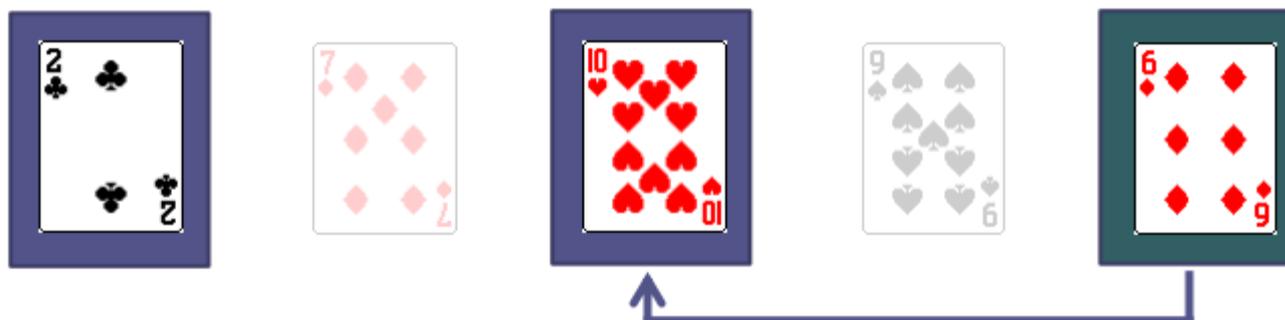
Shellsort (Simulação)

Ordenação
do Grupo 1

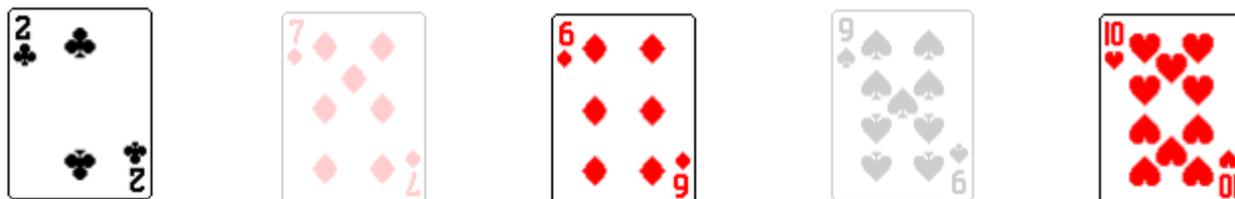
1ª Iteração



2ª Iteração



Ordenação
do grupo 1
concluída



Shellsort (Simulação)

Ordenação
do Grupo 2

1ª Iteração

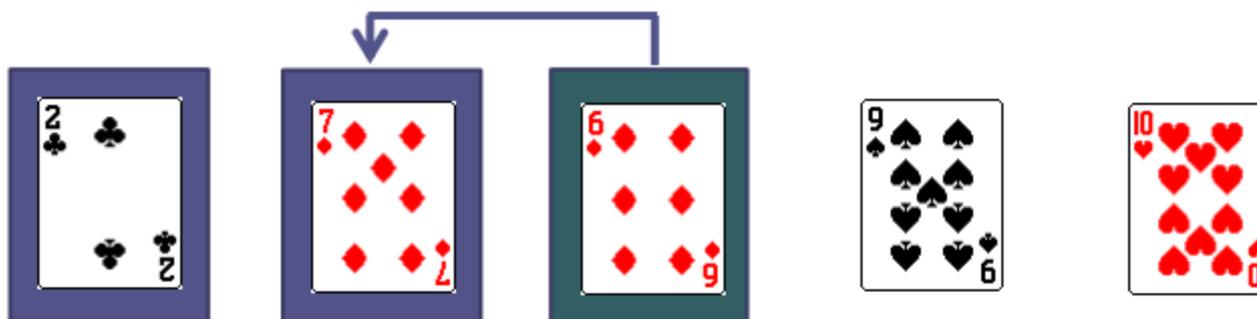


Novo cálculo do número de saltos (valor inteiro)

$$h = h \div 2$$

$$h = 1$$

Observe que a
ordenação da
sequencia
resultante é mínima



Shellsort

- Python

```
def shellSort(nums):  
    n = len(nums)  
    h = int(n / 2)  
    while h > 0:  
        for i in range(h, n):  
            c = nums[i]  
            j = i  
            while j >= h and c < nums[j - h]:  
                nums[j] = nums[j - h]  
                j = j - h  
            nums[j] = c  
        h = int(h / 2.2)
```

CLASSIFICAÇÃO POR SELEÇÃO DIRETA

Métodos Simples

- **Classificação por Trocas**
 - **Método da Bolha (Bubblesort)**
- **Classificação por Inserção**
 - **Método da Inserção Direta**
 - **Método dos Incrementos Decrescentes (Shellsort)**
- **Classificação por Seleção**
 - **Método da Seleção Direta**
- **Classificação por Intercalação**
 - **Método da Intercalação Simples (MergeSort)**

Classificação por Seleção

- Este processo de classificação consiste em uma seleção sucessiva do **menor** ou do **maior** valor contido no vetor, dependendo se a classificação dos elementos será em ordem **crecente** ou **decrecente**

Classificação por Seleção

- **Método:**

- A cada passo, o elemento de menor (ou maior) valor é **selecionado** e colocado em sua **posição correta** dentro do vetor classificado
- Esse processo é **repetido** para o segmento do vetor que contém os elementos ainda não selecionados.

Classificação por Seleção

- **Método:**

- O vetor é dividido em dois segmentos: o **primeiro** contendo os **valores já classificados** e o **segundo** contendo os **elementos ainda não selecionados**
- **Inicialmente**, o primeiro segmento está vazio e o segundo segmento contém todos os elementos do vetor

Classificação por Seleção

- **Algoritmo**

1. É feita uma varredura no segmento que contém os elementos ainda não selecionados, identificando o elemento de **menor** (ou maior) valor
2. O elemento identificado no passo 1 é inserido no segmento classificado na **última posição**
3. O **tamanho do segmento** que contém os elementos ainda não selecionados é atualizado, ou seja, **diminuído de 1**
4. O processo é repetido até que este segmento fique com apenas um elemento, que é o **maior(ou menor)** valor do vetor

Classificação por Seleção – Exemplo

Vetor Inicial (21 27 12 20 37 19 17 15) TAM = 8

Etapa 1: (12 | 27 21 20 37 19 17 15) TAM = 7

Etapa 2: (12 15 | 21 20 37 19 17 27) TAM = 6

Etapa 3: (12 15 17 | 20 37 19 21 27) TAM = 5

Etapa 4: (12 15 17 19 | 37 20 21 27) TAM = 4

Etapa 5: (12 15 17 19 20 | 37 21 27) TAM = 3

Etapa 6: (12 15 17 19 20 21 | 37 27) TAM = 2

Etapa 7: (12 15 17 19 20 21 27 | 37) TAM = 1

Classificação por Seleção

```
procedimento selecao_direta (var v: vetor [1..n] de inteiro; n:inteiro)
var i, j, menor: inteiro
  aux : inteiro
inicio
  para i de 1 até n-1 faça
    menor <-- i
    para j de i+1 ate n faça
      se vetor[j]<vetor[menor] então
        menor<--j
      fim se
    fim para
    aux<--vetor[i]
    vetor[i]<--vetor[menor]
    vetor[menor]<--aux
  fim para
fim
```

Classificação por Seleção – Python

```
def swap(l, a, b):
```

```
    temp = l[a]
```

```
    l[a] = l[b]
```

```
    l[b] = temp
```

```
def selectionSort(lista):
```

```
    n = len(lista)
```

```
    for posPrimDes in range(n-1): # posPrimDes indica o primeiro elmtto não ordenado
```

```
        posMenor = posPrimDes
```

```
        for j in range(posPrimDes+1, n): # for para encontrar o menor elmtto não ordenado
```

```
            if(lista[j] < lista[posMenor]):
```

```
                posMenor = j
```

```
        swap(lista, posPrimDes, posMenor) # troca o 1º elmtto com o menor não ordenado
```

```
    return lista
```

Classificação por Seleção – Análise

- A classificação de um vetor de **n** elementos é feita pela execução de **n-1** passos sucessivos:
 - Em cada passo, determina-se aquele de **menor valor** dentre os elementos ainda **não selecionados**

Classificação por Seleção – Análise

- No **primeiro passo**, são feitas **$n-1$ comparações** para a determinação do **menor valor**
- No **segundo passo**, **$n-2$ comparações**, e assim sucessivamente
- Até que no **último passo** é efetuada apenas **uma comparação**

Classificação por Seleção – Análise

- O número total de comparações é dado por:

$$NC = (n-1) + (n-2) + (n-3) + \dots + 2 + 1$$

- Essa sequência representa a soma de uma progressão aritmética que pode ser generalizada com a seguinte fórmula:

Fórmula da soma dos termos de uma progressão aritmética

$$S_n = (n/2) * (a_1 + a_n)$$

$$S_n = (n/2) * (n - 1 + 1)$$

$$S_n = (n/2) * n$$

$$S_n = n^2/2$$

Classificação por Seleção – Análise

- O desempenho médio do método é da ordem de n^2 **$O(n^2)$** , ou seja, é proporcional ao quadrado do número de elementos do vetor
- Esse método **não é indicado** para vetores com muito elementos

Exercício

- Ordene o vetor a seguir utilizando o método da seleção direta. Escreva passo a passo a execução do algoritmo no vetor.

(15 27 23 45 10 90 80)

Classificação por Intercalação

- **MergeSort**

- Intercala mais de um vetor classificado em um terceiro

- **Passos:**

1. Divida um vetor em n sub-vetores de tamanho um.
2. Intercale pares adjacentes (sub-vetores de tamanho 2).
3. Repita o passo 2 até que o sub-vetor tenha tamanho n .

Mergesort

- Para o vetor de entrada:
 - [25] [57] [38] [37] [12] [92] [86] [33]
- Na passagem 1 temos os sub-vetores:
 - [25 57] [37 38] [12 92] [33 86]
- Na passagem 2 temos os sub-vetores:
 - [25 37 38 57] [12 33 86 92]
- Na passagem 3 temos os sub-vetores:
 - [12 25 33 37 38 57 86 92]

Métodos Eficientes

- Classificação por troca
 - QuickSort
- Classificação por seleção
 - HeapSort (Seleção por árvores)

QuickSort

- O mais eficiente método para o caso médio.
- Também conhecido como classificação por troca de partição:
- Passos:
 - Ache o local correto do primeiro elemento do vetor; $a=x[0]$ colocando à sua esquerda todos os valores menores que ele, e deixando à sua direita os que são maiores.
 - Ordene os dois sub-vetores usando os quicksort até que o subvetor seja de apenas um elemento.

QuickSort - exemplo

- Vetor de entrada: 25 57 48 37 12 92 86 33
 - $A = 25$
- Com o valor de a na posição correta o problema se reduz a classificação de dois sub-vetores
 - (12) 25 (57 48 37 92 86 33)

QuickSort - exemplo

- O sub-vetor (12) já está ordenado
- O outro sub-vetor ou partição (57 48 37 92 86 33) será tratado da mesma forma
 - Novo $a = 57$
- Com o valor do novo a na posição, temos mais dois sub-vetores menores:
 - 12 25 (48 37 33) 57 (92 86)

QuickSort - Exemplo

- As repetições posteriores são:
 - 12 25 (48 37 33) 57 (92 86)
 - 12 25 (37 33) 48 57 (92 86)
 - 12 25 33 37 48 57 (92 86)
 - 12 25 33 37 48 57 (86) 92
 - 12 25 33 37 48 57 86 92
- Conclusão: o quicksort é recursivo!

Conclusão

- Os algoritmos estudados são apenas alguns dos muitos existentes.
- Quase todos os algoritmos podem usar estruturas com alocação dinâmica.
- Na maior parte dos casos os algoritmos de classificação ordenam chaves de estruturas complexas.
- Não existe um algoritmo ideal (melhor) para classificação – tudo depende dos dados.

Bibliografia

- Cormen, Thomas H. et. al. Algoritmos: Teoria e Prática. Editora Campus, 2002.
- Ziviani, Nivio. Projeto de Algoritmos. Editora Nova Fronteira, 2004.
- Complexidade (Prof. Jones Albuquerque)
 - http://www.cin.ufpe.br/~joa/menu_options/school/cursos/ppd/aulas/complexidade.pdf

Applets

- Buble sort:
 - <http://www.cs.oswego.edu/~mohammad/classes/csc241/samples/sort/Sort2-E.html>
- Insert sort
 - <http://www.cs.pitt.edu/~kirk/cs1501/animations/Sort2.html>
- Selection sort
 - <http://www.cs.pitt.edu/~kirk/cs1501/animations/Sort2.html>
- MergeSort
 - <http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/mergeSort/mergeSort.html>