Algoritmos de Ordenação

PROFA. CRISTIANE IMAMURA

Roteiro

- O problema da ordenação
- Onde são aplicáveis?
- Chave da ordenação
- Algoritmos estáveis x instáveis
- Algoritmos de ordenação
  - BogoSort
  - Algoritmo de Inserção
  - BubbleSort
  - SelectionSort
  - QuickSort

O problema da Ordenação

Tentar rearranjar os N elementos de um vetor V, de forma que eles fiquem dispostos da seguinte forma:

- V[0]<= V[1]<=V[2]<=...<=V[N-1]

Onde são aplicáveis

- Lista de alunos ordenadas de acordo com o rendimento escolar;
- Lista telefônica ordenada alfabeticamente pelo nome do cliente;
- Lista de prioridade por tempo de chegada;
- Lista de prioridade por gravidade da enfermidade;
- Entre outras aplicações...

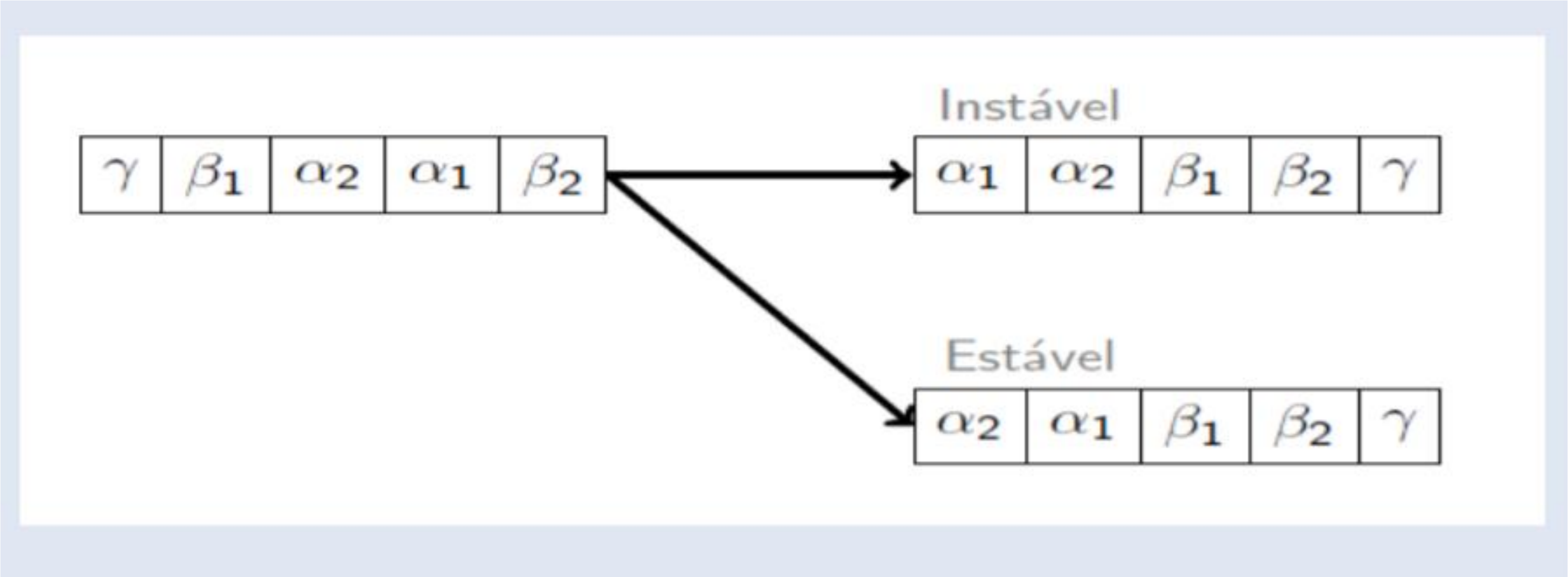
Chave da ordenação

- É o tipo de informação que será usada para compor uma regra bem definida de ordenação.
- Exemplo:
  - Em um vetor de número reais, ordenar de acordo com a parte inteira do número;
  - Em um vetor de registros de alunos, contendo nome e média, usar a média para ordenar.

Algoritmos estáveis x instáveis

Dizemos que um algoritmo de **ordenação é estável** se, e somente se, para todo par de dados *x* e *y*, se antes da ordenação *x* aparece antes de *y* (índice menor) e a chave de *x* for igual à chave de *y*, então após a ordenação *x* continuará antes de *y*.

Algoritmos estáveis x instáveis



Algoritmos de Ordenação

Método Ineficiente:

- Bogosort: O (∞)

Métodos elementares:

- Inserção: $O(n^2)$
- Bubblesort: $O(n^2)$
- Seleção: $O(n^2)$

Métodos Eficientes:

- Quicksort: $O(n \log n)$ (no pior caso é $O(n^2)$)
- Heapsort: $O(n \log n)$
- Mergesort: $O(n \log n)$

BogoSort

Algoritmo é probabilístico que consiste em arranjar aleatoriamente os elementos.

```
while(!em_ordem(lista))
    embaralha(lista);
```

Segue o mesmo principio do teorema do macaco infinito.

Complexidade

- Tempo: $O(\infty)$
  - médio: $O(n \cdot n!)$
- Espaço: $O(n)$

Algoritmo por Inserção

A ideia é separar a lista em duas: a primeira ordenada e a segunda aleatória

1. Para o segundo item em diante, selecione o i-ésimo item.
2. Coloque-o no na parte ordenada, na posição correta.

	D	A	C	B	E
1:	A	D	C	B	E
2:	A	C	D	B	E
3:	A	B	C	D	E
4:	A	B	C	D	E

Algoritmo por Inserção

```
FUNCAO INSERTION_SORT (A[], tamanho)

        VARIAVEIS
        var i, j, elemento;


        PARA j <- 1 ATÉ tamanho - 1 FAÇA
                elemento <- A[j];
                i <- j – 1;

                ENQUANTO ((i >= 0) E (A[i] > elemento)) FAÇA
                        A[i+1] <- A[i]
                        A[i] <- elemento
                        i <- i–1
                FIM_ENQUANTO

        FIM_PARA

FIM
```

Complexidade

- Tempo: $O(n^2)$
- Espaço: $O(n)$

Exemplo do insertion sort

6 5 3 1 8 7 2 4

Analisando o desempenho

- Quando o vetor já se encontra ordenado ascendentemente
  - Menor número de comparações
  - $O(n)$

- Quando o vetor se encontra ordenado descendentemente
  - Maior número de comparações
  - $O(n^2)$

No caso médio e no pior caso tem-se: $O(n^2)$

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

3	2	3	4	2

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

3	**2**	**3**	**4**	**2**

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

3	**2**	**3**	**4**	**2**

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

2	**3**	**3**	**4**	**2**

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

2	**3**	**3**	**4**	**2**

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

2	**3**	**3**	**4**	**2**

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

2	**3**	**3**	**4**	**2**

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

2	**3**	**3**	**2**	**4**

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

Analise a classificação do algoritmo na ordenação do seguinte vetor:

2	3	2	3	4

Classificando o algoritmo de inserção

- O Insertion_Sort é um algoritmo estável?

- Exercício:

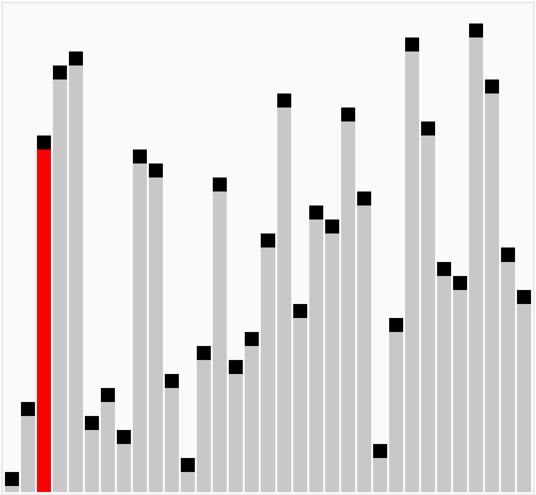
Analise a classificação do algoritmo na ordenação do seguinte vetor:

2	2	3	3	4

BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

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



BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

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

BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

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

BubleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

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

BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

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

BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada “empurrar” o maior elemento para o final.
- Exemplo:

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

BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

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

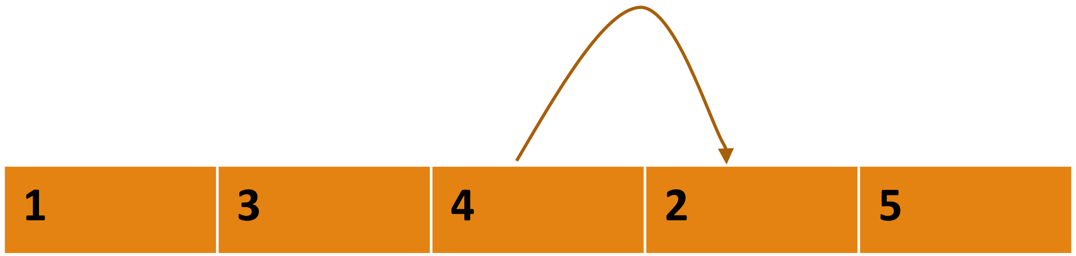
BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

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

BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:



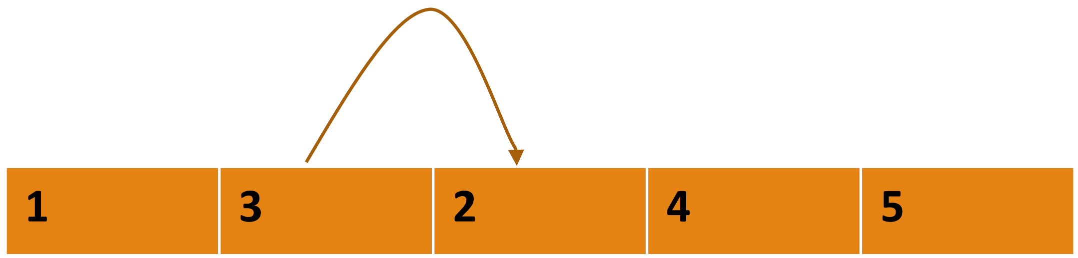
BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

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

BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada “empurrar” o maior elemento para o final.
- Exemplo:



BubbleSort

- Consiste em percorrer o vetor N-1 vezes, e a cada passada "empurrar" o maior elemento para o final.
- Exemplo:

2	**3**	**4**	**4**	**4**

Algoritmo BubbleSort

```
FUNCAO BUBBLE_SORT(V[], tamanho)
VARIAVEIS
var i, j, trocou<-1;
i<- tamanho-1;
ENQUANTO i>= 1 E trocou=1 FAÇA
    trocou<-0;
   PARA j<-0 até i-1 FAÇA
      SE(v[ j ] > v [ j + 1]) ENTÃO
         troca(V[j], V[j + 1]);
          trocou<-1;
    FIM_SE
   FIM_PARA
  i<-i-1;
FIM_ENQUANTO
FIM
```

Algoritmo alterado em aula para responder o exercício do próximo slide

Complexidade

- Tempo: $O(n^2)$
- Espaço: $O(n)$

Exercício

Altere o algoritmo BubbleSort para torna-lo um pouco mais eficiente, de forma a parar a varredura, se não houver trocas.

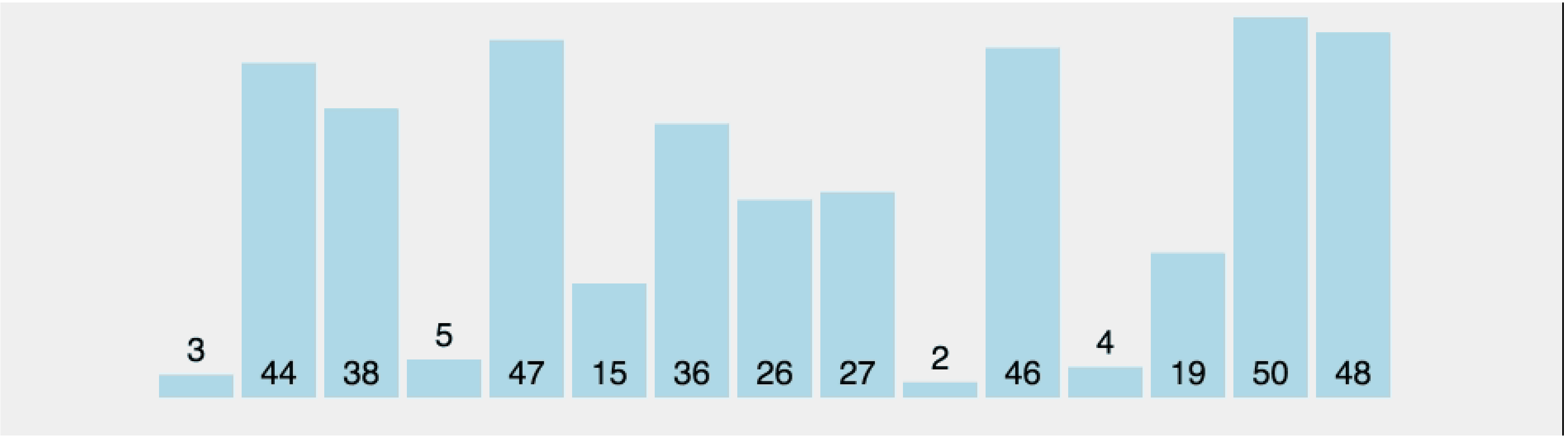
Exercício

O BubbleSort é instável ou estável?

Mostre a execução do algoritmo que justifique a sua resposta.

Algoritmo por Seleção

Consiste em selecionar o menor elemento do vetor e posicioná-lo na posição correta.



Selection_Sort

Exemplo:

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

i

Achar o menor elemento de todo o vetor

Selection_Sort

Exemplo:

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

i

Trocar pelo i-ésimo elemento

Selection_Sort

Exemplo:

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

i

Avançar o i

Selection_Sort

Exemplo:

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

i

Achar o menor elemento a partir da posição i

Selection_Sort

Exemplo:

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

i

Trocar pelo i-ésimo elemento

Selection_Sort

Exemplo:

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

i

Avançar o i

Selection_Sort

Exemplo:

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

i

Encontrar o menor elemento a partir de i

Selection_Sort

Exemplo:

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

i

Avançar o i

Selection_Sort

Exemplo:

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

i

Achar o menor elemento a partir de i

Selection_Sort

Exemplo:

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

i

Trocar o menor elemento pelo i-ésimo

Algoritmo

```
FUNCAO SELECTION_SORT(V[], tamanho)
VARIAVEIS
var i, j, min;
PARA i<-0 até tamanho-2 FAÇA
min <- i;
   PARA j<-i+1 até tamanho-1 FAÇA
      SE(V[ j ] <= V [ min]) ENTÃO
       min <-j;
     FIM_SE
    FIM_PARA
    troca(V[i], V[min]);
FIM_PARA
FIM
```

Complexidade

- Tempo:
  - comparações: $O(n^2)$
  - movimentações $O(n)$
- Espaço: $O(n)$

Características do SelectionSort

- Possui melhor desempenho quando o vetor é menor;
- O desempenho não melhora se o vetor já estiver ordenado;
- Não é um algoritmo estável.

QuickSort

- Proposto em 1960 por Charles Hoare.
- É um dos algoritmos mais conhecidos.
- Possui um excelente desempenho na maioria dos casos O(n log n).
- Não é estável.

A ideia do QuickSort

- Dividir para conquistar
  - Passo 1- Divida o conjunto original a ser ordenado em dois conjuntos menores;
    - Eleja um pivô
    - Mantenha a esquerda do pivô todos os elementos menores e à direita todos os elementos maiores que o pivô.
  - Passo2 - Ordene cada um dos conjuntos menores individualmente (pode envolver a execução do algoritmo novamente para esse passo)
  - Passo 3 – Combine os resultados do passo 2 para produzir a solução final.

Exemplo de Execução

6 5 3 1 8 7 2 4

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

1	5	4	10	3	2	9	7	8	6

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

1	5	4	10	3	2	9	7	8	6

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

1	5	4	10	3	2	9	7	8	6

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

1	5	4	10	3	2	9	7	8	6

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

1	5	4	10	3	2	9	7	8	6

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

1	5	4	6	3	2	9	7	8	10

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

1	**5**	**4**	**6**	**3**	**2**	**9**	**7**	**8**	**10**

O processo de comparações continua para garantir que todos a esquerda do pivô sejam menores que o pivô e todos à direita sejam maiores.

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

1	**5**	**4**	**6**	**3**	**2**	**9**	**7**	**8**	**10**

Todos os n-1 elementos farão parte de uma única partição. Se para esta for selecionando o maior ou o menor elemento como pivô, o problema se repetirá.

A escolha do Pivô

O algoritmo QuickSort depende da escolha do Pivô, uma vez que ele determinará as partições a serem ordenadas e rearranjadas.

O que acontece se o pivô escolhido for sempre o menor elemento da lista, ou o maior elemento da lista?

- Nesses casos haverá um particionamento no qual apenas o pivô fará parte de uma das partições e todo o restante dos elementos precisa ser processado novamente.
- Nesse caso o algoritmo terá o Pior desempenho: O (n^2)

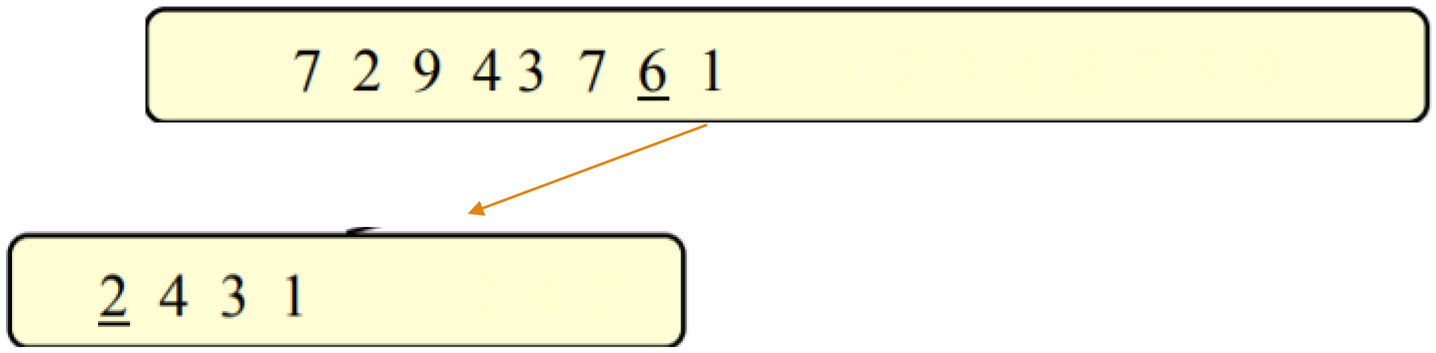
A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.

7 2 9 4 3 7 <u>6</u> 1   1 2 3 4 6 7 8 9

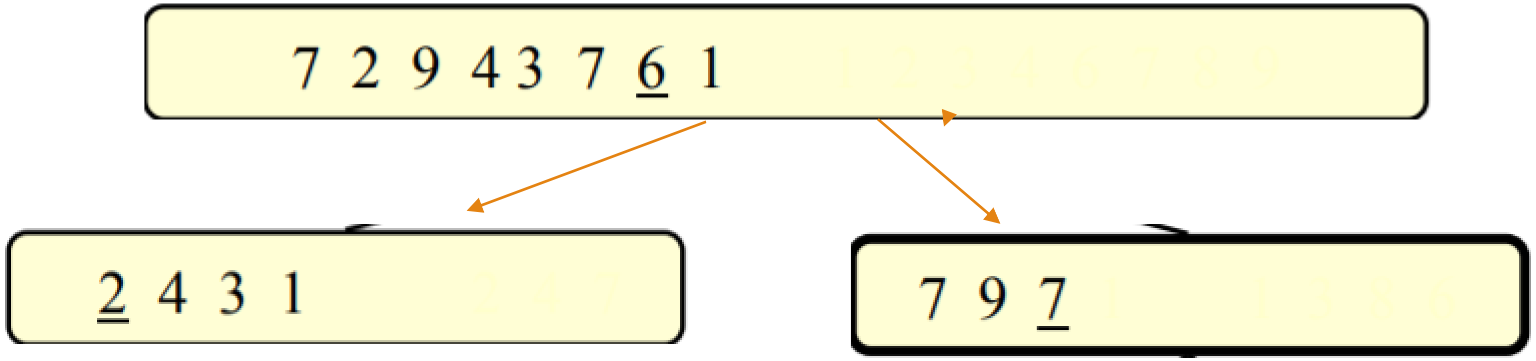
A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.



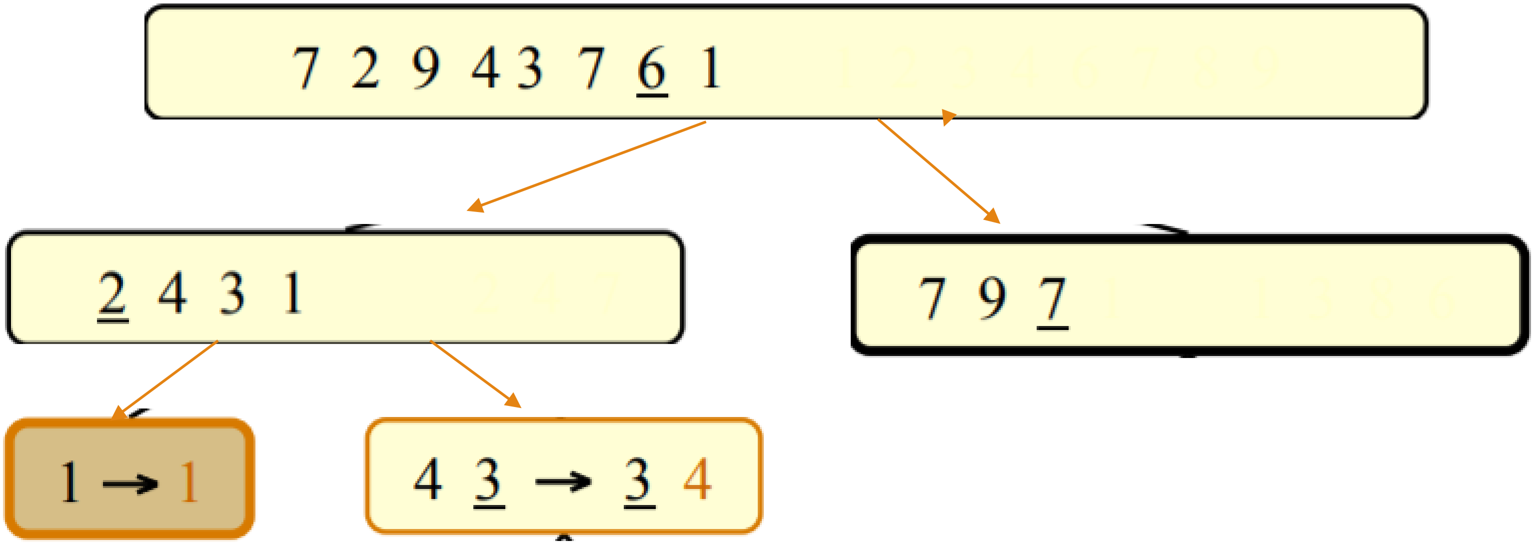
A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.



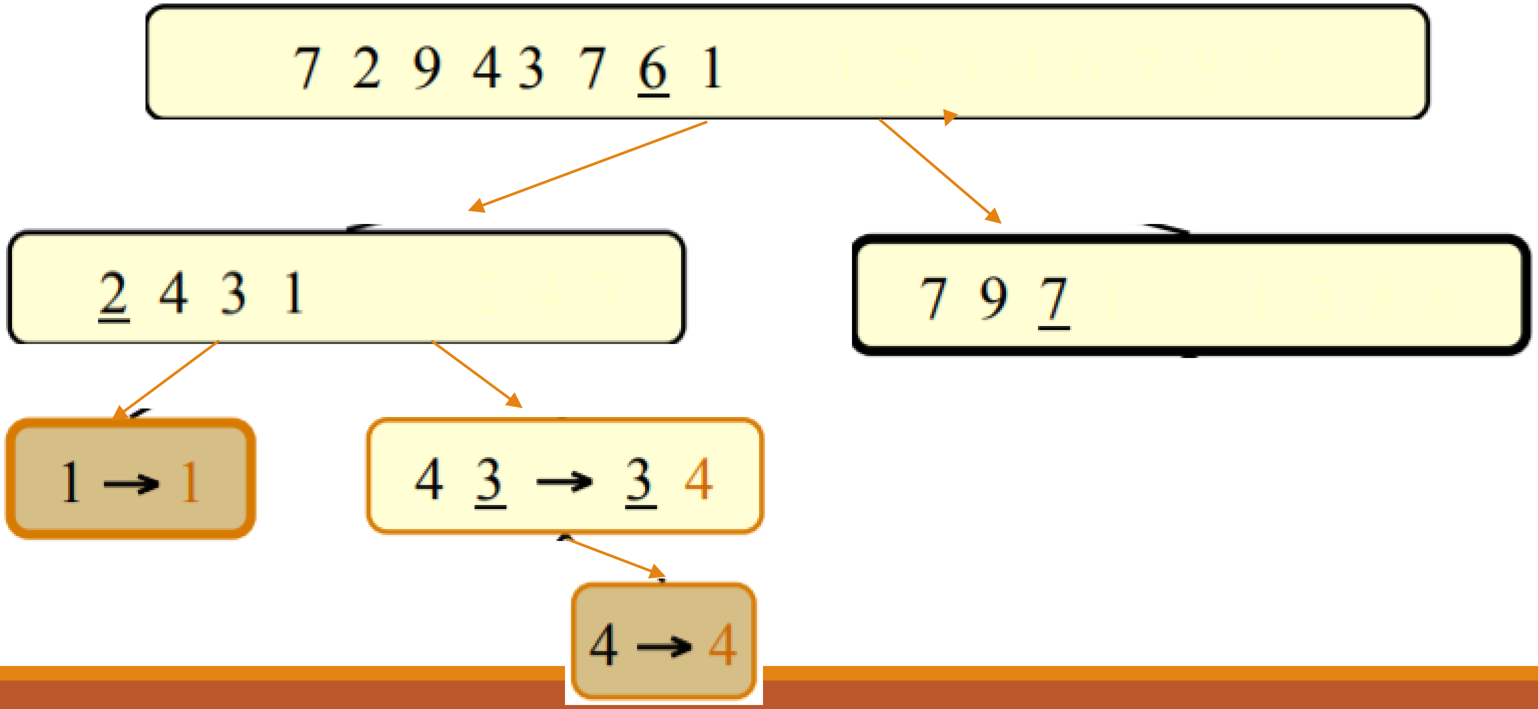
A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.



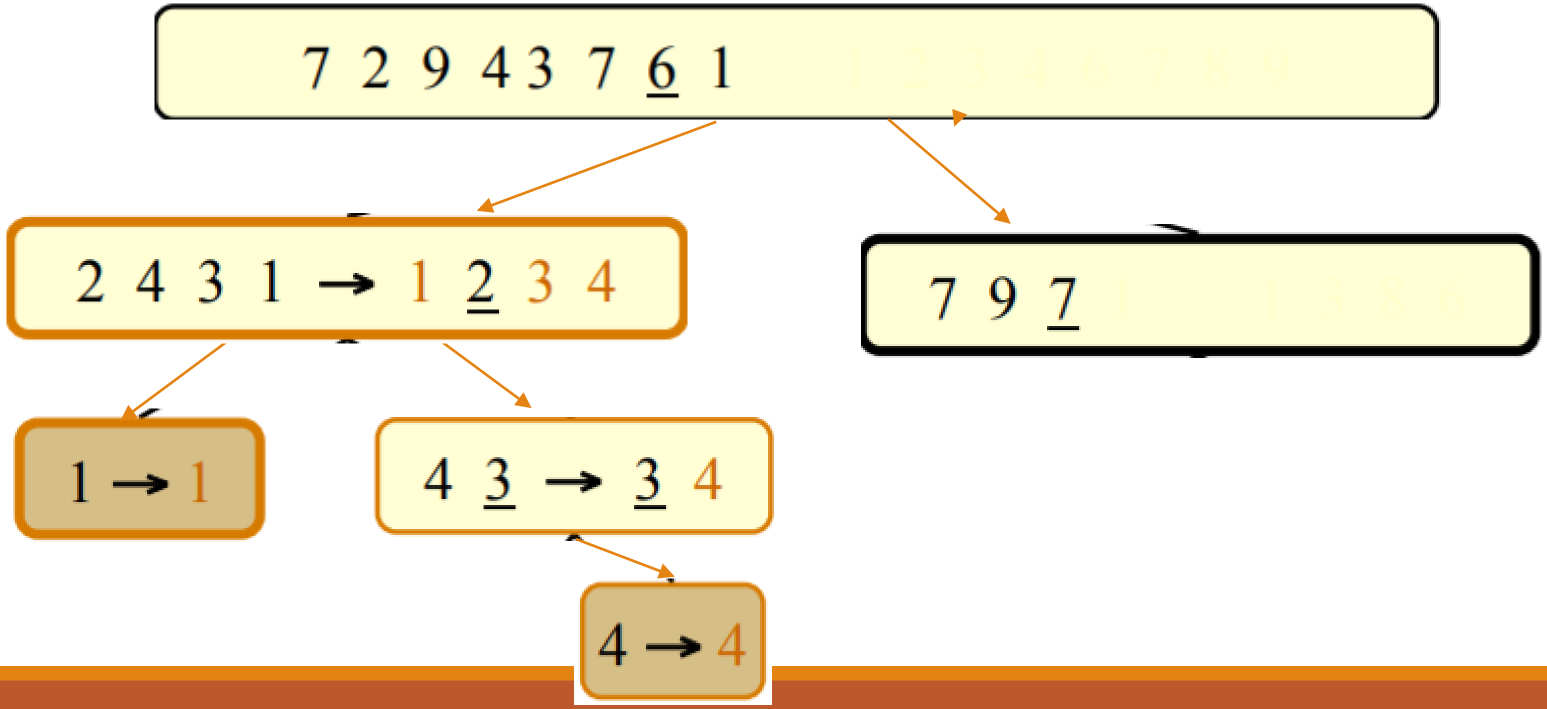
A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.



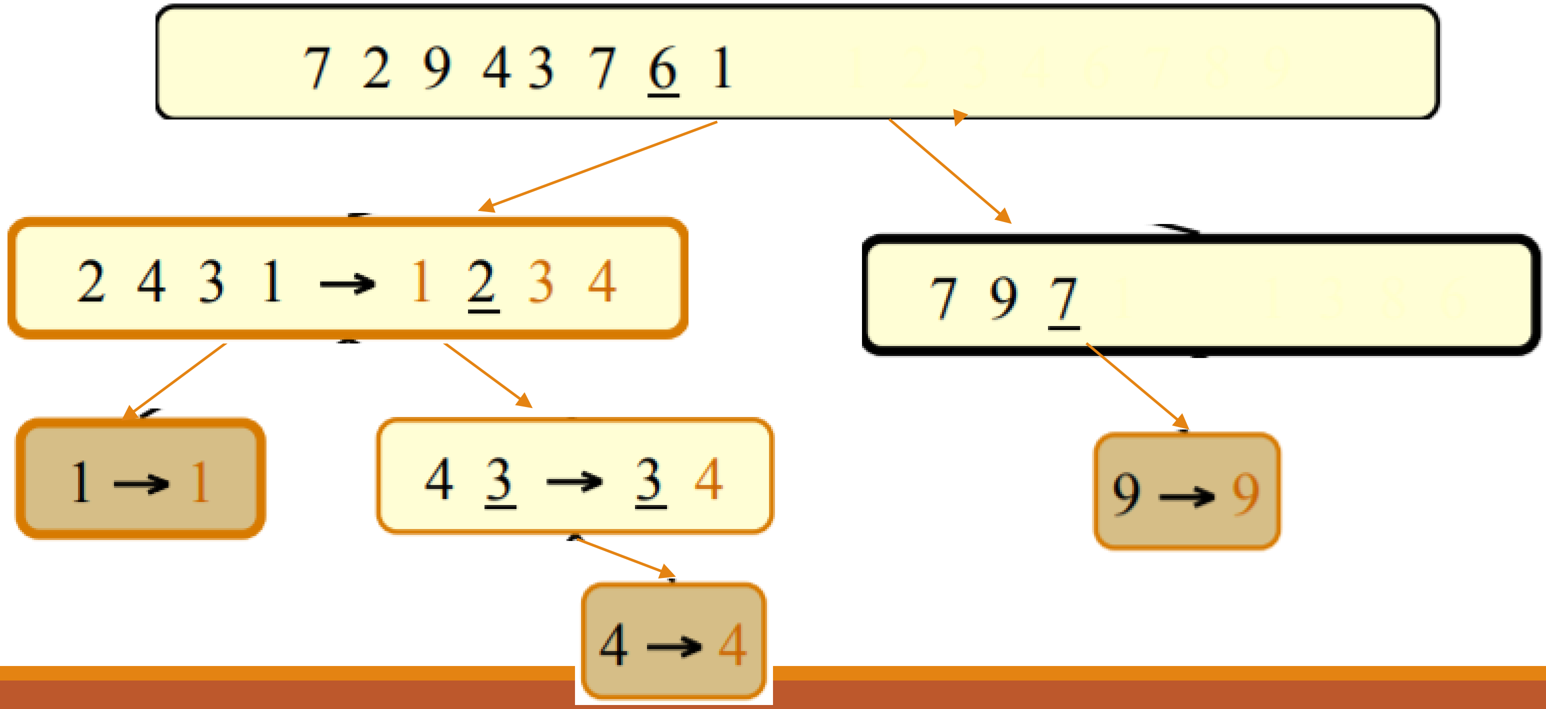
A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.



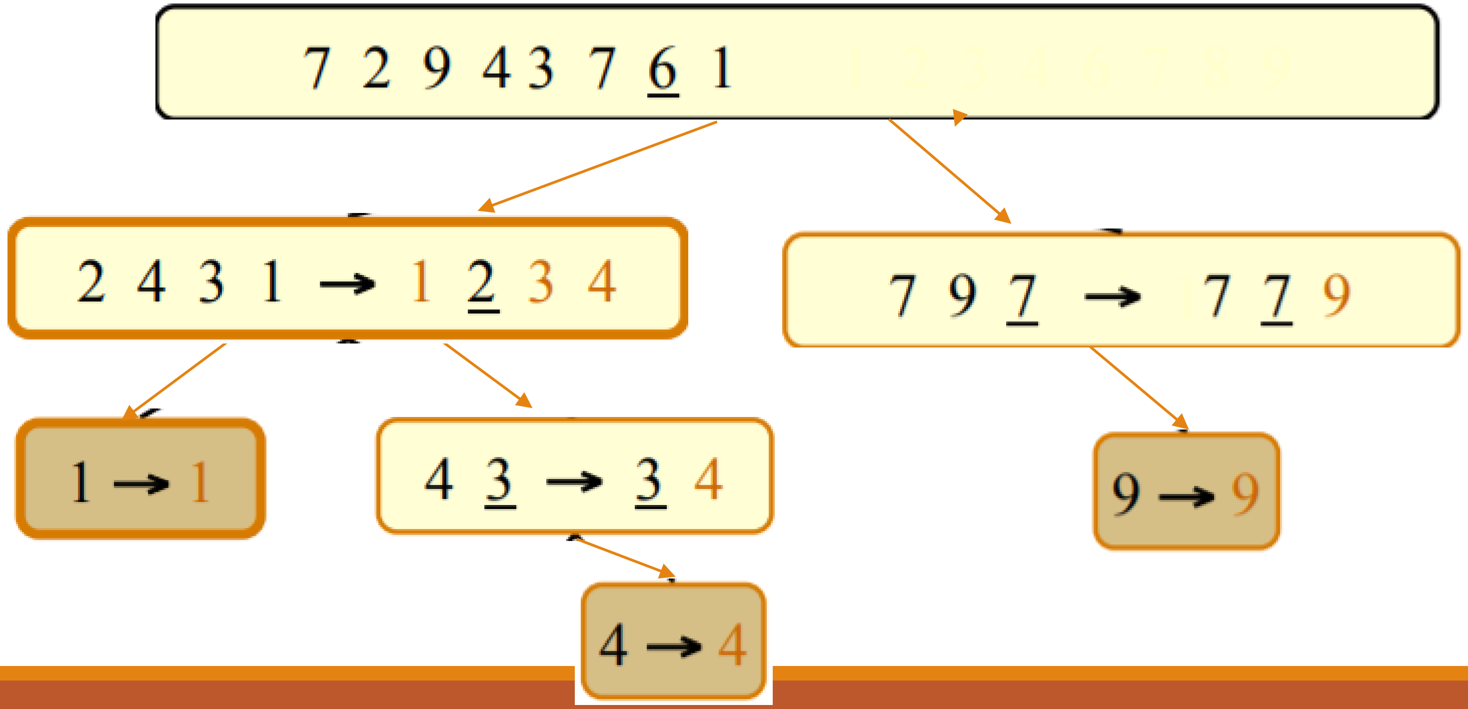
A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.



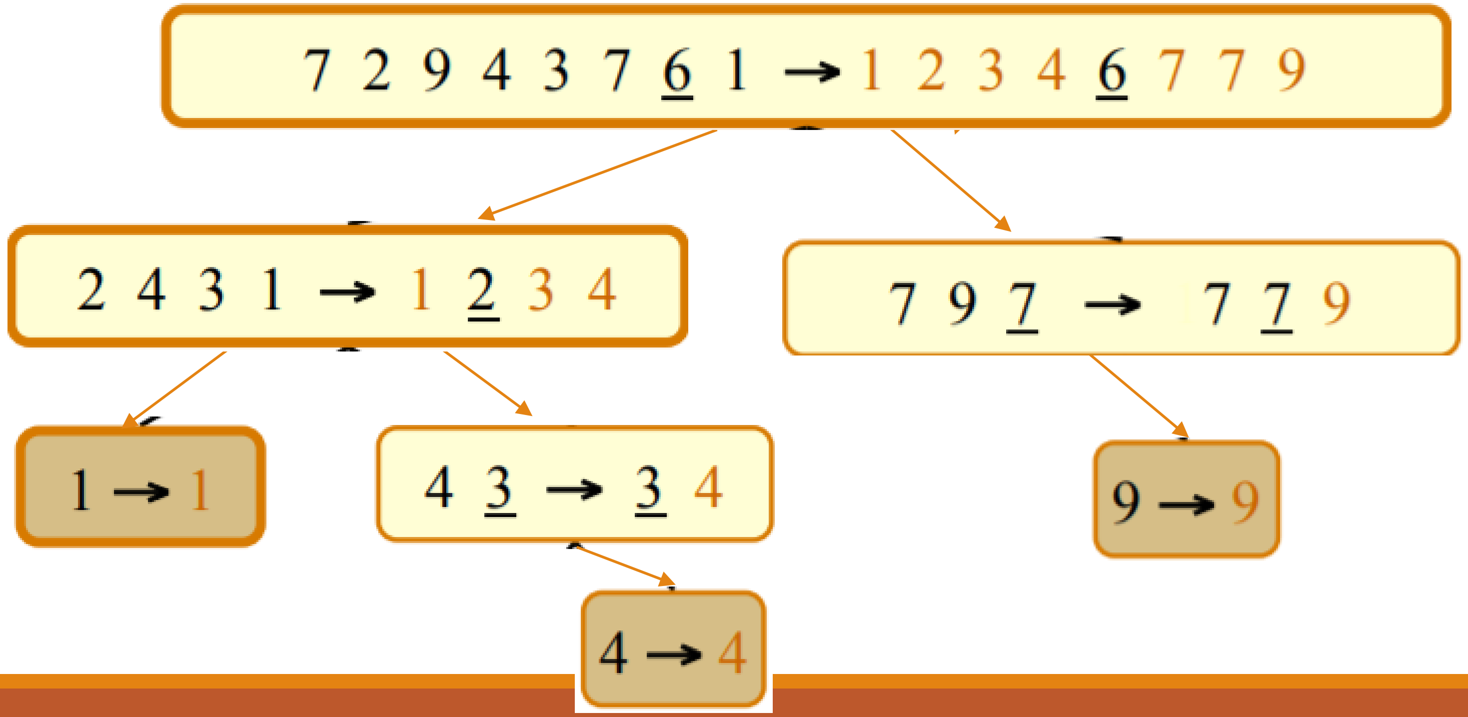
A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.



A escolha do Pivô

No caso médio e melhor caso, o QuickSort, tem tempo de O(nlogn), em função de poder se aproximar de uma estrutura em árvore binária.



Uma proposta de uma algoritmo de Partição

```
typedef int itemType;
#define key(A) (A)
#define less(A, B) (key(A) < key(B))
#define exch(A, B) { itemType t = A; A = B; B = t; }

// Recebe vetor a[p..r] com p < r. Rearranja os elementos
// do vetor e devolve i em p..r tal que
// a[p..i-1] <= a[i] <= a[i+1..r].
//
int partition(itemType a[], int p, int r)
{
    int i = p-1, j = r;
    itemType v = a[r];

    for (;;) {
        while (less(a[++i], v)) ;
        while (less(v, a[--j]))
            if (/*X*/ j == p) break;
        if (i >= j) break;
        exch(a[i], a[j]);
    }
    exch(a[i], a[r]);
    return i;
}
```

P é o início da partição;
r é o fim.
v é o valor do pivô inicialmente

Algoritmo QuickSort

```
// A função rearranja o vetor a[p..r], com p <= r+1,
// de modo que ele fique em ordem crescente.
//
void quicksort(itemType a[], int p, int r) {
    int i;
    if (p < r) {
       i = partition(a, p, r);
       quicksort(a, p, i-1);
       quicksort(a, i+1, r);
    }
}
```

Materiais de apoio a construção desse Material

- Aulas de **MAC0122 Princípios de Desenvolvimento de Algoritmos – IME-USP – Professor** *Paulo Feofiloff*
- *Aulas de* Algoritmos e Estruturas de Dados I – Decom -Universidade Federal de Ouro Preto - Professor Túlio Toffolo
- Aulas de Estruturas de Dados – Departamento de Ciência da Computação – Universidade de Brasília - Professor Eduardo Alchieri