Estrutura de Dados II

Métodos de Ordenação
Parte 3

Profª Márcio Bueno
ed2tarde@marciobueno.com / ed2noite@marciobueno.com

Material baseado nos materiais
do Prof. Robson Lins

Classificação em Memória Primária

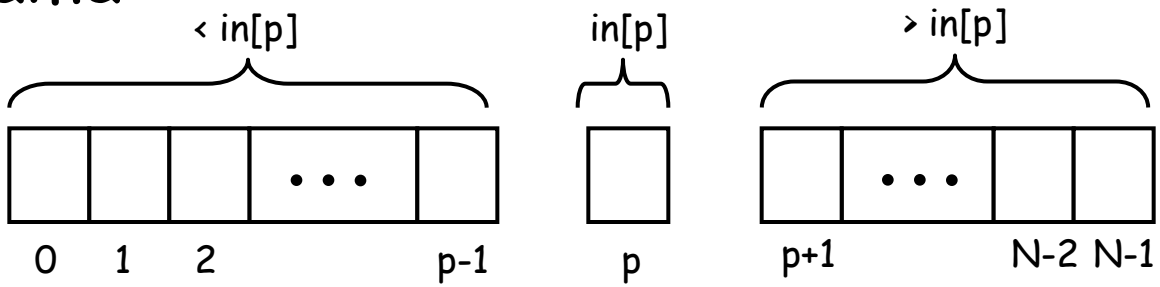
- **Métodos Eficientes (Sofisticados)**
  - ✓Classificação por Troca
    - Método de Partição e Troca - Quicksort
  - ✓Classificação por Seleção
    - Método de Seleção em Árvore - Heapsort

Classificação por Troca - Quicksort

- Proposto por Hoare e tem como base dois princípios:
  - ✓Recursividade
  - ✓Abordagem **dividir para conquistar**
- Consiste em:
  - ✓ Dado um conjunto C de elementos a ser ordenado
  - ✓ Escolher qualquer elemento k do conjunto C para dividir o conjunto em elementos pequenos e grandes. Este elemento é chamado de pivô.
  - ✓ Dividir C em dois sub-conjuntos C1 e C2, onde os elementos de C1 são menores que o pivô e os elementos de C2 são maiores que o pivô.
  - ✓ Repete-se o processo em C1 e C2

Classificação por Troca - Quicksort

- Diagrama:



- Idealmente, o pivô deveria ser selecionado de modo que aproximadamente metade dos elementos ficassem a esquerda do pivô e a outra metade do lado direito do pivô.
- Considere o caso onde o menor ou o maior elemento é escolhido como pivô. Nesse caso um conjunto (C1) ficaria vazio e o outro (C2) com n-1 elementos.

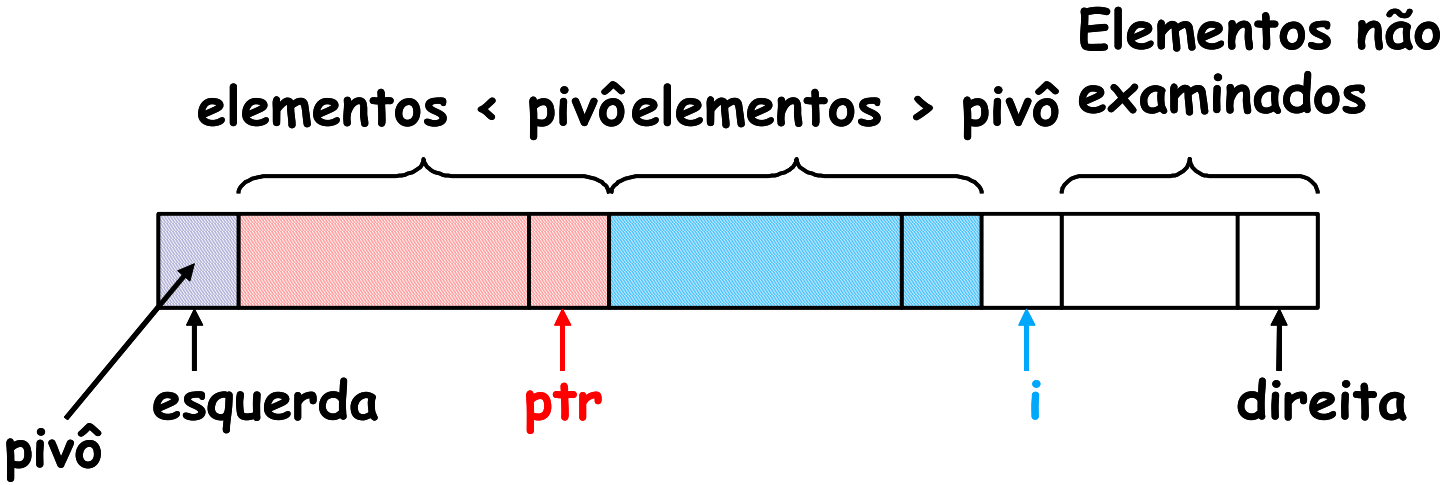
Classificação por Troca - Quicksort

- Escolha do Pivô → Depende da implementação
  - ✓ Usar o primeiro elemento
    - Aceitável se a entrada é aleatória
    - Se a entrada for pré-ordenada ou estiver na ordem inversa é uma péssima escolha
  - ✓ Escolha aleatória
    - Estratégia segura de um modo geral
    - A geração de números aleatórios não implica em um desempenho melhor no restante do algoritmo
  - ✓ Média das entradas
    - Difícil de calcular
    - Prejudicaria o desempenho
    - Pegar 3 elementos e calcular a média

Classificação por Troca - Quicksort

- Escolha do Pivô
  - ✓Por questões de simplicidade, vamos escolher a chave que se encontra na posição inicial do vetor para ser a particionadora.

Classificação por Troca - Quicksort



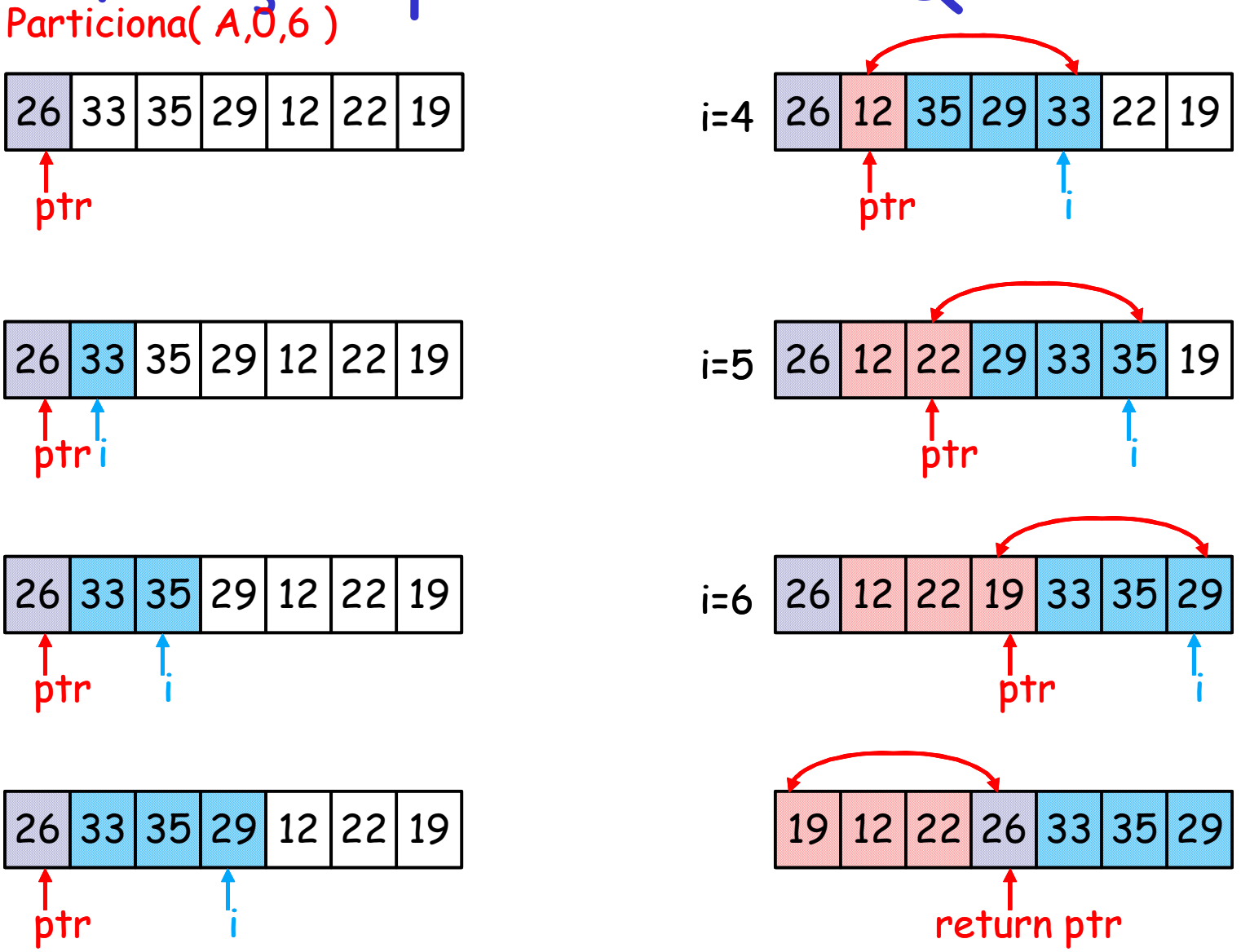
Classificação por Troca - Quicksort

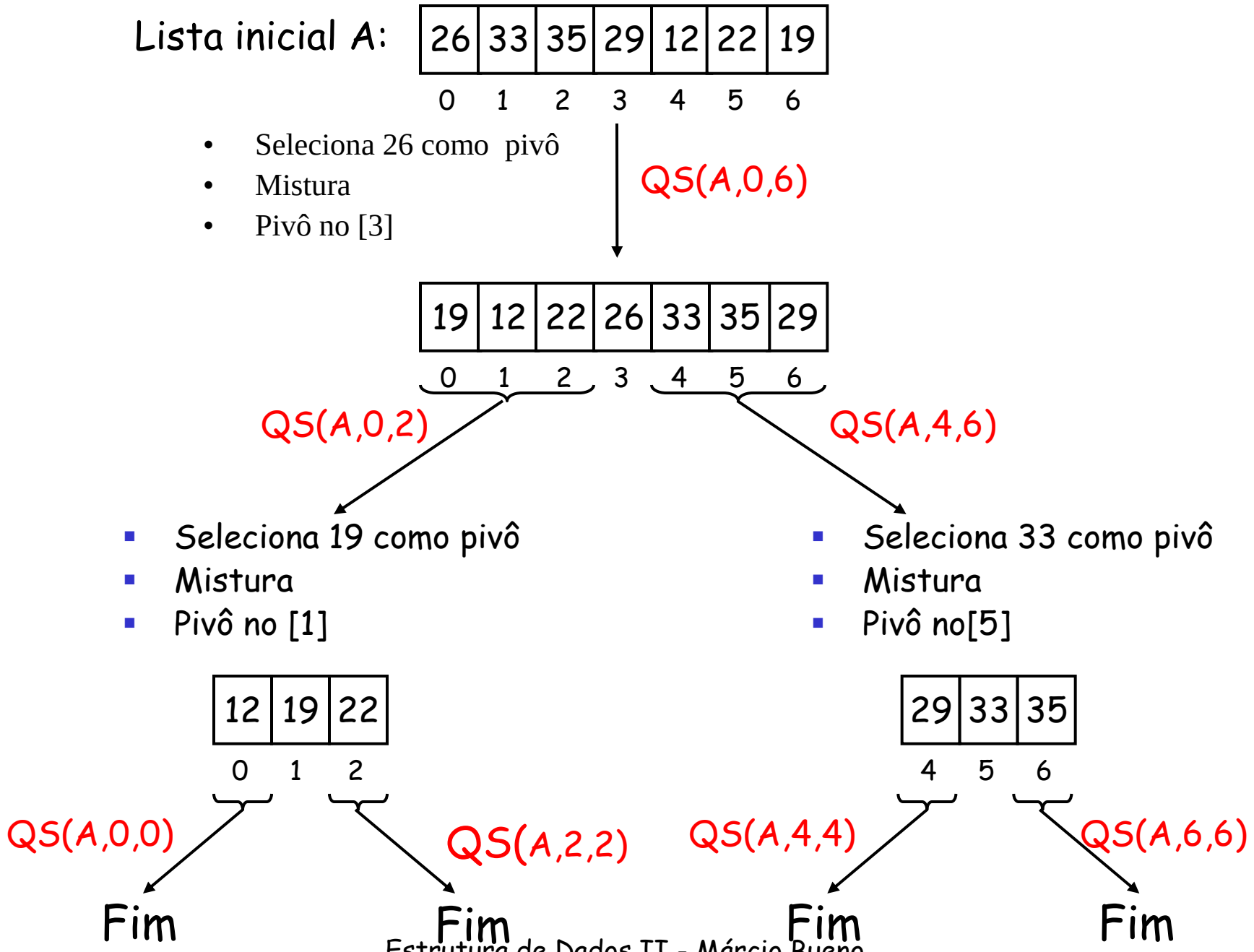
```
int particiona( int A[ ], int esquerda, int direita )
{
    int i, temp;
    int ptr = esquerda;
    int pivo = A[esquerda];     /* pivô é primeiro elemento */

    /* Separa o vetor em elementos pequenos e grandes – em relação ao
            pivô */
   for (i = esquerda+1; i<=direita; i++)
    if( A[i] <= pivo ){
            ptr = ptr + 1;
            troca ( &(A[i]),  &(A[ptr]) );
     }
   /* coloca o pivo entre os elementos pequenos e grandes */
   troca ( &(A[esquerda]),  &(A[ptr]) );
   return ptr;
}
```

Classificação por Troca - Quicksort

```
void troca(int *a, int *b){
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

Classificação por Troca - Quicksort




Lista inicial A:
26 33 35 29 12 22 19
0 1 2 3 4 5 6
Seleciona 26 como pivô
Mistura
Pivô no [3]
QS(A,0,6)
19 12 22 26 33 35 29
0 1 2 3 4 5 6
QS(A,0,2)
QS(A,4,6)
Seleciona 19 como pivô
Mistura
Pivô no [1]
Seleciona 33 como pivô
Mistura
Pivô no[5]
12 19 22
0 1 2
29 33 35
4 5 6
QS(A,0,0)
QS(A,2,2)
QS(A,4,4)
QS(A,6,6)
Fim
Fim
Fim
Fim

Classificação por Troca - Quicksort

```
void quickSort( int A[ ], int esquerda, int direita )
{
    int pivo = particiona (A, esquerda, direita);
    if  ( pivo > esquerda)
      quickSort( A,  esquerda, pivo - 1 );
    if  ( pivo  < direita)
      quickSort( A,  pivo + 1, direita );
}
```

Classificação por Troca - Quicksort

- Comparado com os demais métodos é o que apresenta, em média, o menor tempo de classificação.
- Tem um desempenho logarítmico - **$O(n \log_2 n)$**
  - ✓ É o que apresenta o menor número de operações elementares.
- As razões porque quicksort é mais eficiente
  - ✓ É que o problema de divisão é executado no mesmo vetor e muito eficientemente
  - ✓ Não precisa copiar o vetor ordenado
- Para vetores muito pequenos (N<=20) quicksort não é tão eficiente quanto o algoritmo de inserção, pois como ele é recursivo seria chamado várias vezes diminuindo sua eficiência.
- Não é estável!

Classificação por Troca - QuickSort

- Exercícios
  - ✓ Mostre o passo a passo de executar quicksort para o seguinte conjunto de dados {9, 25, 10, 18, 5, 7, 15, 3}, usando o primeiro elemento como pivô.
  - ✓ Faça o mesmo para esse conjunto de dados {8, 5, 4, 7, 6, 1, 6, 3, 8, 12, 10}.

Classificação por Seleção - Heapsort

- **O Método da Seleção em Árvore – Heapsort**
  - ✓ Inventado por John Williams (1964) e usa a abordagem inerente à ordenação por seleção.
  - ✓ O método utiliza a seleção em árvore para a obtenção dos elementos do vetor na ordem desejada.
  - ✓ Ele consiste em duas fases distintas:
    1) Primeiro é montada uma árvore binária (heap) contendo todos os elementos do vetor, de tal forma que o valor contido em qualquer nó seja maior do que os valores de seus filhos;
    2) Em seguida, o heap é usado para a seleção dos elementos na ordem desejada.

Classificação por Seleção - Heapsort

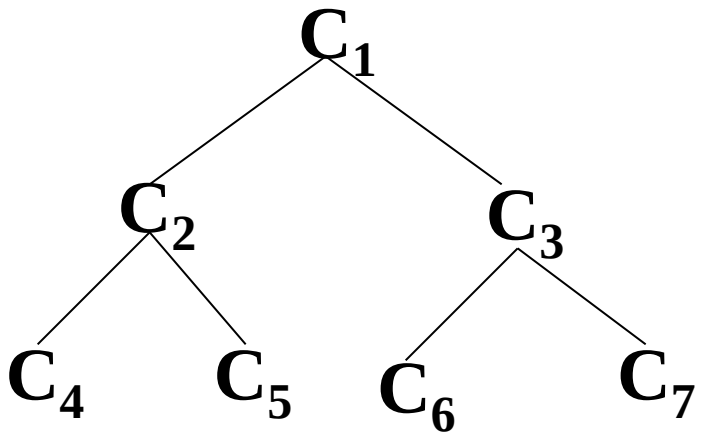
- **Método Heapsort - Descrição do Algoritmo**
  - ✓ Dado um vetor de chaves $C_1, C_2, \ldots, C_N$, consideramos este vetor como sendo a representação de uma árvore binária, usando a seguinte interpretação dos índices das chaves:

$$\begin{cases} C_1 \text{ é a raiz da árvore;} \\ C_{2i} = \text{subárvore da esquerda de } C_i \\ C_{2i+1} = \text{subárvore da direita de } C_i \end{cases}$$

Para i = 1, ..., N div 2

Classificação por Seleção - Heapsort

- **Método Heapsort - Descrição do Algoritmo**
  - ✓ Exemplificando: dado um vetor $V_{1..7}$, e utilizando a interpretação dada, podemos vê-lo como sendo a representação da seguinte árvore binária:



Para i = 1 até 7 div 2 = 1 .. 3

i = 1 -> Pai C_1 e Filhos: C_2 e C_3

i = 2 -> Pai C_2 e Filhos: C_4 e C_5

i = 3 -> Pai C_3 e Filhos: C_6 e C_7

Classificação por Seleção - Heapsort

- **Método Heapsort - Descrição do Algoritmo**
  - ✓ O passo seguinte consiste em trocar as chaves de posição dentro do vetor, de tal forma que estas passem a formar uma hierarquia, na qual todas as raízes das subárvores sejam maiores ou iguais a qualquer uma das suas sucessoras, ou seja, cada raiz deve satisfazer as seguintes condições:
    - $C_i \geq C_{2i}$
    - $C_i \geq C_{2i+1}$ Para i = 1, ..., N div 2.
  - ✓ Quando todas as raízes das subárvores satisfazem essas condições, dizemos que a árvore forma um ***heap***.

Classificação por Seleção - Heapsort

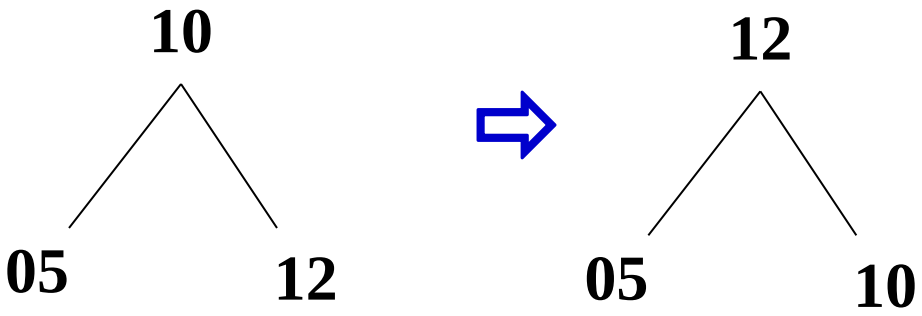
- **Método Heapsort - Descrição do Algoritmo**
  - ✓ O processo de troca de posições das chaves no vetor, de forma que a árvore representada passe a ser um heap, pode ser feito testando-se cada uma das subárvores para verificar se elas, separadamente, satisfazem a condição de heap.
  - ✓ Apenas as árvores que possuem pelo menos um sucessor devem ser testada.

Classificação por Seleção - Heapsort

- **Método Heapsort - Descrição do Algoritmo**
  - ✓ A maneira mais simples de realizar o teste é iniciando pela última subárvore, ou seja, aquela cuja raiz está na posição **N div 2** do vetor de chaves, prosseguindo, a partir daí, para as subárvores que a antecedem, até chegar à raiz da árvore.
  - ✓ No exemplo, a primeira subárvore a ser testada é aquela cuja raiz é C_3, depois a de raiz C_2 e finalizando com a de raiz C_1.

Classificação por Seleção - Heapsort

- **Método Heapsort - Descrição do Algoritmo**
  - ✓ Sempre que for encontrada uma subárvore que não forme um heap, seus componentes devem ser rearranjados de modo a formar o heap.



Classificação por Seleção - Heapsort

- **Método Heapsort - Descrição do Algoritmo**
  - ✓ Pode ocorrer também que, ao rearranjarmos uma subárvore, isto venha a afetar outra, do nível imediatamente inferior, fazendo que esta deixe de formar um heap.
  - ✓ Esta possibilidade obriga a verificar, sempre que for rearranjada uma subárvore, se a sucessora do nível abaixo não teve a sua condição de heap desfeita.

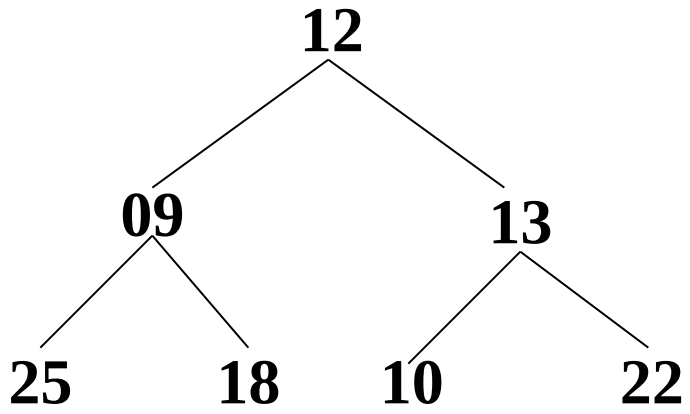
Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Objetivo: Seleção da maior chave

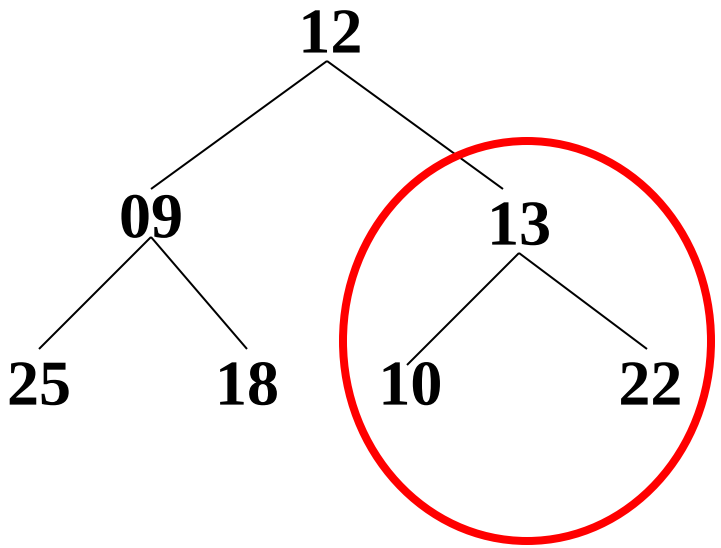
  Vetor inicial: 12 09 13 25 18 10 22

  Representação em árvore:



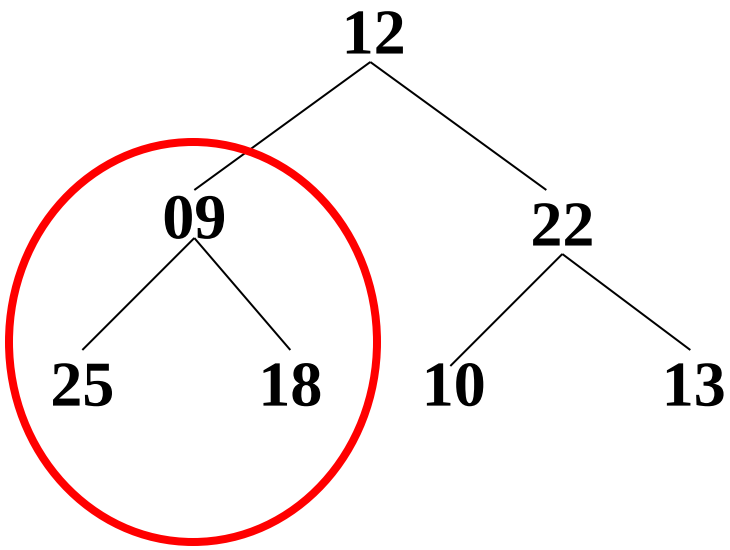
Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**
  - ✓ A transformação dessa árvore em heap inicia pela subárvore cuja raiz é **13**, já que seu índice, no vetor, é **7 div 2 = 3**.



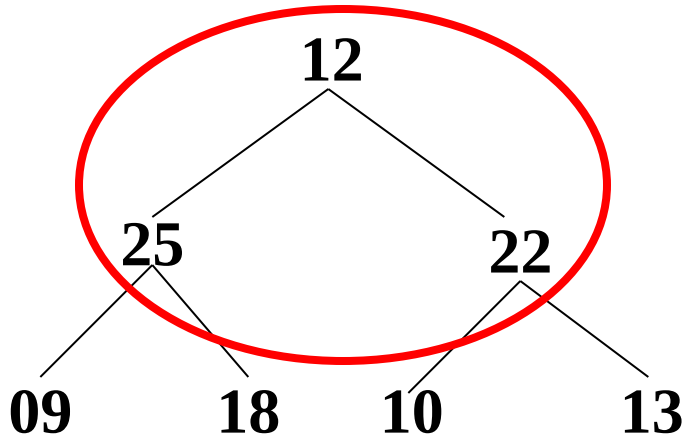
Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**
  - ✓ Vetor e árvore resultantes:
    - 12 09 22 25 18 10 13



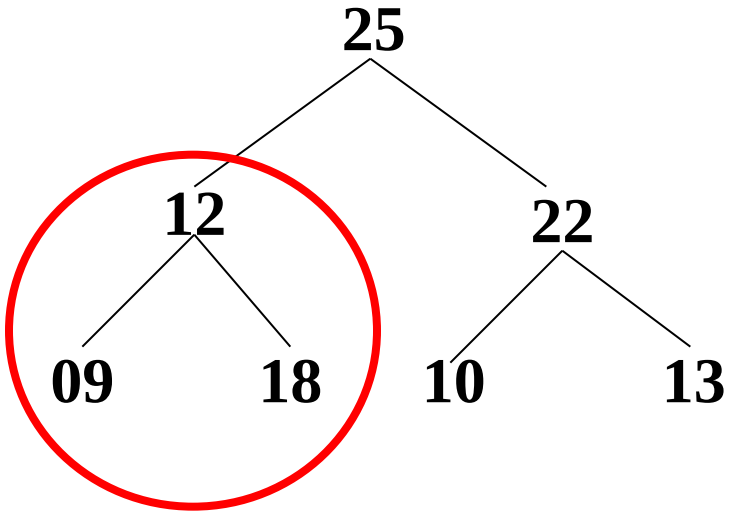
Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**
  - ✓ Vetor e árvore resultantes:
    - 12 25 22 09 18 10 13



Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**
  - ✓ Vetor e árvore resultantes: 25 12 22 09 18 10 13

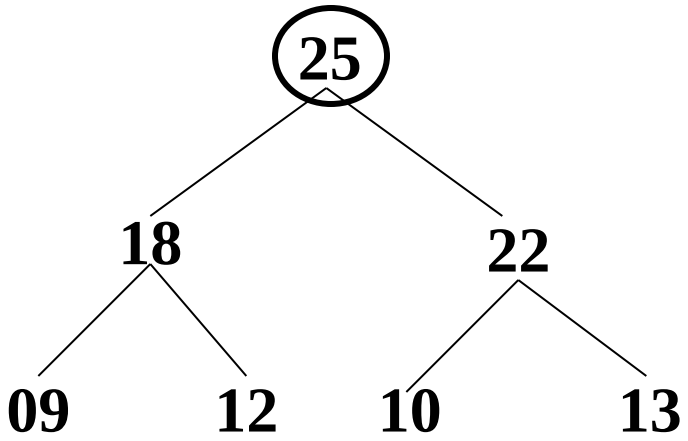


  - ✓ Neste caso ocorreu que a transformação da subárvore afetou outra de um nível mais abaixo. A subárvore afetada deve ser rearranjada.

Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Seleção das chaves: 25 18 22 09 12 10 13



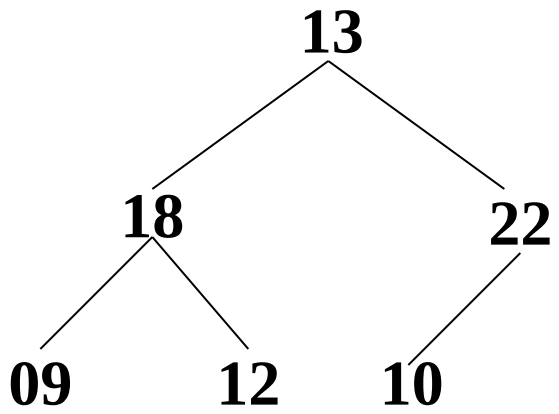
  - ✓ Se a chave que está na raiz é a maior chave de todas, então sua posição definitiva correta na ordem crescente é na última posição do vetor, onde ela é colocada, por troca com a chave que ocupa aquela posição.

Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Seleção das chaves: 13 18 22 09 12 10 | 25

  ✓ Com a maior chave já ocupando a sua posição definitiva podemos, a partir de agora, considerar o vetor como tendo um elemento a menos.

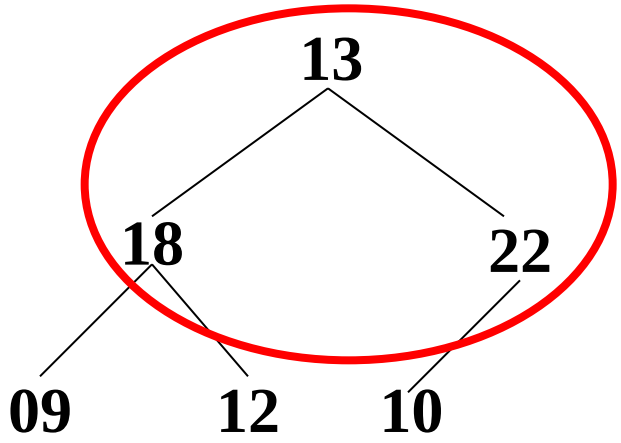


Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

Seleção das chaves: 13 18 22 09 12 10 | 25

✓ Para selecionar a próxima chave, deve-se fazer com que a árvore volte a ser um heap.

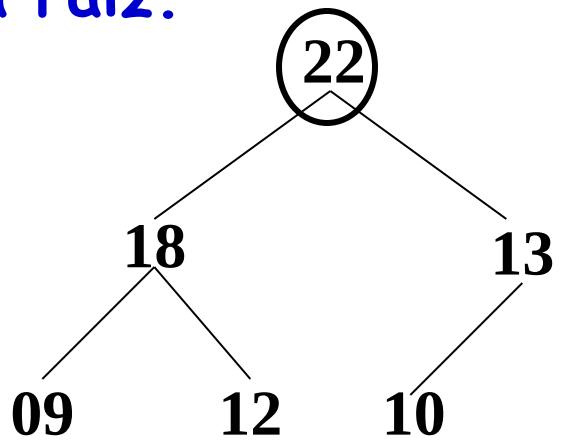


Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Seleção das chaves: 22 18 13 09 12 10 | 25

  ✓ **Novamente a maior chave dentre as restantes aparece na raiz.**

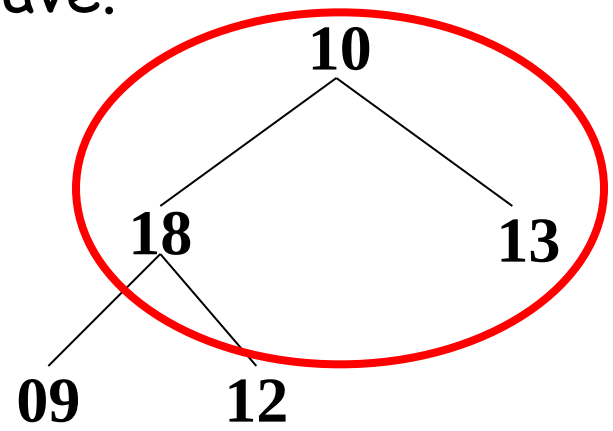


Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Seleção das chaves: 22 18 13 09 12 10 | 25

  - ✓ Vetor e árvore resultantes:10 18 13 09 12 | 22 25
  - ✓ A seguir a árvore é novamente rearranjada para formar um heap, o que permitirá selecionar a próxima chave.

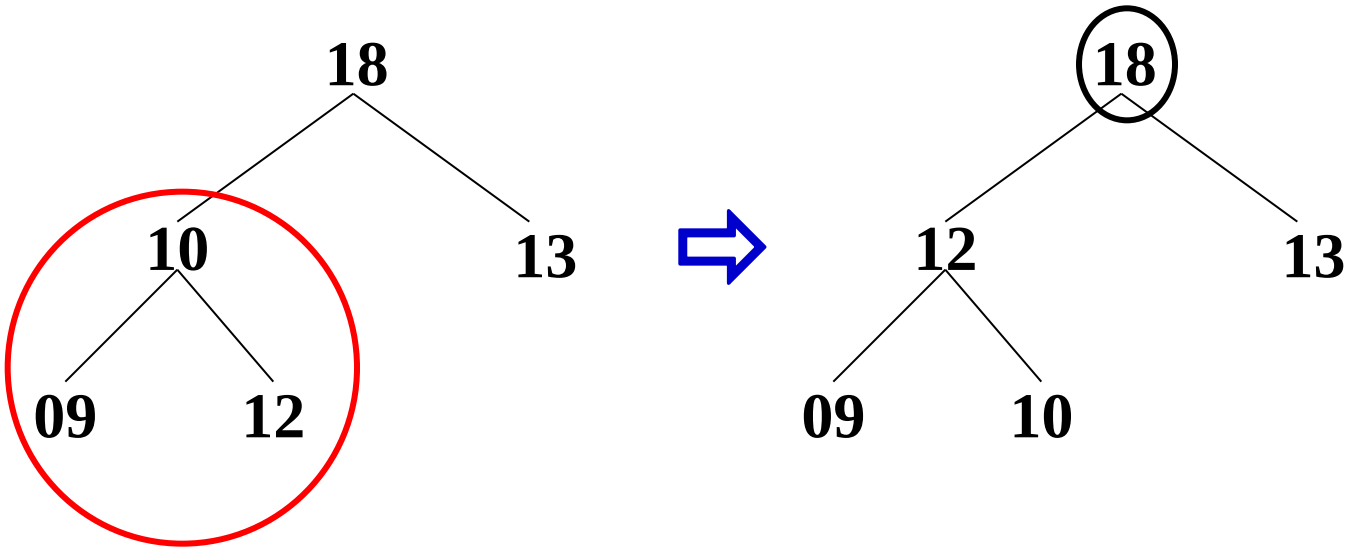


Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Seleção das chaves: 10 18 13 09 12 | 22 25

  - ✓ Vetor intermediário: 18 10 13 09 12 | 22 25
  - ✓ Vetor resultante: 18 12 13 09 10 | 22 25



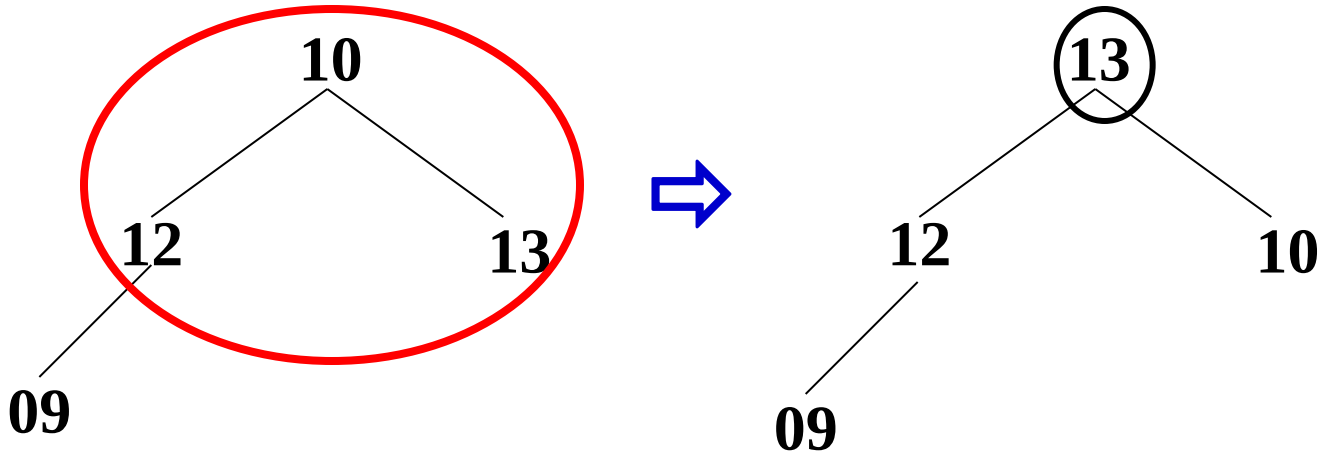
Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Seleção das chaves: 18 12 13 09 10 | 22 25

  ✓ Vetor intermediário: 10 12 13 09 | 18 22 25

  ✓ Vetor resultante: 13 12 10 09 | 18 22 25



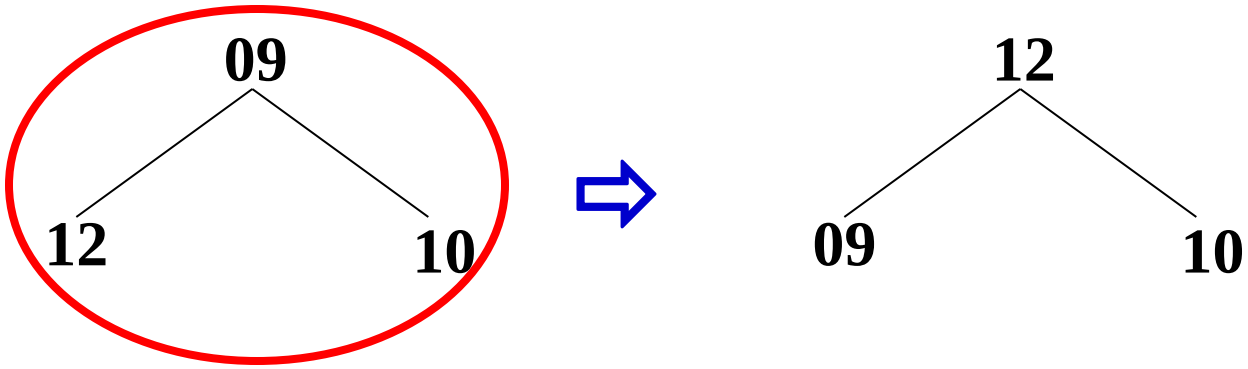
Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Seleção das chaves: 13 12 10 09 | 18 22 25

  ✓ Vetor intermediário: 09 12 10 | 13 18 22 25

  ✓ Vetor resultante: 12 09 10 | 13 18 22 25



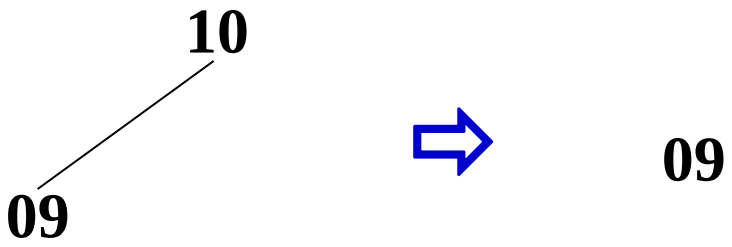
Classificação por Seleção - Heapsort

- **Método Heapsort - Exemplo**

  Seleção das chaves: 10 09 | 12 13 18 22 25

  ✓ Vetor resultante: 09 | 10 12 13 18 22 25

  Vetor final: 09 10 12 13 18 22 25



Classificação por Seleção - Heapsort

```
void heapsort( int a[], int n)
{
   int i = n/2, pai, filho, t;
   for (;;) {
      if (i > 0) {
         i--;
         t = a[i];
      }else{
         n--;
         if (n == 0)
            return;
         t = a[n];
         a[n] = a[0];
      }
      pai = i;
      filho = i*2 + 1;

    /* continua */
 }
}
```

Classificação por Seleção - Heapsort

```
void heapsort( int a[], int n)
{
    /* Continuando ...*/
     while (filho < n) {
          if ((filho + 1 < n)  &&  (a[filho + 1] > a[filho]))
              filho++;
          if (a[filho] > t) {
             a[pai] = a[filho];
             pai = filho;
             filho = pai*2 + 1;
          } else
             break;
      }
      a[pai] = t;

    }/* fim for*/
}
```

Classificação por Seleção - Heapsort

- Tem um bom desempenho em tempo de execução para conjuntos ordenados aleatoriamente. Tem um bom uso de memória e o seu desempenho no pior caso é praticamente igual ao desempenho no caso médio.
- Alguns algoritmos de ordenação rápidos têm desempenhos espetacularmente ruins no pior cenário para tempo de execução e uso da memória.
- O tempo de execução no pior caso para ordenar *n* elementos é de O (*n* log *n*). Para valores de *n*, razoavelmente grande, o tempo log *n* é quase constante, de modo que o tempo de ordenação é quase linear com o número de itens a ordenar.
- Características:
  - ✓ Comparações no pior caso: 2n log2n + O(n)
  - ✓ Trocas no pior caso: n log2n + O(n)
  - ✓ Melhor e pior caso: O(n log2n)

Classificação por Seleção - Heapsort

- **Método Heapsort – Exercício:**
  - ✓Utilizando o heapsort ordene o seguinte conjunto de elementos {85,99,98,97,96,95,94,93,92,91,90, 89,87,86}.