HEAPS

Python

HEAPS

Características:

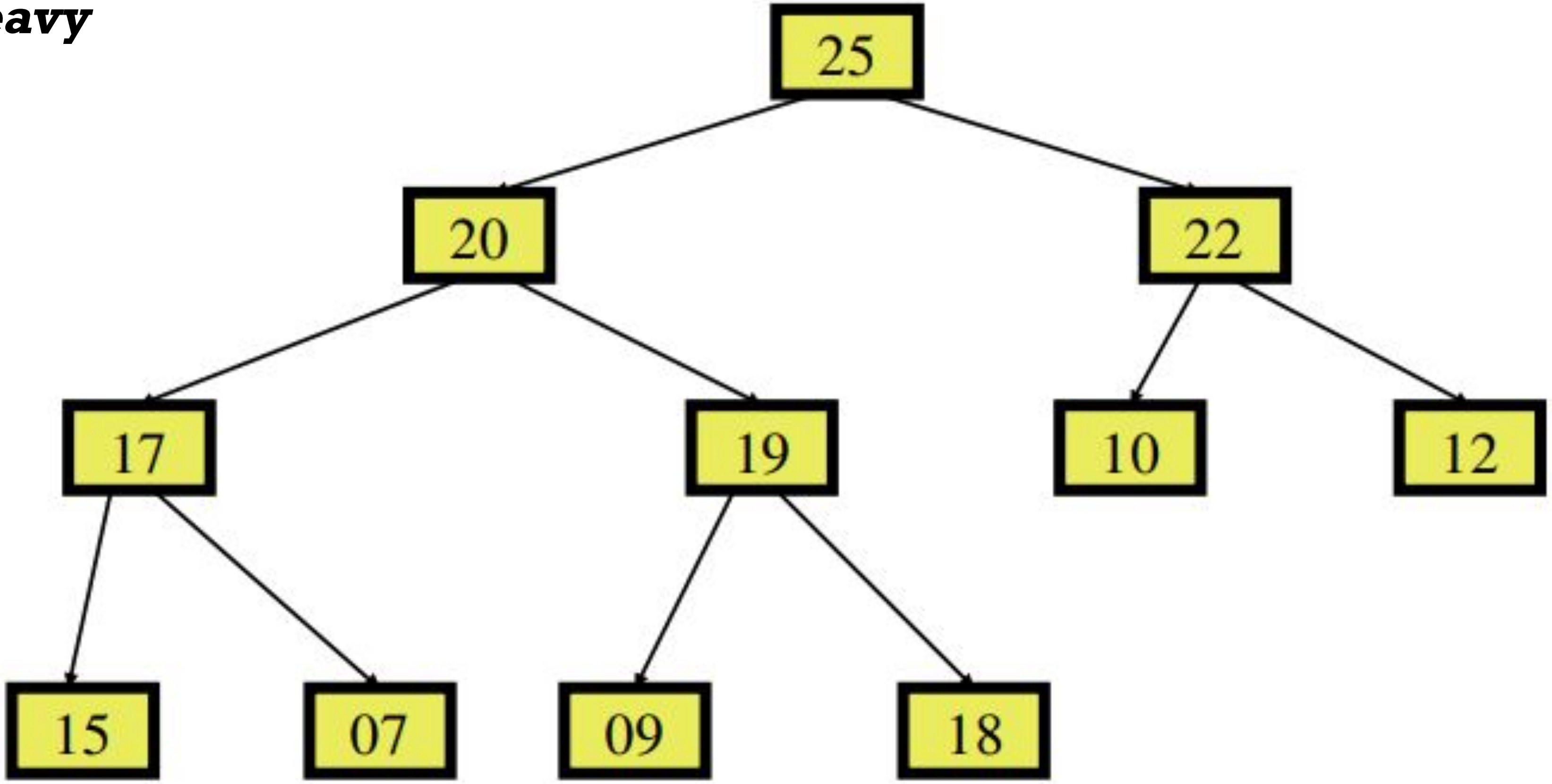
- Árvore na qual o valor existente no nó é menor que o valor do nó pai;
- O nó raiz armazena o maior valor da árvore;
- É uma estrutura parcialmente ordenada, pois cada galho tem uma ordenação específica;
- É possível a ordem dos valores dos nós;
- É uma árvore de prioridade já que o valor de cada nó é menor/maior ou igual que o valor dos seus nós filhos;

Conceitos:

- *Top-heavy* □ cada nó filho tem um valor menor que o do seu nó pai;
- *Bottom-heavy* □ cada nó filho tem um valor maior que o do seu nó pai;

HEAPS

Top-heavy

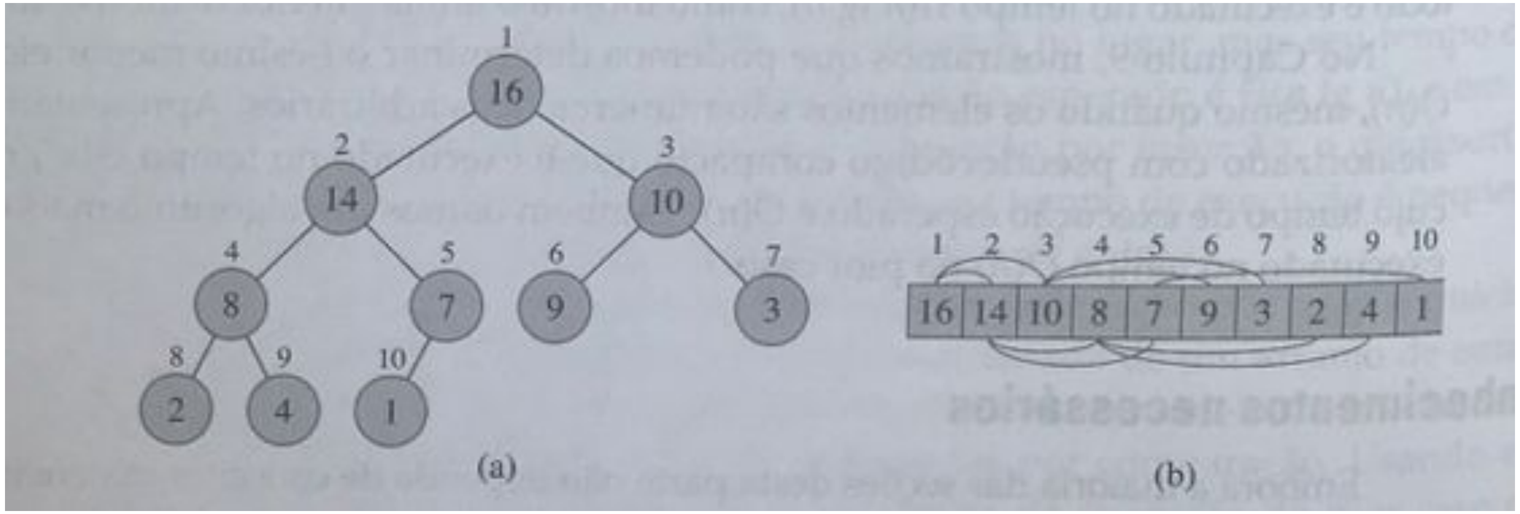


Rômulo Silva de Oliveira, DAS-UFSC, maio/

HEAPS

O que :

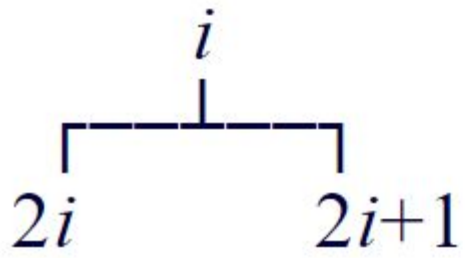
- Trata-se de um algoritmo de ordenação;
- É um arranjo que pode ser entendido com uma árvore binária quase completa;
- Cada nó da árvore corresponde a um elemento do arranjo;
- É uma árvore que está completamente preenchida em todos os níveis, exceto no nível mais baixo, que é preenchido a partir da esquerda, até um ponto;
- Útil na construção de filas com prioridades;



HEAPS

Cálculo dos índices :

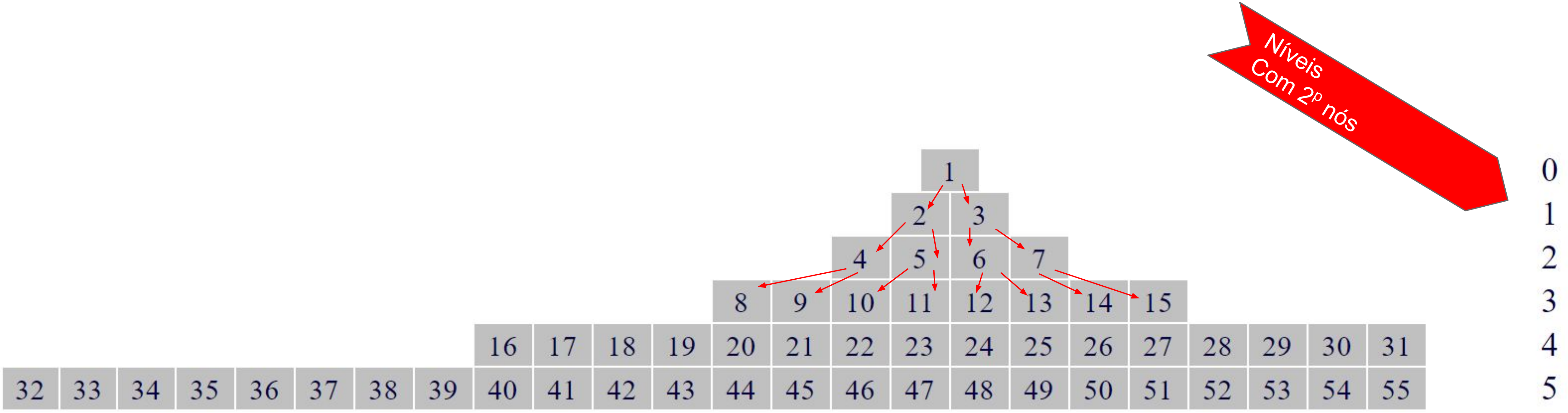
- Nó pai: [i / 2]
- Nó filho esquerdo: [2i]
- Nó filho direito: [2i + 1]
- O nível i tem 2^i nós (2^p, 2^p+1,2^p+2,...,$2^{p+1}-1$);



Altura de um nó

- Número de nós no caminho descendente mais longo, desde o nó até uma folha;
- Corresponde a altura de uma raiz, uma vez que podemos estar calculando a altura de uma sub-árvore;
- A altura de um heap (visto como uma árvore binária) corresponde a altura de uma raiz;
- Sua altura é baseada na altura de uma árvore completa - □(lg n)

HEAPS


Níveis
Com 2p nós
1
2 3
4 5 6 7
8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55
0
1
2
3
4
5

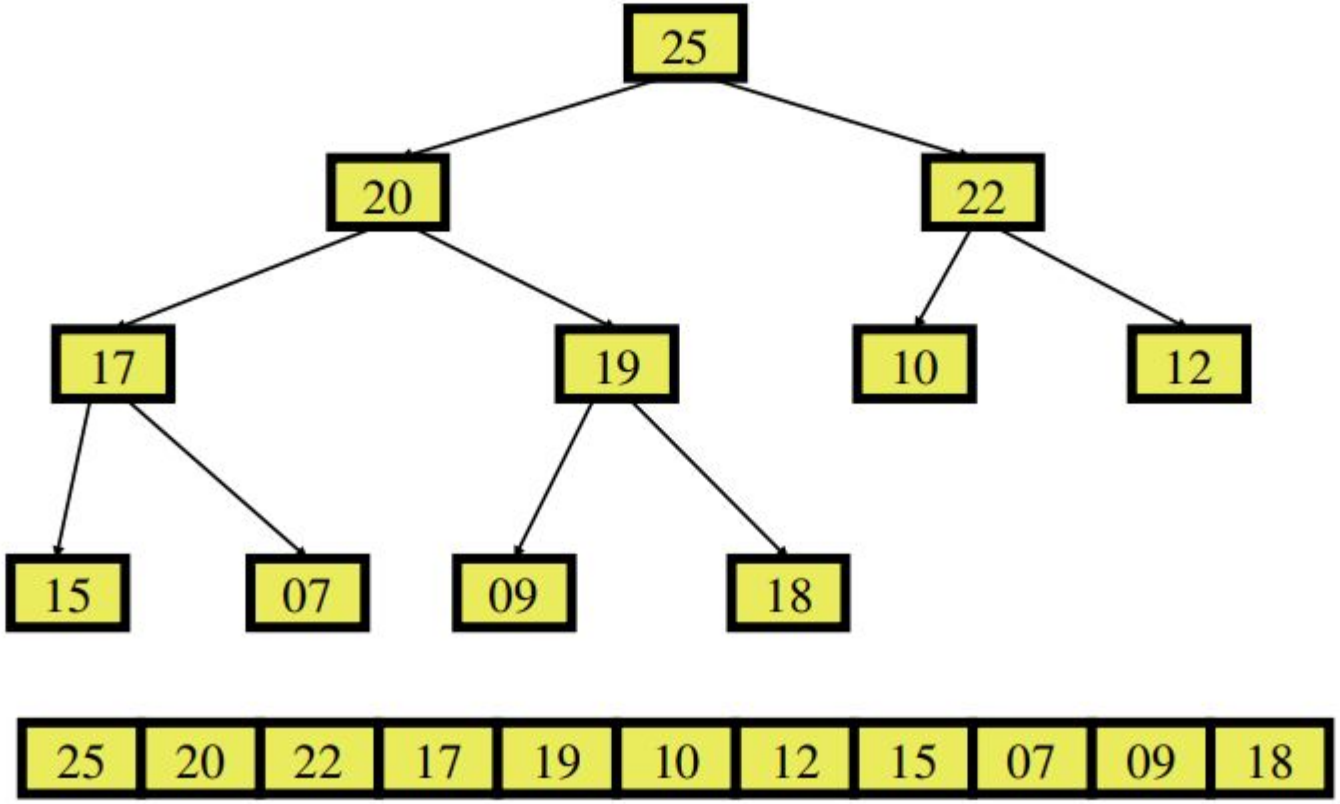
HEAPS

Características:

- Os valores podem ser armazenados em um array mantendo a sua ordem original;
- Os elementos do vetor são identificados pelos índices 1 a n
- O índice 1 não tem pai;
- um índice p só tem filho esquerdo se $2p \leq n$;
- um índice p só tem filho direito se $2p+1 \leq n$
- O nó pai de cada nó na posição p está na posição $(p - 1)/2$;
- O nó filho esquerdo está na posição $(2p + 1)$;
- O nó filho direito está na posição $(2p)$;
- O último nó é aquele que se encontra mais à direita, no último nível;
- O nó p pertence ao nível $\lfloor \lg p \rfloor$;
- O número total de níveis é $1 + \lfloor \lg p \rfloor$;

HEAPS

Heap representado como array:



Rômulo Silva de Oliveira, DAS-UFSC, maio/2011

A [i, 2i, 2i+1, 4i, 4i+1, 4i+2, 4i+3, 8i, … , 8i+7, …]

HEAPS

Tempo das operações básicas:

- São calculadas em tempo, no máximo, proporcional à altura da árvore -> O(lg n);

Tipos de Heaps binários

- Heap de máximo: todo nó i, exceto o raiz:
    - A[Pai(i)] >= A[i] (no nó esquerdo) ou seja, A[i/2] >= A[i], para i = 2, …, n
    - A[j] >= A[2j] e A[j] >= A[2j+1]
    - O maior elemento está na raiz;
    - É mais fácil encontrar o elemento máximo;
    - Fácil de corrigir o heap quando o elemento raiz for alterado;
    - É o tipo de Heap padrão;
    - Um vetor qualquer pode ser transformado em um heap rapidamente;

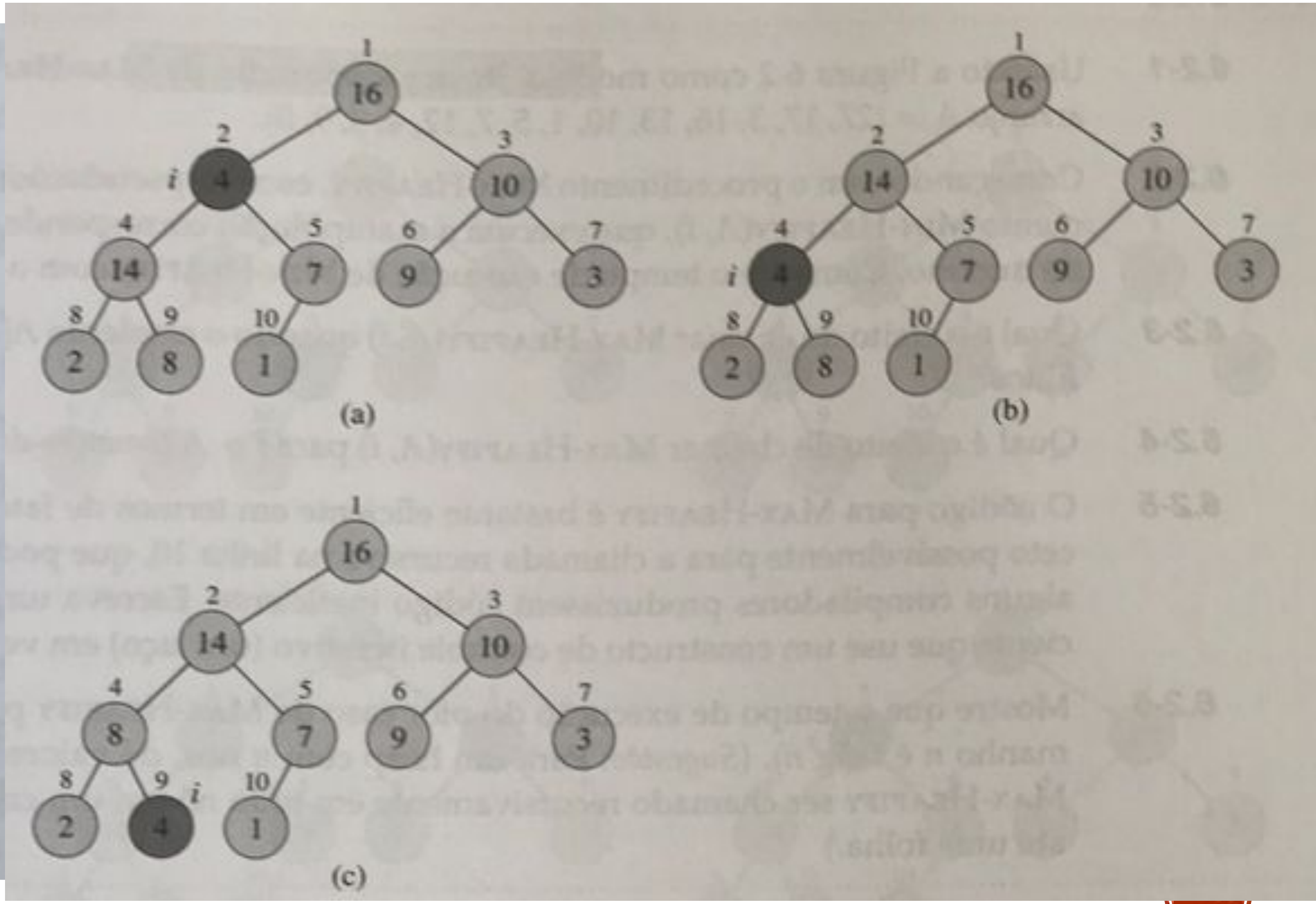
HEAPS

- Heap de mínimo: todo nó i, exceto o raiz:
  - A[Pai(i)] <= A[i] ou seja, A[i/2] <= A[i], para i = 2, …, n;
  - O menor elemento está na raiz;
  - A[j] <= A[2j] e A[j] <= A[2j+1]
  - O menor elemento está na raiz;
  - É mais fácil encontrar o menor elemento;
  - Fácil de corrigir o heap quando o elemento raiz for alterado;
  - Um vetor qualquer pode ser transformado em um heap rapidamente;

HEAPS

Algoritmo Max-Heapify

```
MAX-HEAPIFY(A, i)
  l = LEFT(i)
  r = RIGHT(i)
  if l ≤ A·tamanho-do-heap e A[l] > A[i]
      maior = l
  else maior = i
  if r ≤ A·tamanho-do-heap e A[r] > A[maior]
      maior = r
  if maior ≠ i
      trocar A[i] com A[maior]
      MAX-HEAPIFY(A, maior)
```



(a) (b) (c)

HEAPS

Algoritmo Max-Heapify

```
Corrige-Descendo (A, n, i)
    j := i
    enquanto 2 j ≤ n
        f := 2 j
        se f < n e A[f] < A[f + 1]
            f := f + 1
        se A[ j] ≥ A[f]
            j := n
        senão troque A[ j] :=: A[f]
            j := f
```

```
Corrige-Subindo (A, m)
    i := m
    enquanto i ≥ 2 e A[⌊i/2⌋] < A[i]
        troque A[⌊i/2⌋] :=: A[i]
        i := ⌊i/2⌋
```

```
Constrói-Max-Heap (A, n)
    para i := ⌊n/2⌋ decrescendo até 1
        Corrige-Descendo (A, n, i)
```

HEAPS

Construindo um Heap

- Pode-se usar o procedimento Max-Heapify para converter um arranjo A[1..n] onde, em um heap de máximo,

n = A.comprimento;

Método build-Max-Heap

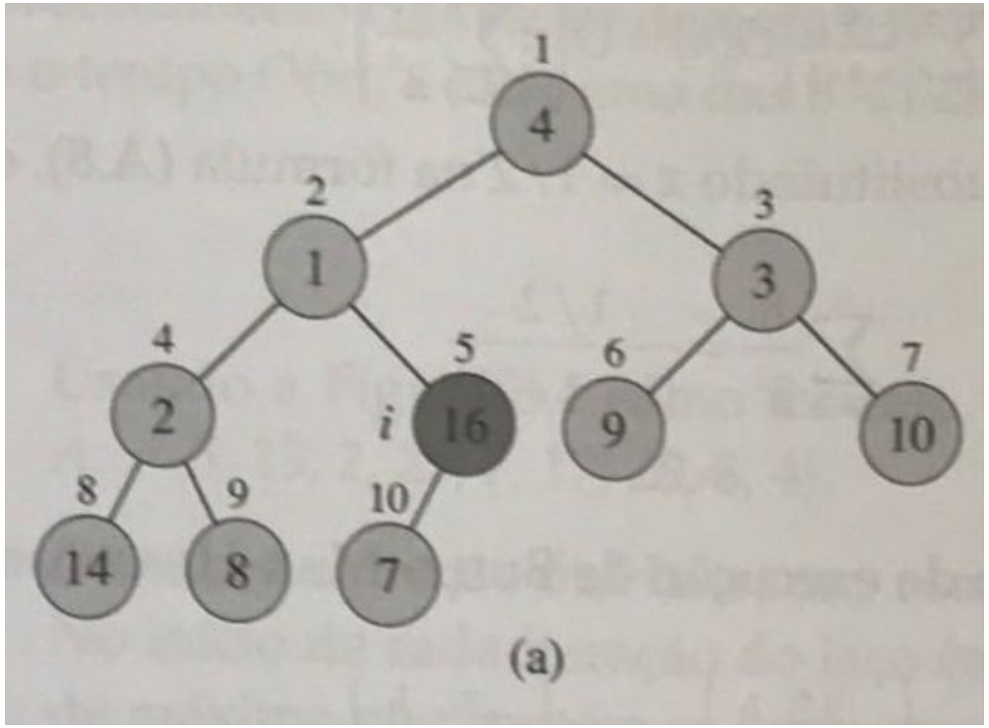
- Os filhos do nó i possuem índices maiores que i;
- Cada nó que possui filhos é raiz de heap de máximo;
- Esse método percorre os nós da árvore e executa o procedimento Max-HeapiFy;

```
BUILD-MAX-HEAP(A)
  A·tamanho-do-heap = A·comprimento
  for i = ⌊comprimento[A]/2⌋ downto 1
      MAX-HEAPIFY(A, i)
```

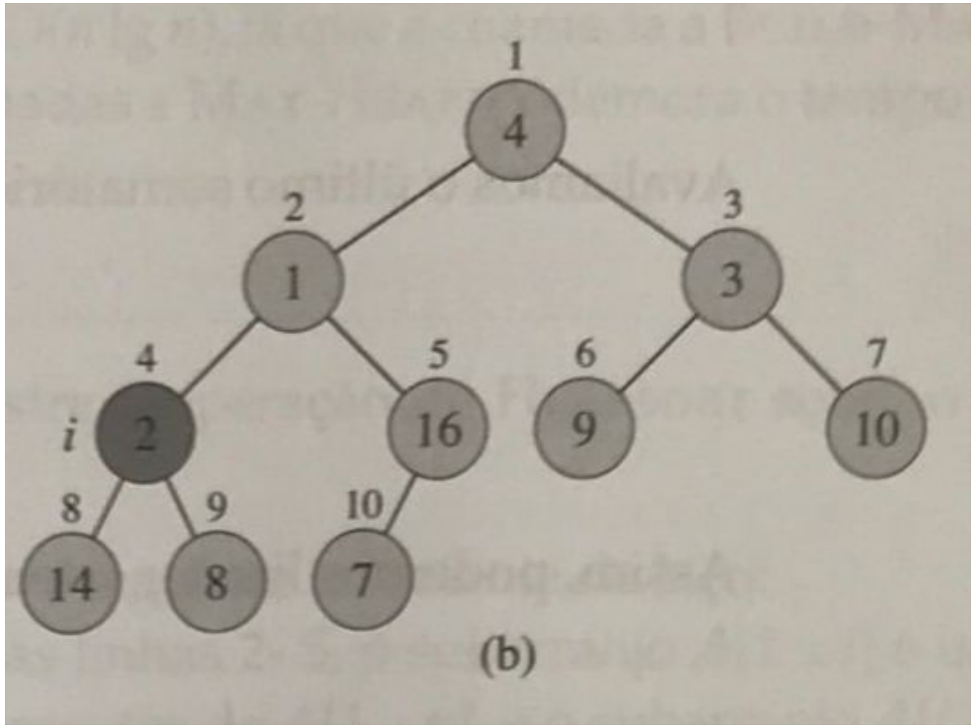
HEAPS

Construindo um Heap

A = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7}



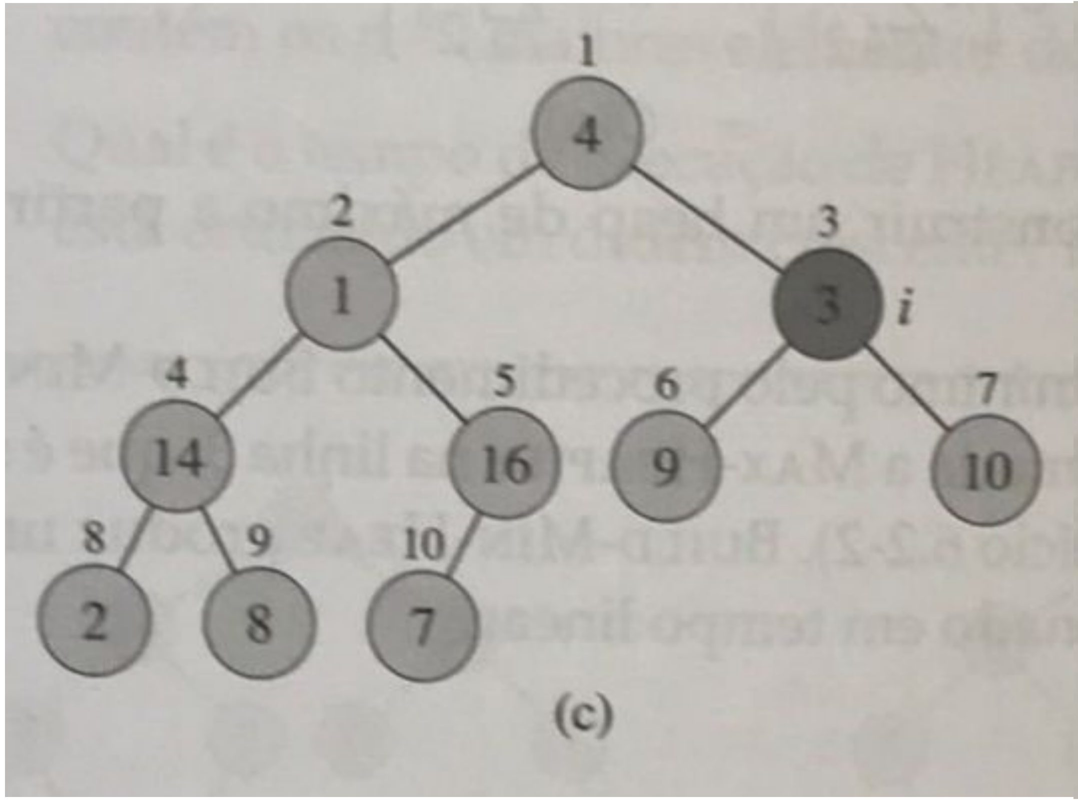
(a)



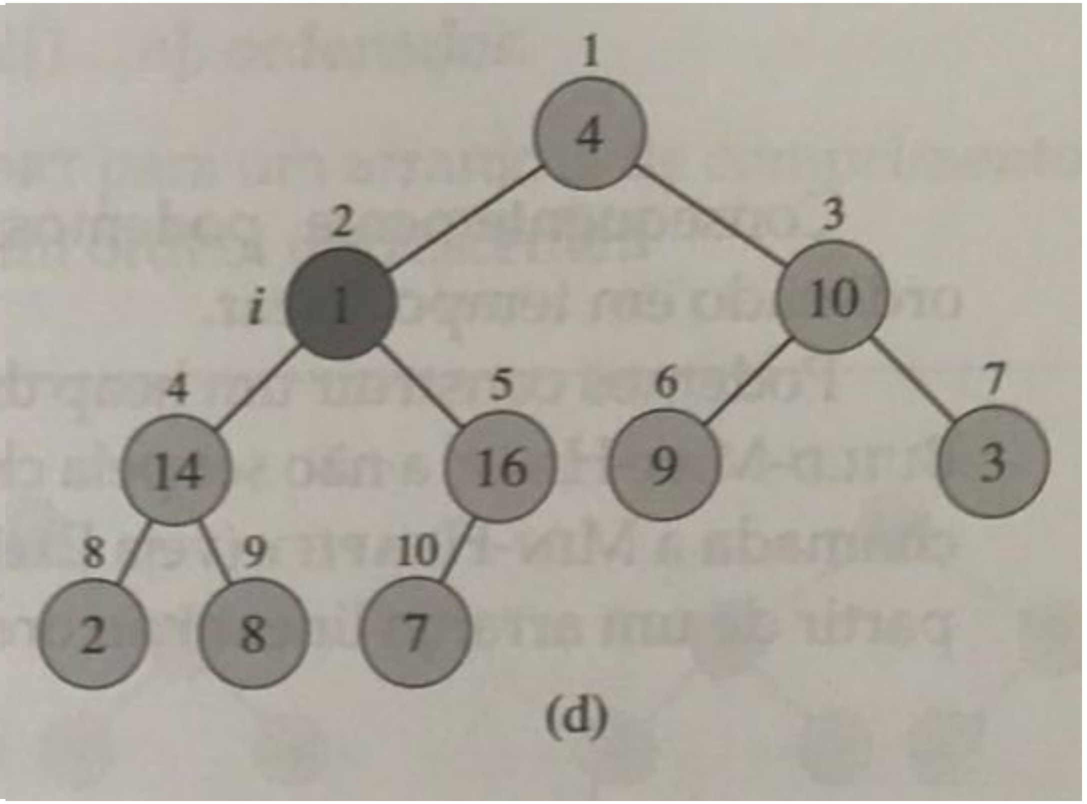
(b)

HEAPS

Construindo um Heap



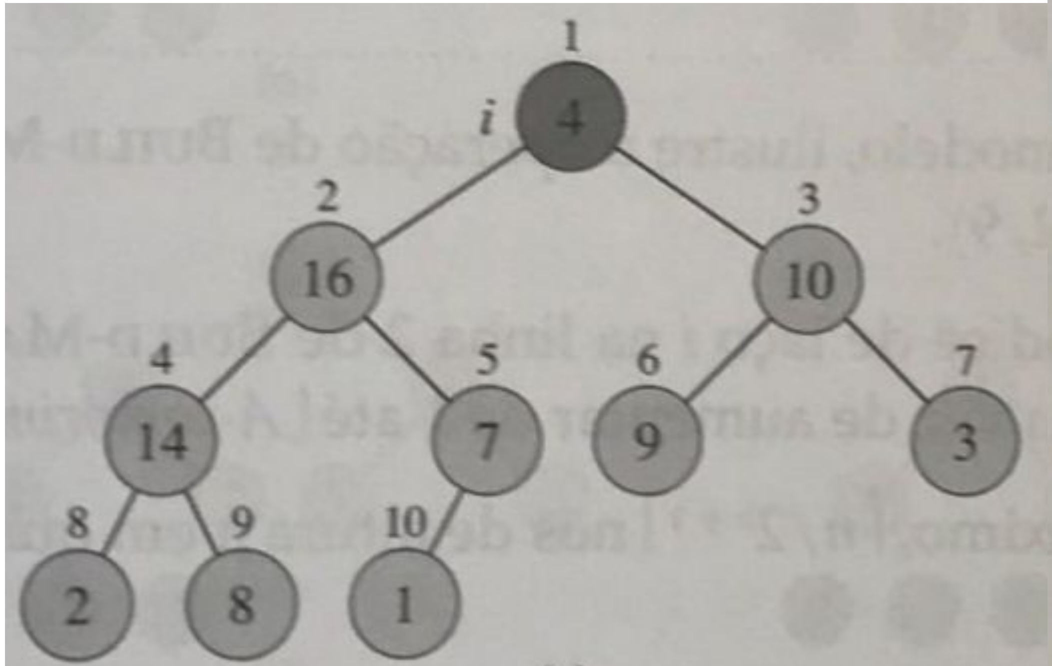
(c)



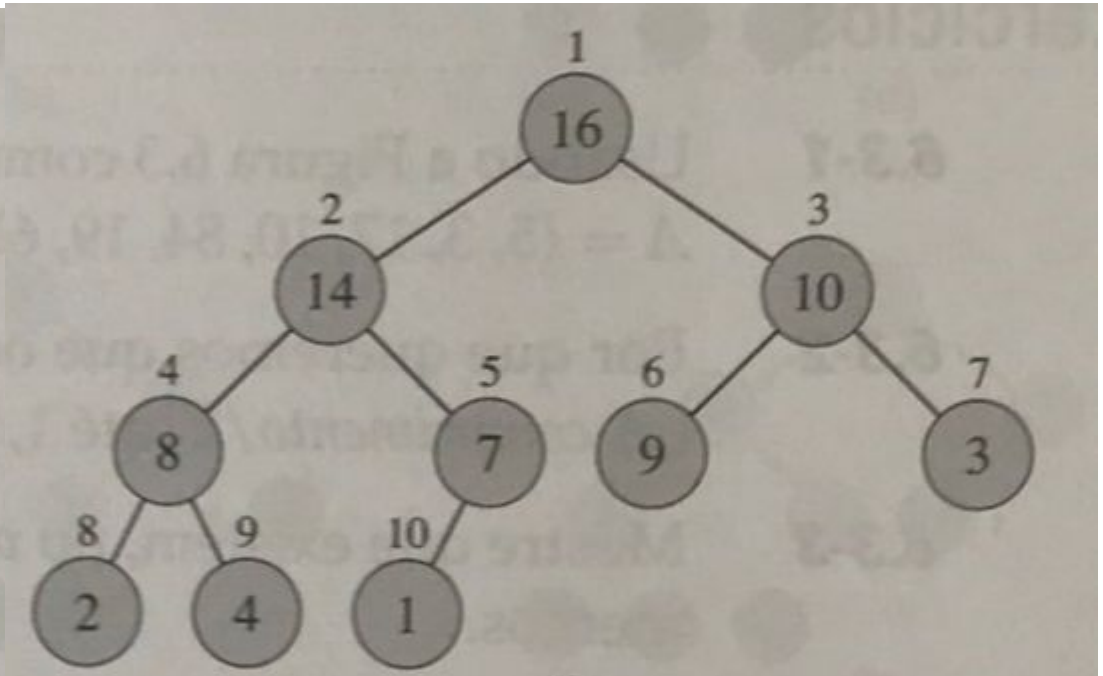
(d)

HEAPS

Construindo um Heap



(e)

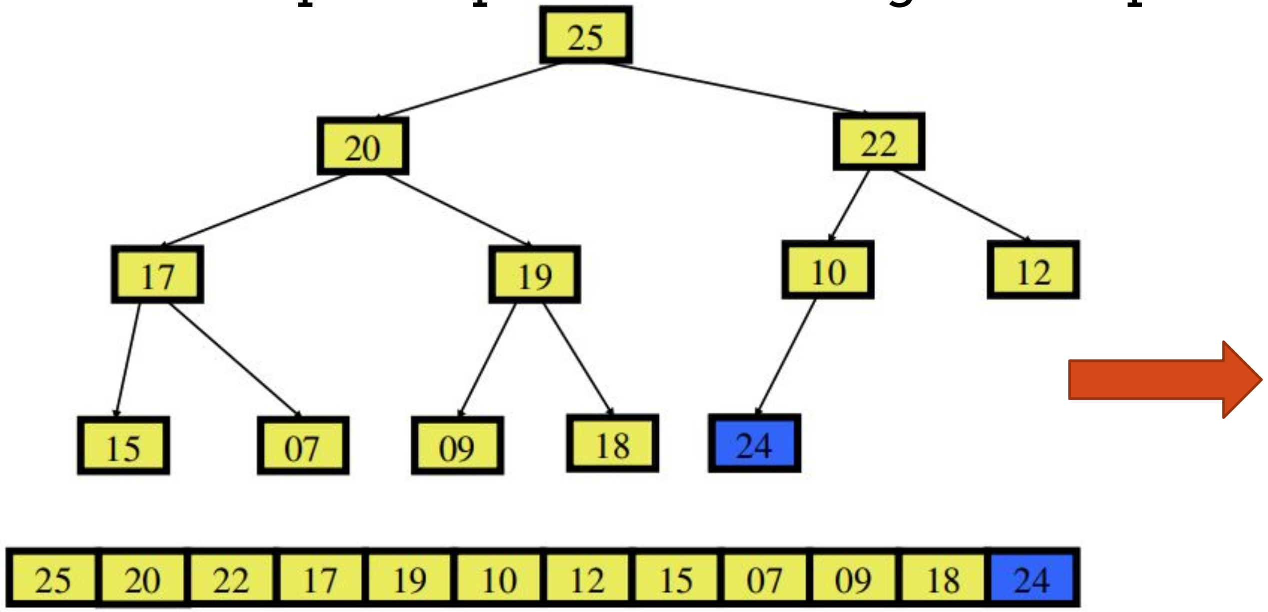


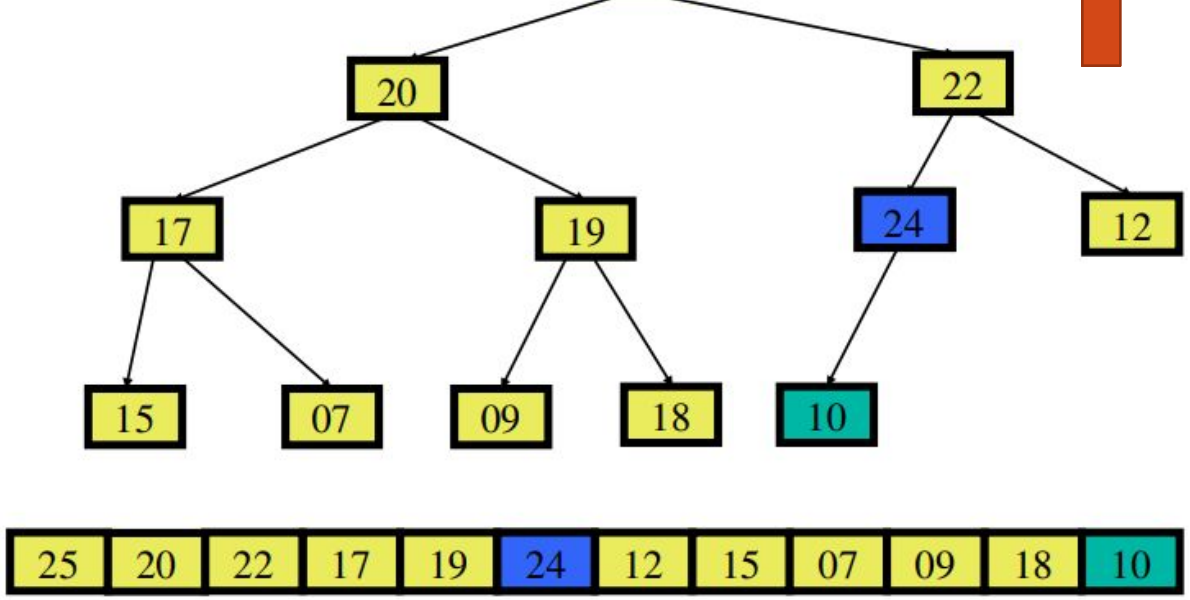
(f)

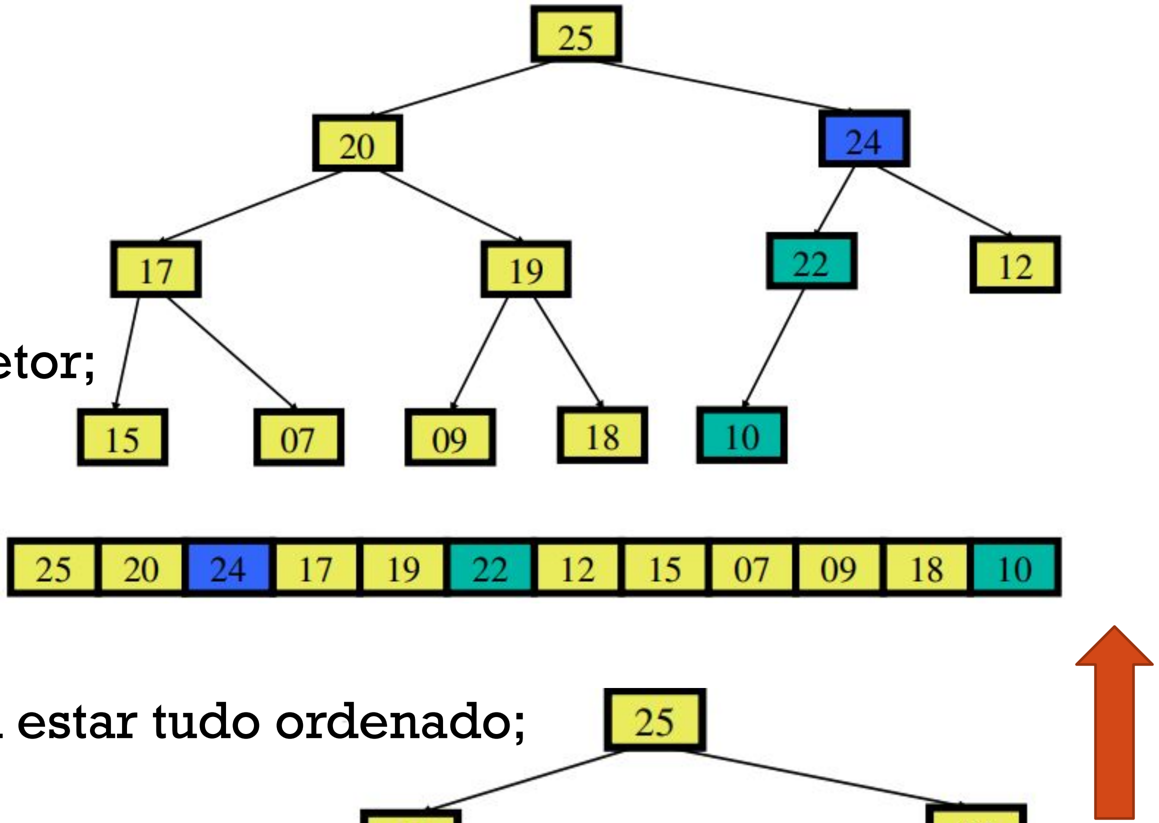
HEAPS

Inserindo um novo valor:

- Insere o novo valor na última posição do vetor;
- Reordena os valores comparando com o valor do nó pai;
- O maior valor sempre fica no nó pai;
- Repetir o processo até chegar ao nó pai ou estar tudo ordenado;

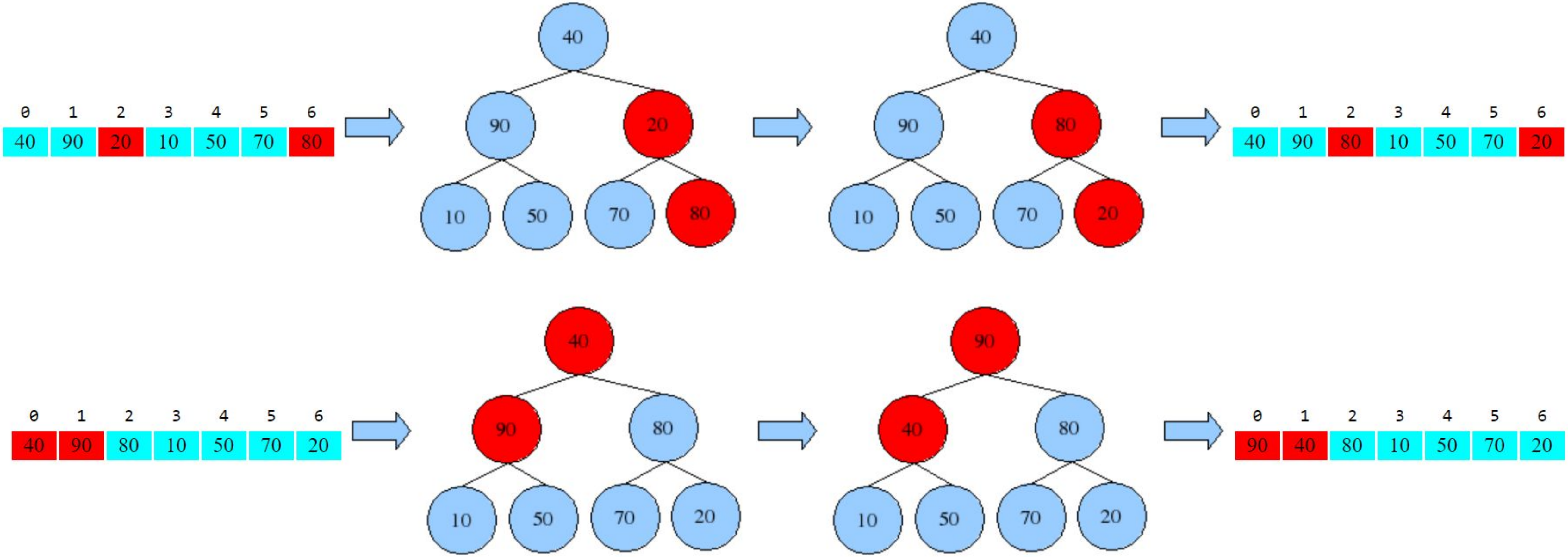






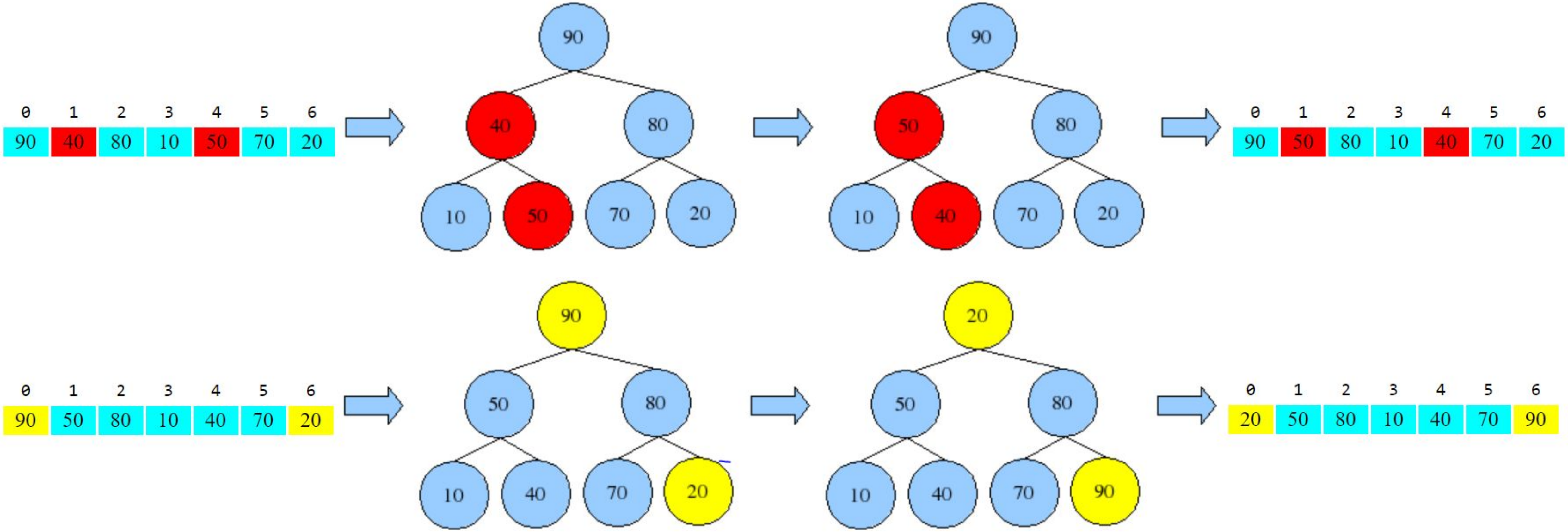
HEAPS

Inserindo um novo valor:



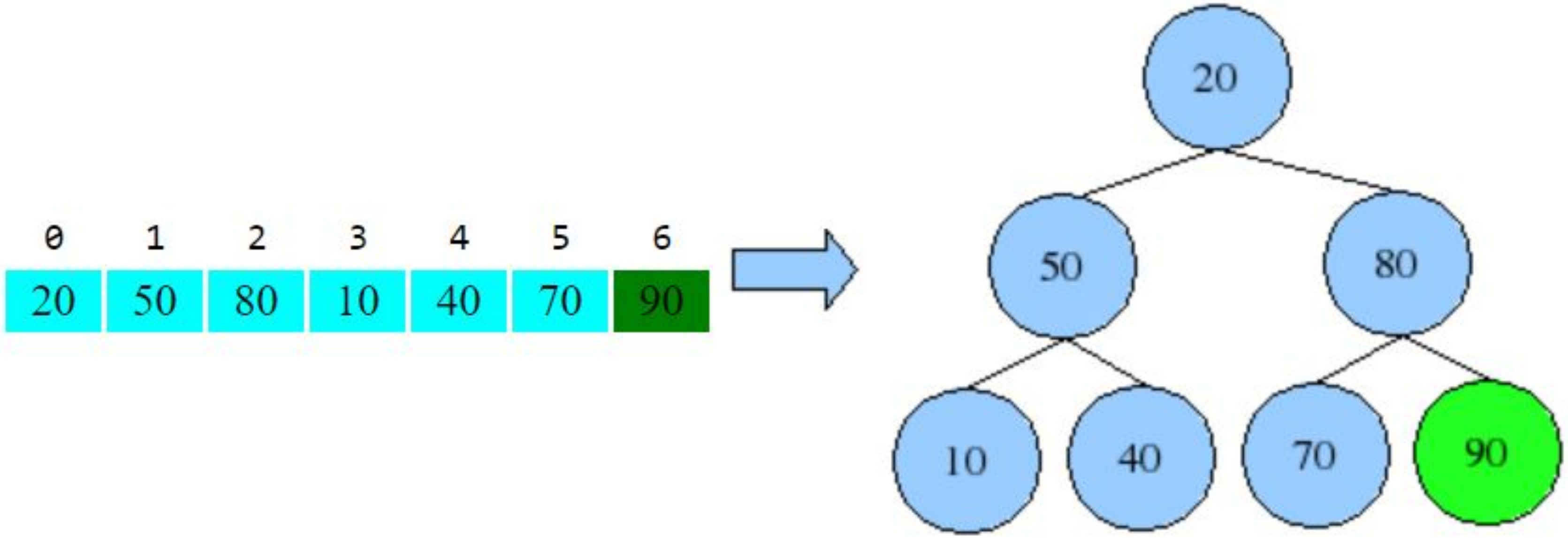
HEAPS

Inserindo um novo valor:



HEAPS

Inserindo um novo valor:

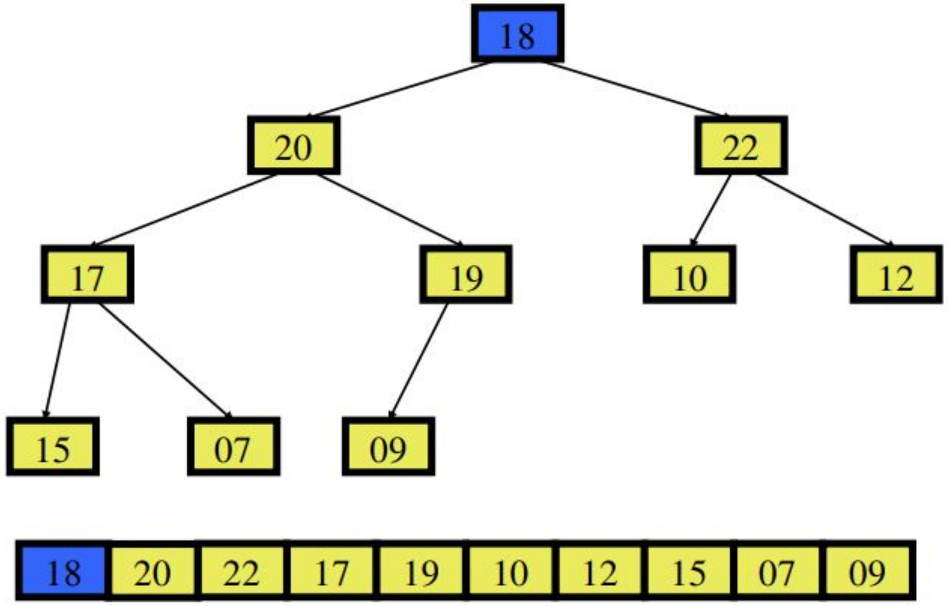


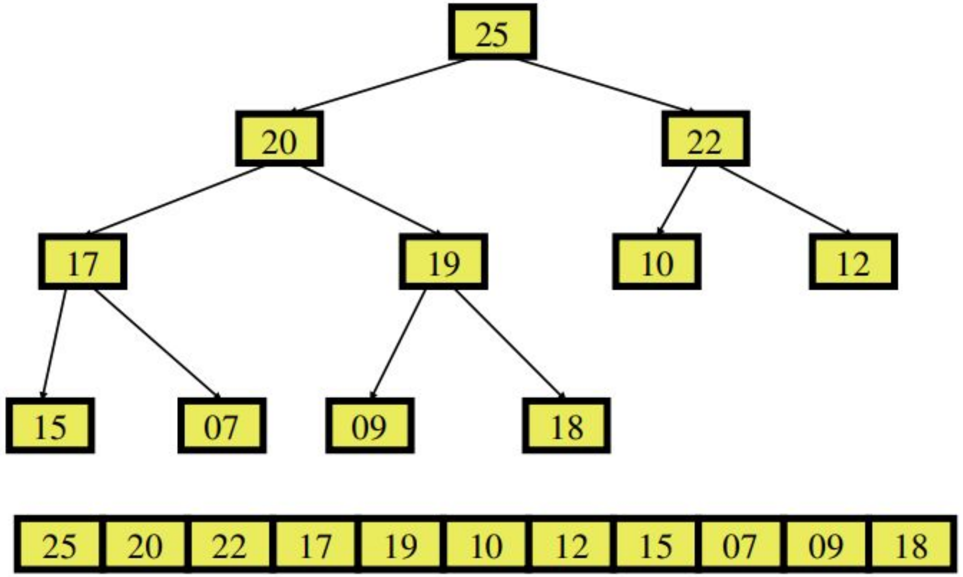
Se o nó pai for modificado o processo de ordenação recomeça.

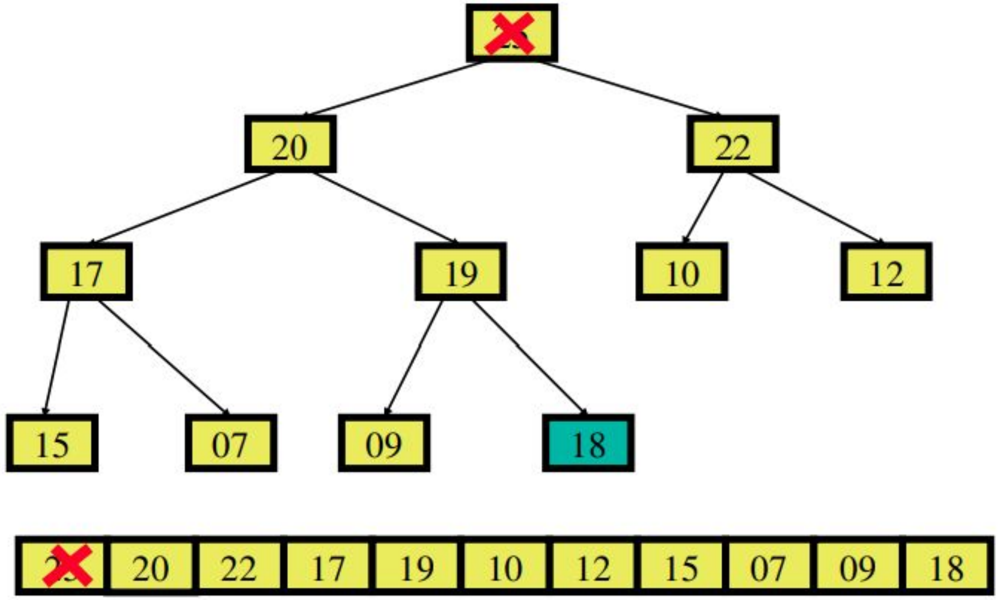
HEAPS

Removendo um novo valor:

- Remove o valor do primeiro nó (nó do topo);
- O valor do último nó é movido para o topo;
- Reordenação dos valores do heap, trocando-os continuamente até chegar ao topo ou não haver mais valores para reposicionar;



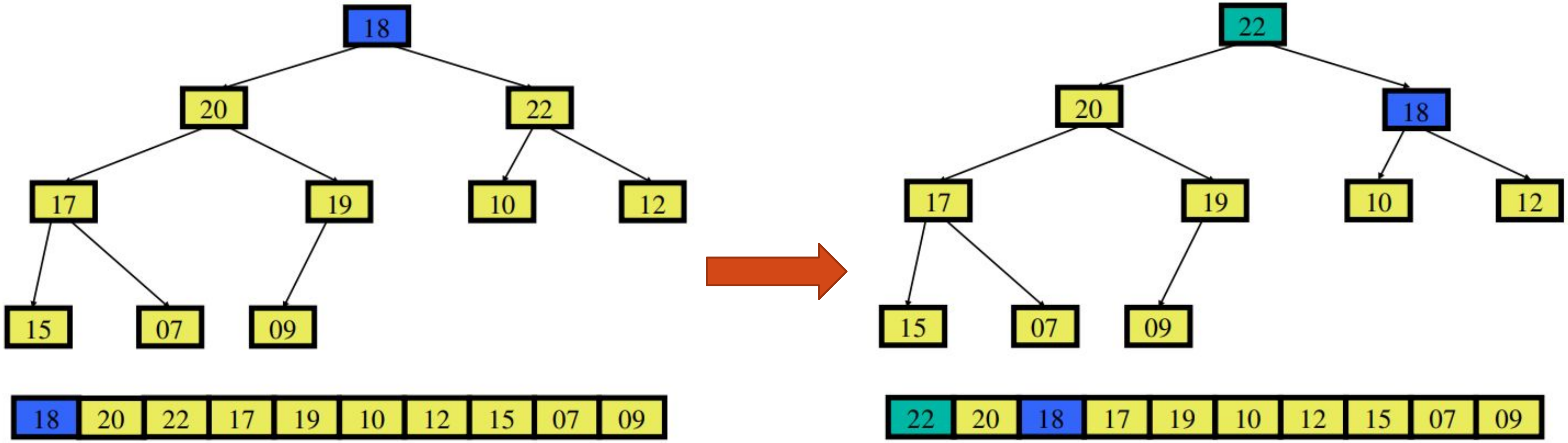




HEAPS

Removendo um novo valor:

- Remove o valor do primeiro nó (nó do topo);
- O valor do último nó é movido para o topo;
- Reordenação dos valores do heap, trocando-os continuamente até chegar ao topo ou não haver mais valores para reposicionar;



REFERÊNCIAS

https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/heap.html

https://www.cos.ufrj.br/~rfarias/cos121/aula_09.html

https://docente.ifrn.edu.br/robinsonalves/disciplinas/estruturas-de-dados-2013/08Heap.pdf

http://www2.ic.uff.br/~boeres/slides_ed/ed_Heap.pdf

https://www2.dc.ufscar.br/~mario/ensino/2019s1/aed2/aula12.pdf

http://www.inf.puc-rio.br/~amoura/inf1010/Heap.pdf

https://www.cin.ufpe.br/~afga/Heaps.pdf