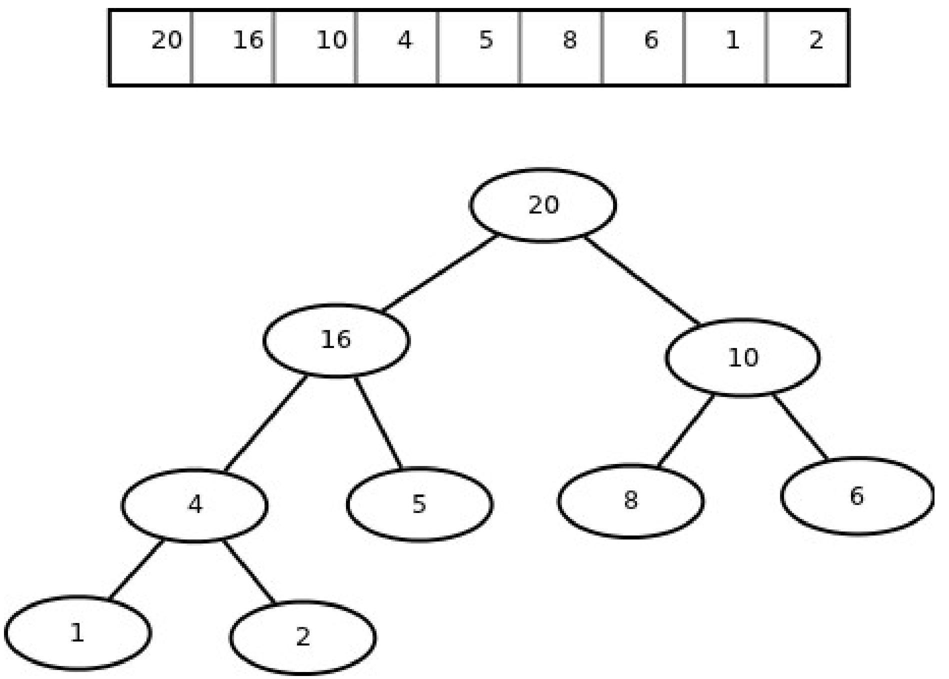
Heap



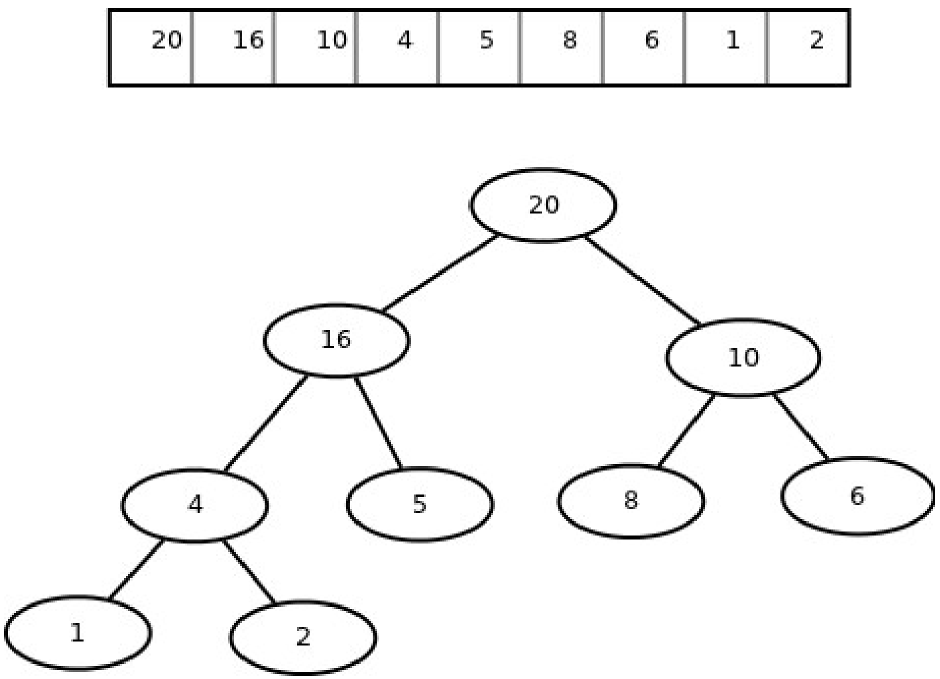
- Cada nó da árvore corresponde a um elemento do vetor que armazena o valor no nó.
- A árvore está completamente preenchida em todos os níveis, exceto possivelmente no nível mais baixo, que é preenchido da esquerda para a direita até certo ponto.
- Um vetor V representa uma estrutura heap através de dois parâmetros:
  - tamanho total do vetor: (V.length);
  - comprimento da parte do vetor que contém elementos da estrutura heap: heap_length[V]
- A raiz da árvore representada na estrutura heap é V[0].
- Dado um elemento da estrutura heap de índice i:
  - pai(i): (i-1)/2
  - esquerda(i): $2 * i + 1$
  - direita(i): $2 * i + 2$
-

2 “tipos” de Heap

- Propriedades de heap máximo
- A[pai(i)] ≥ A[i].
  - Isto é, o valor de um nó é no máximo o valor de seu pai.
- O maior elemento do heap está na raiz.
- As subárvores de um nó possuem valores menores ou iguais ao do nó.

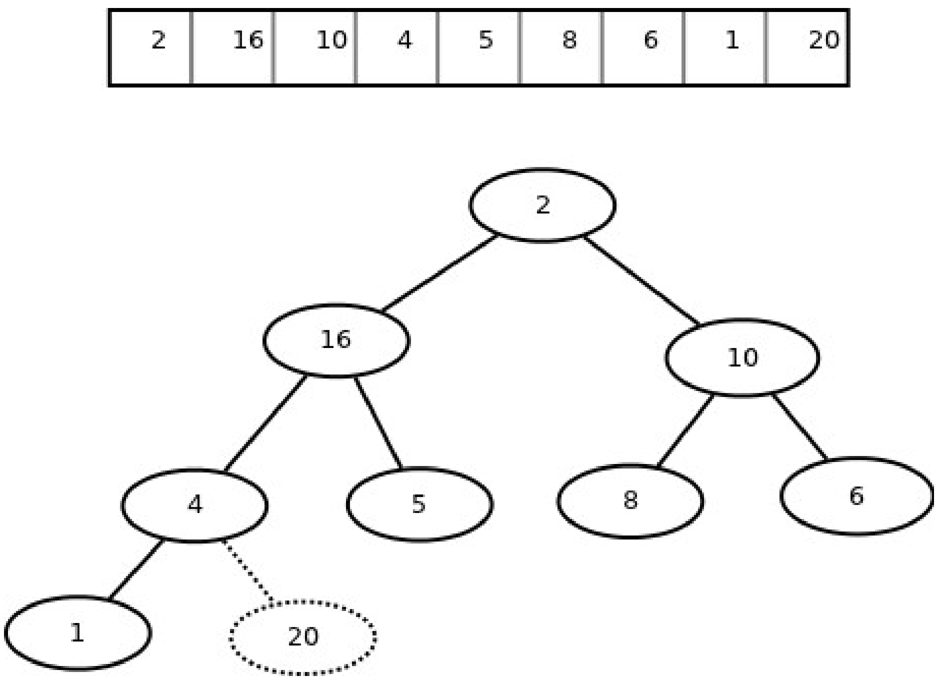
- Propriedades de heap mínimo
- A[pai(i)] ≤ A[i].
  - Isto é, o valor de um nó é maior ou igual o valor de seu pai.
- O menor elemento do heap está na raiz
- As subárvores de um nó possuem valores maiores ou iguais ao do nó.

Este é máximo ou mínimo?



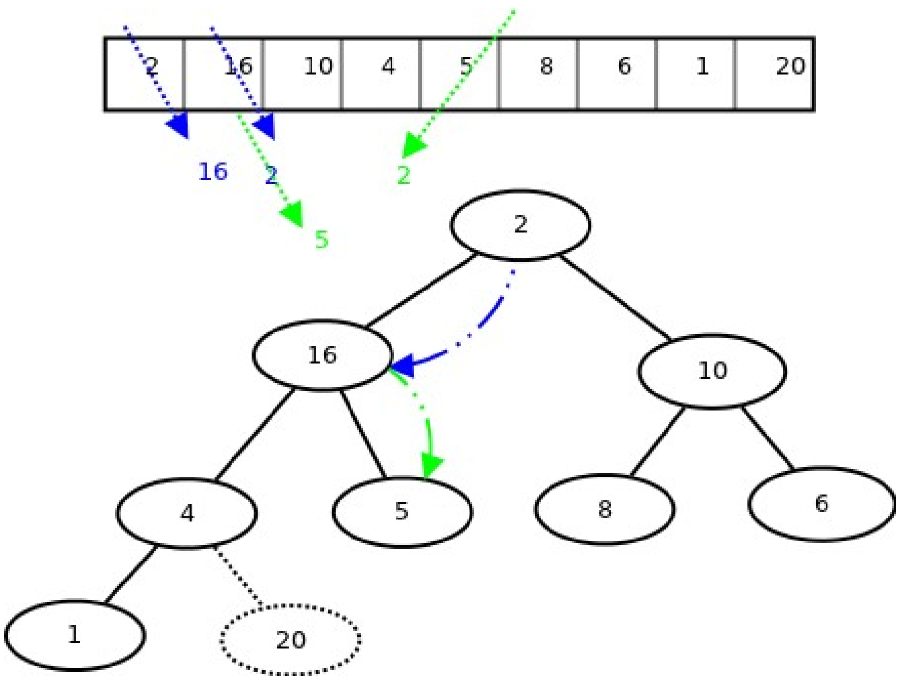
- Como remover o elemento da raiz e consertar o heap?
- Quando remove o elemento, cria-se um buraco e o comprimento do heap diminui em 1 unidade

...se tirar o elemento da raiz?



- Como o heap diminui, então pode pegar o último e colocar na raiz... mas estraga a propriedade do heap.
- … tem que consertar...

Conserto



Max-Heapify(A, i)

left ← 2i

right ← 2i + 1

largest ← i

if left ≤ heap_length[A] and A[left] > A[i] then:

largest ← left

if right ≤ heap_length[A] and

A[right] > A[largest] then:

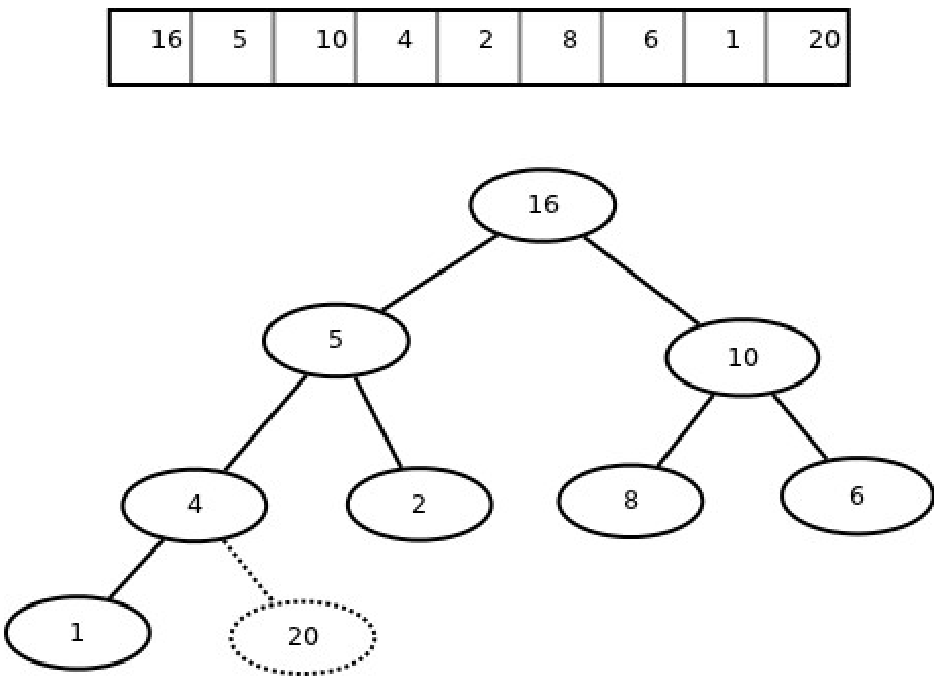
largest ← right

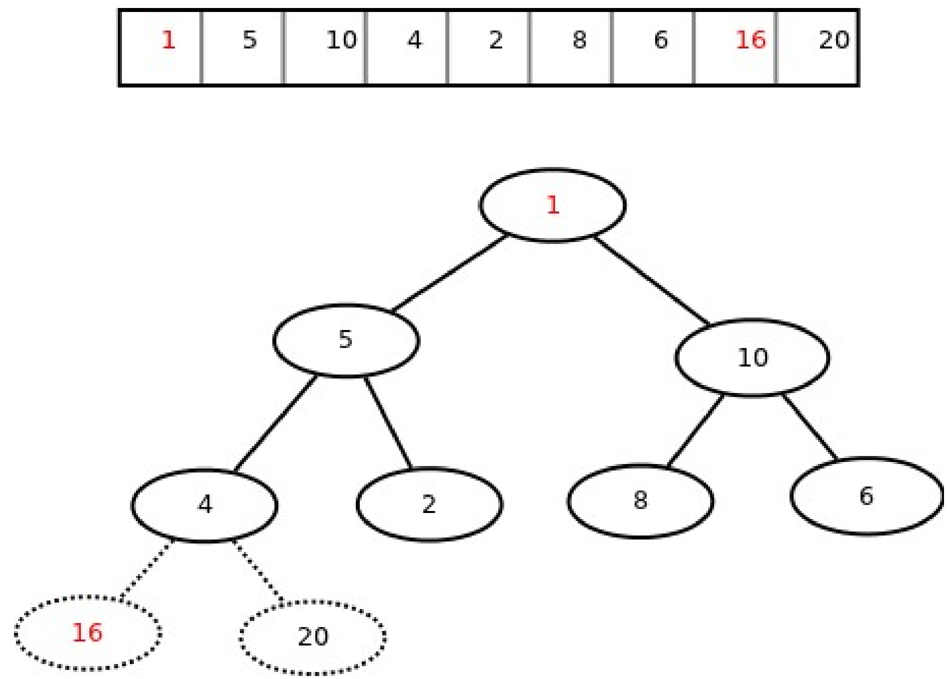
if largest ≠ i then:

swap A[i] ↔ A[largest]

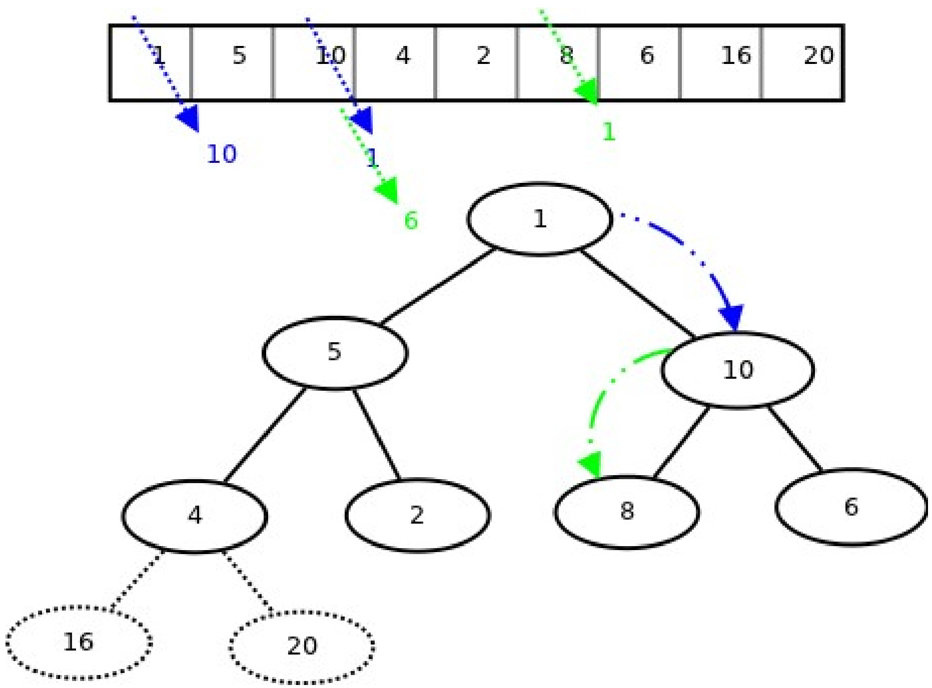
Max-Heapify(A, largest)

... tirando mais um...





Consertando de novo



Max-Heapify(A, i)

    left ← 2i

    right ← 2i + 1

    largest ← i

    if left ≤ heap_length[A] and A[left] > A[i] then:

        largest ← left

    if right ≤ heap_length[A] and

                A[right] > A[largest] then:

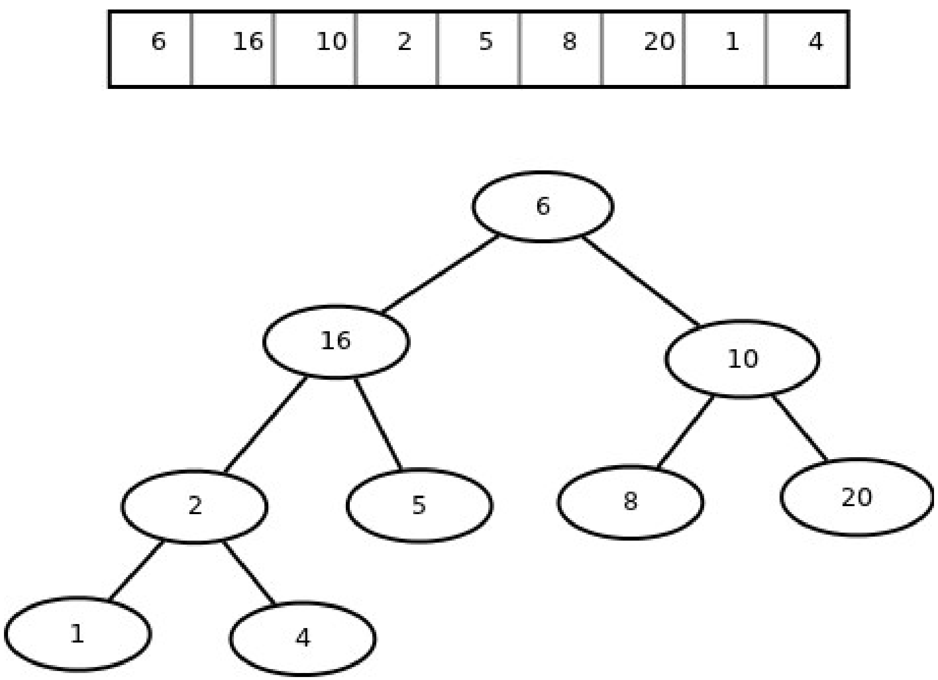
        largest ← right

    if largest ≠ i then:

        swap A[i] ↔ A[largest]

        Max-Heapify(A, largest)

A partir de um array qualquer...



- Dado o array representando uma árvore binária. A árvore/array é um heap?
- Como transformá-lo em um heap?

...aplicando Heapify do último para o primeiro elemento com filhos

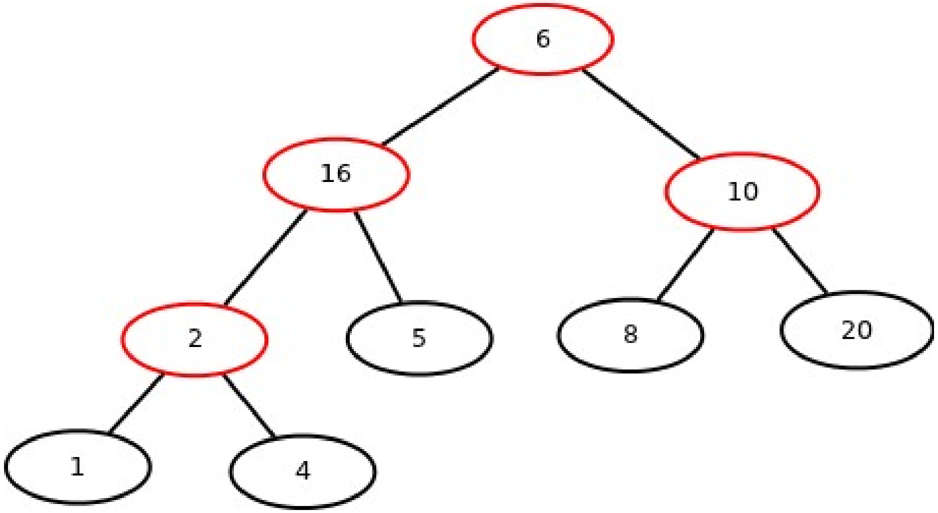
Build-Max-Heap(A)

heap_length[A] ← length[A]

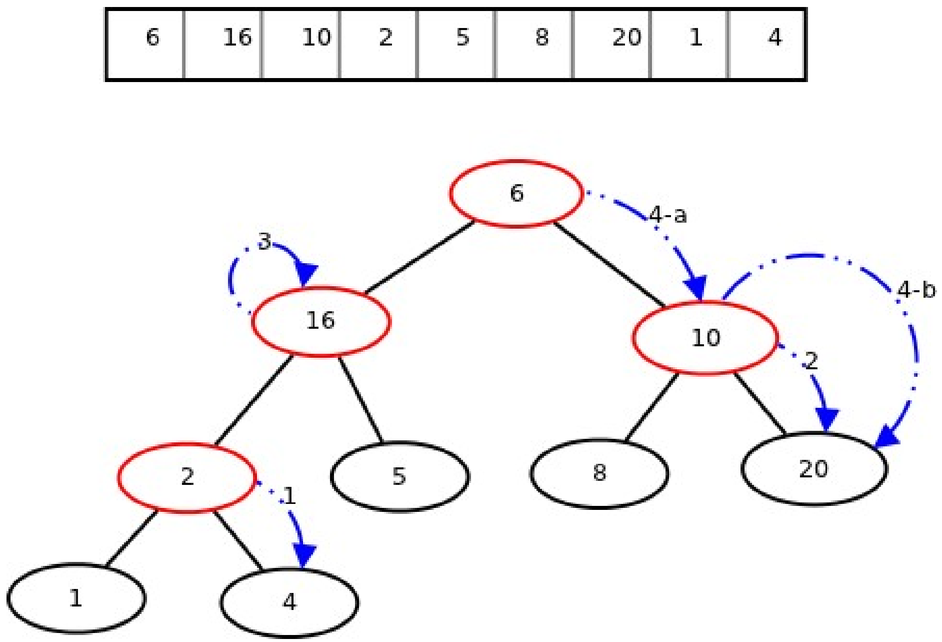
for i ← floor(length[A]/2) downto 1 do

Max-Heapify(A, i)

6	16	10	2	5	8	20	1	4



Constrói heap



Build-Max-Heap(A)

heap_length[A] ← length[A]

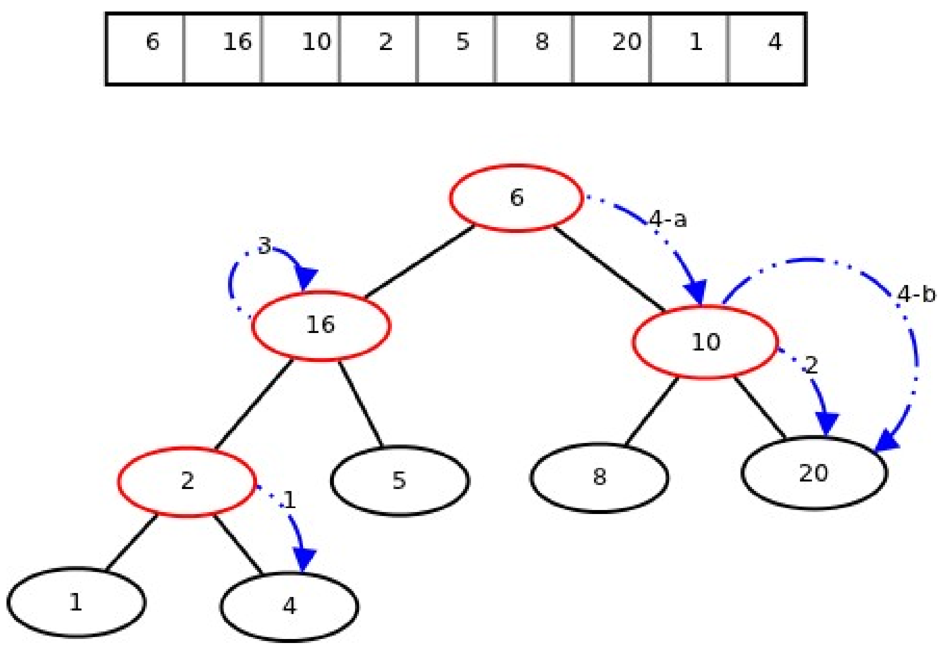
for i ← floor(length[A]/2) downto 1 do

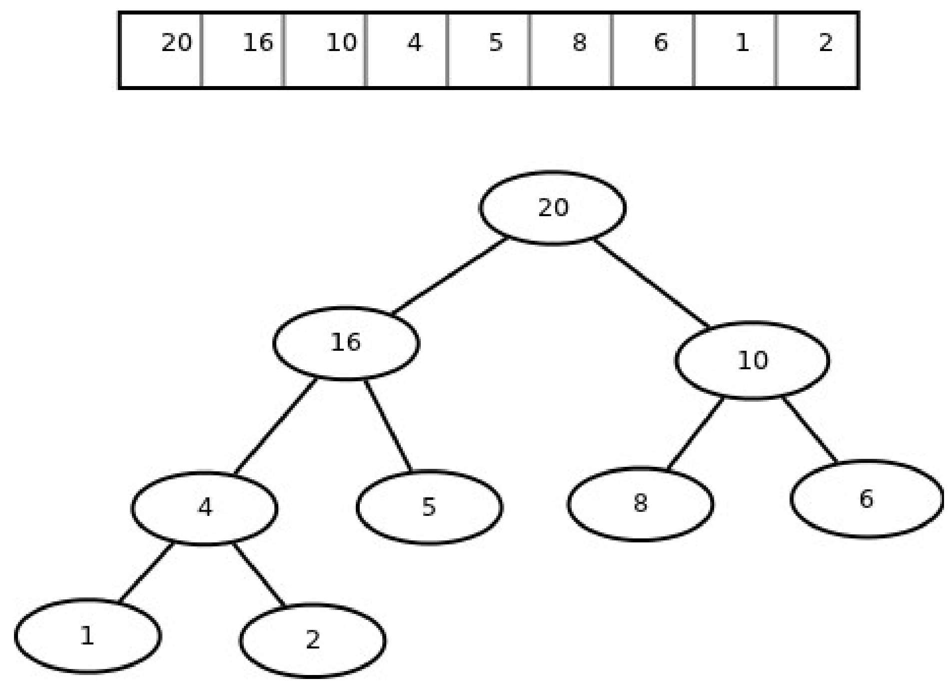
Max-Heapify(A, i)

Lembre que heapify, se necessário, desce até a folha – como no caso 4-(a,b).

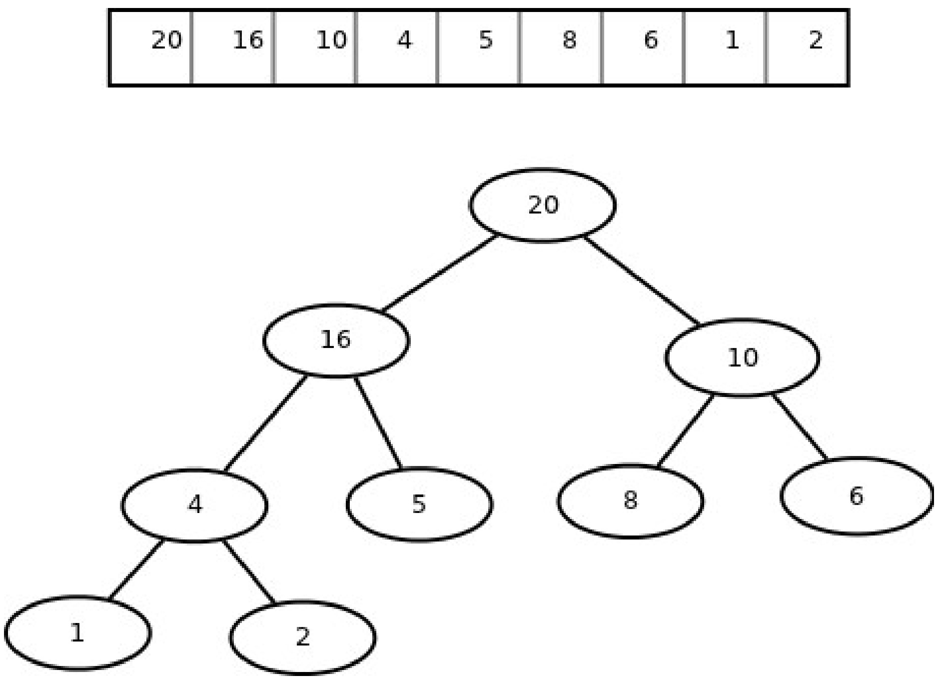
E COMO TERMINAR COM O ARRAY ORDENADO??

Exemplo


6 16 10 2 5 8 20 1 4
6
4-a
3
16
10
4-b
2
2
5
8
20
1
1
4


20 16 10 4 5 8 6 1 2
20
16
10
4
5
8
6
1
2

Ordenando o array



Heapsort (A)

BuildHeap(A)

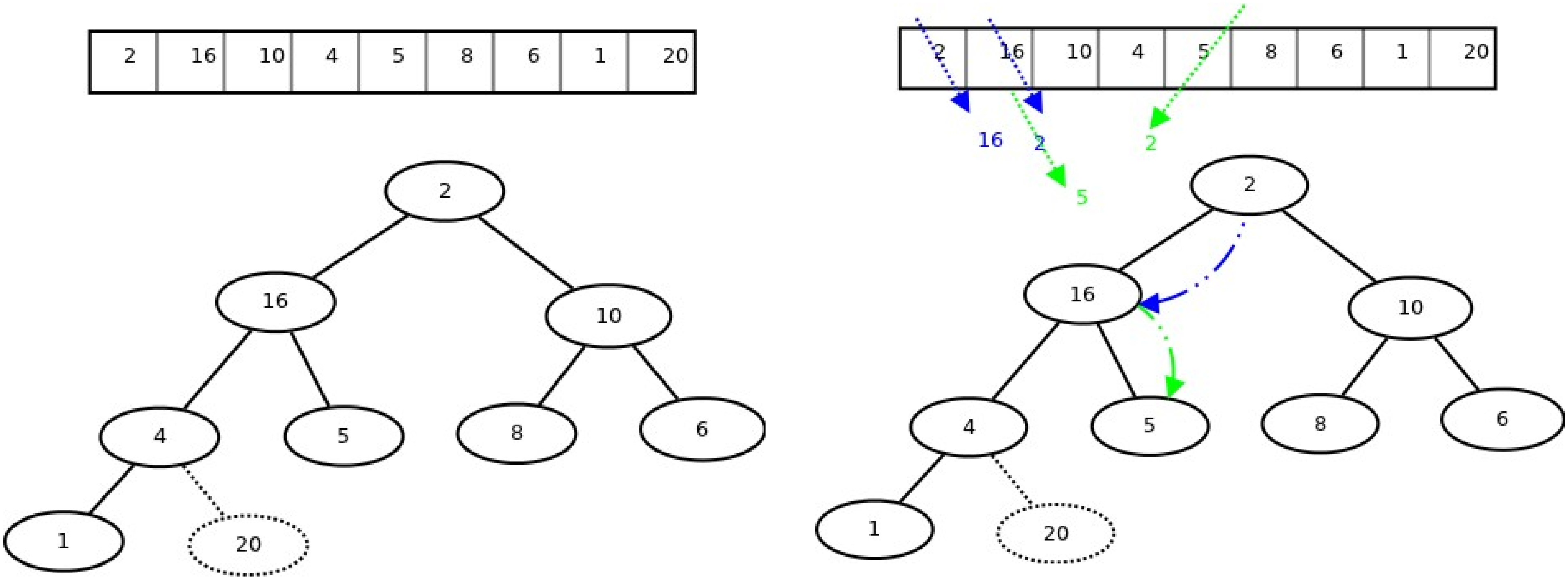
for i ← length[A] downto 2 do

Swap (A,1,i)

heapsize(A) ← heapsize(A)-1

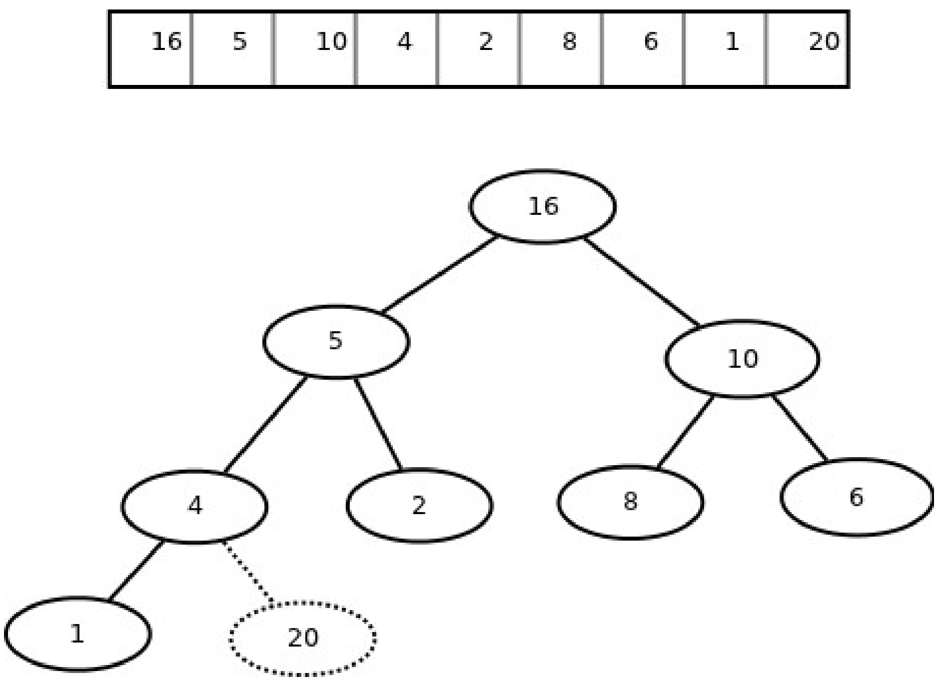
Max-Heapify(A, 1)

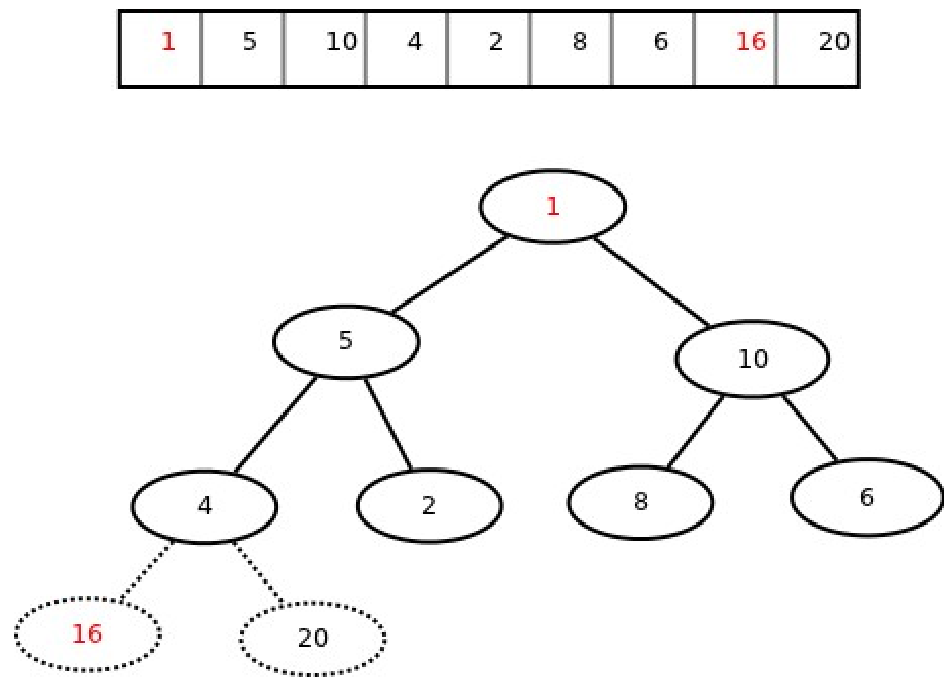
Tirando um...

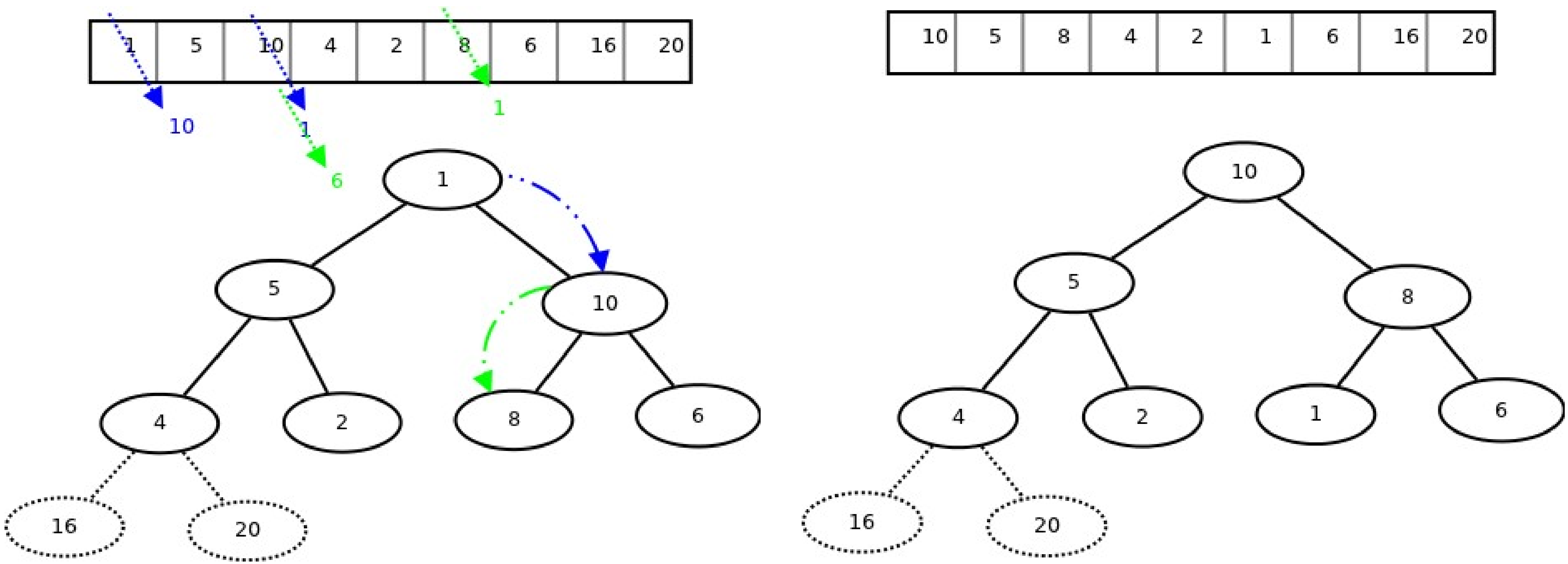


2 16 10 4 5 8 6 1 20
2
16
10
4
5
8
6
1
20
2 16 10 4 5 8 6 1 20
16
2
2
5
2
16
10
4
5
8
6
1
20

… tirando mais um...

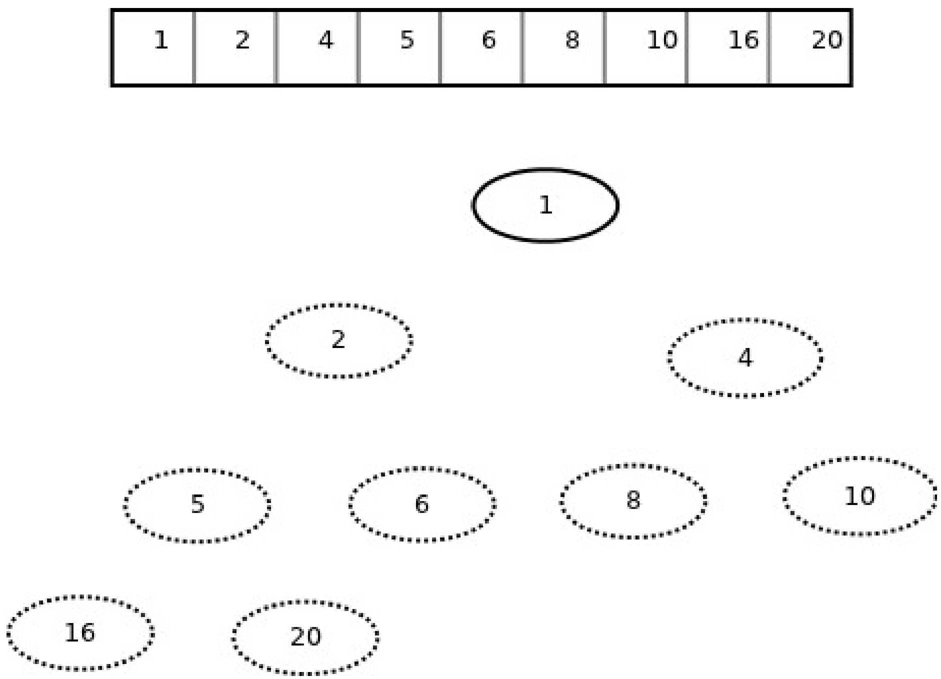






1 5 10 4 2 8 6 16 20
10
1
1
6
1
5
10
4
2
8
6
16
20
10 5 8 4 2 1 6 16 20
10
5
8
4
2
1
6
16
20

… e assim até que o heap contenha um único elemento



Sumário da semana

- Mergesort
  - Gerado utilizando a estratégia de Divisão e Conquista
  - Merge: intercala arrays ordenados
  - Merge-sort: divide até obter n arrays unitários e retorna combinado os arrays dois a dois (no caso de duas vias)
- T(n)=2*T(n/2)+c1*n+c2 – caso 2 do TM
- Em três vias a recorrência é T(n)=3*T(n/3)+c1*n+c2 – continua no caso 2.

Sumário da semana

- Heapsort
- Baseado em seleção sobre uma estrutura "esperta" (heap)
- Heapify: restaura a propriedade do heap quando um elemento está fora do lugar.
- BuildHeap: a partir de um array qualquer constrói um heap
- Heapsort: constrói heap, e enquanto houver elementos no heap, retira o elemento na raiz e restaura a propriedade

- $T(n)=T_{build}(n)+(n-1)*T_{heapify}(n-1)$
- $T_{heapify}(n)=O(lg(n))$
- $T_{build}(n)=O(n*lg(n))$ (assintoticamente folgado)
- ou
- $T_{build}(n)=O(n)$ (assintoticamente justo)
- Logo:
- $T(n)=O(n*lg(n))$
-