Universidade Federal da Paraíba
Centro de Informática

Heap Binário e Heapsort

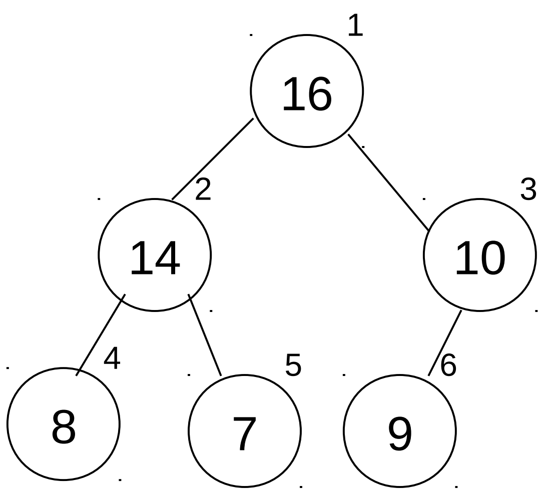
Prof. Gilberto Farias de Sousa

Roteiro

- Heap Binário
- Heap Máximo e Mínimo
- Procedimento MAX-HEAPIFY
- Construção de um Heap
- Algoritmo Heapsort
- Filas de Prioridade Máxima

Heap Binário

Heap Binário é um arranjo que pode ser visto como uma árvore binária praticamente completa.



Elementos representados como árvores

16	14	10	8	7	9
1	2	3	4	5	6

Arranjo de Elementos

Heap Binário

Um Heap A é um objeto com dois atributos:

– **comprimento[A]** : que é o número possível de elementos no arranjo;

– **tamanho_heap[A]** : o número de elementos no heap.

onde,

tamanho_heap[A] ≤ comprimento[A]

Heap Binário

- A raiz de um heap binário é A[1], o primeiro elemento.
- Dado um elemento **i** do heap temos:
  - PAI(i) = i/2;
  - ESQUERDO(i) = 2i;
  - DIREITO(i) = 2i + 1;

Heap Máximo e Mínimo

Em um Heap Máximo, todo nó **i** diferente da raiz tem:

$$\mathbf{A[PAI(i)] \geq A[i]}$$

Ou seja, o valor de um nó **i** é no máximo o valor de seu pai, logo, o maior elemento de um heap máximo está na raiz.

O heap mínimo é organizado de forma oposta:

$$\mathbf{A[PAI(i)] \leq A[i]}$$

Exercício

- A seqüência <23, 17, 14, 6, 13, 10, 15, 7, 12> é um heap máximo?

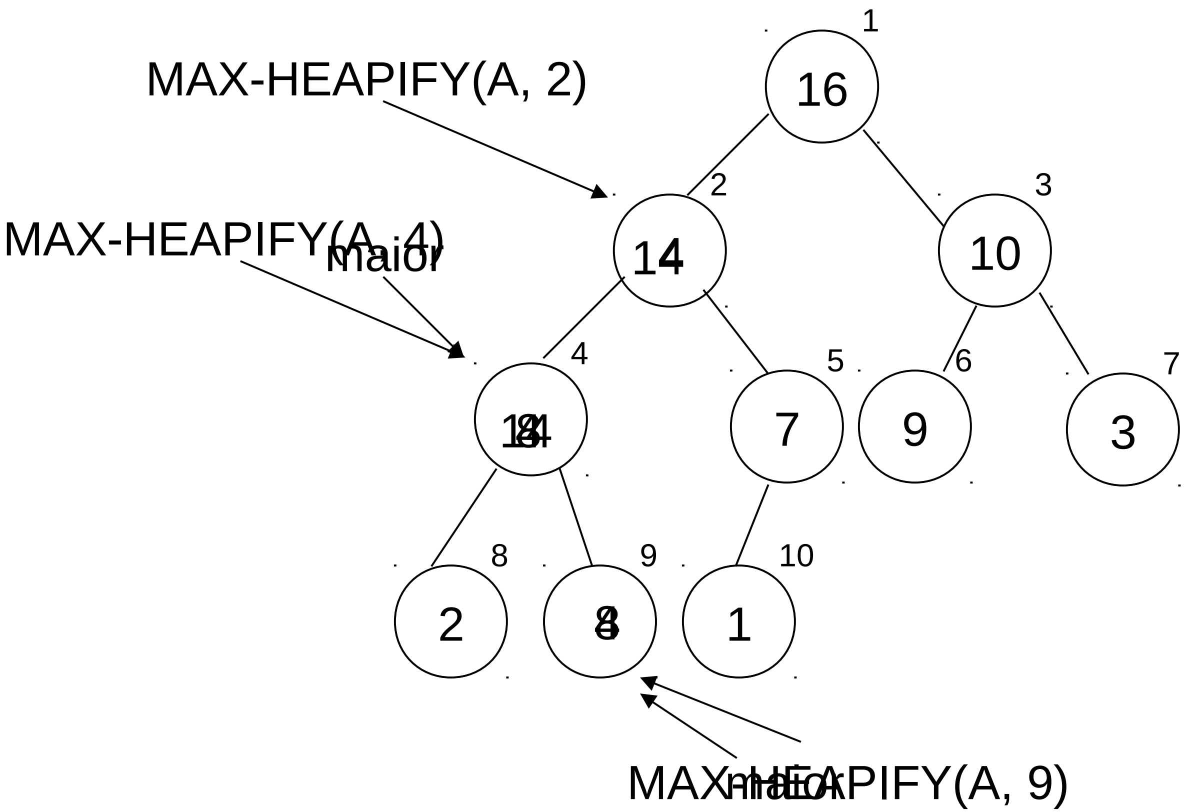
Procedimento MAX-HEAPIFY

- É uma subrotina que mantém a propriedade de heap máximo.
- Quando MAX-HEAPIFY é chamado, supomos que:
  - As árvores binárias com raízes ESQUERDO(i) e DIREITO(i) são heaps máximos
  - A[i] pode ser menor que seus filhos
- Logo, MAX-HEAPIFY tem a função de deixar A[i] “flutuar para baixo” no heap máximo, até que todo o arranjo se torne um heap máximo.

Procedimento MAX-HEAPIFY

```
MAX-HEAPIFY( A,  i)
        l ← ESQUERDO(i);
        r ← DIREITO(i);
        if(l ≤ tamanho_heap[A]) e (A[l] > A[i])
                maior ← l;
        else
                maior ← i;
        if(r ≤ tamanho_heap[A]) e (A[r] > A[maior])
                maior ← r;
        if (maior ≠ i)
                trocar A[i] ↔ A[maior]
                MAX-HEAPIFY( A, maior)
```

Procedimento MAX-HEAPIFY



Exercício

- Ilustre a operação MAX_HEAPIFY(A, 3) sobre o arranjo:

  A = <27, 17, 3, 16, 13, 10, 15, 7 , 12, 4, 8, 9, 0>

Construção de um Heap

- Os elementos do subarranjo **A[(⌊n/2⌋+1) .. n)]** , sendo n = tamanho_heap[A], são todos folhas da árvore e então cada um deles é um heap máximo de 1 elemento.
- Para construir um heap máximo a partir de um arranjo A, o procedimento BUILD-MAX-HEAP percorre os nós restantes da árvore e executa MAX-HEAPIFY sobre cada um de baixo para cima.

Construção de um Heap

BUILD-MAX-HEAP(A)

tamanho_heap[A] ← comprimento[A];

for (i ← ⌊comprimento[A]/2⌋ ***downto 1****)*

MAX-HEAPIFY(A, i);

A eficiência do algoritmo BUILD-MAX-HEAP é O(n).

Construção de um Heap

Dado o arranjo abaixo, construa o heap máximo usando o BUILD-MAX-HEAP

A	4	1	3	2	16	9	10	14	8	7
	1	2	3	4	5	6	7	8	9	10

Algoritmo HeapSort

- O algoritmo HeapSort começa usando BUILD-MAX-HEAP para construir um heap no arranjo de entrada A[1 .. n].
- Tendo em vista que o elemento máximo do arranjo está na raiz A[i], ele pode ser colocado em sua posição final correta, trocando-se esse elemento por A[n].
- Se agora diminuirmos o valor de tamanho_heap[A] podemos transformar o arranjo A novamente em um heap máximo chamado MAX_HEAPIFY(A, 1) .
- O algoritmo HeapSort repete este processo até que o arranjo tenha apenas um elemento.

Algoritmo HeapSort

HeapSort (A)

  BUILD-MAX-HEAP(A)

  for(i ← comprimento[A]; i > 1; --i)

    trocar A[1] ↔ A[i];

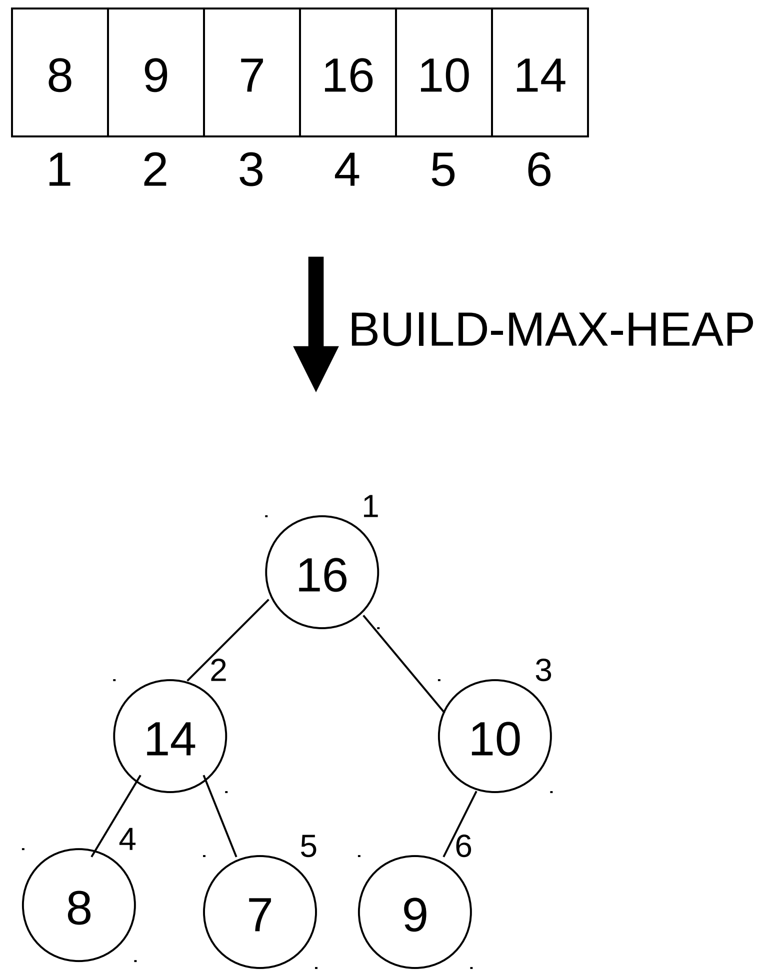
    tamanho_heap[a] -= 1;

    MAX-HEAPIFY(A,1);

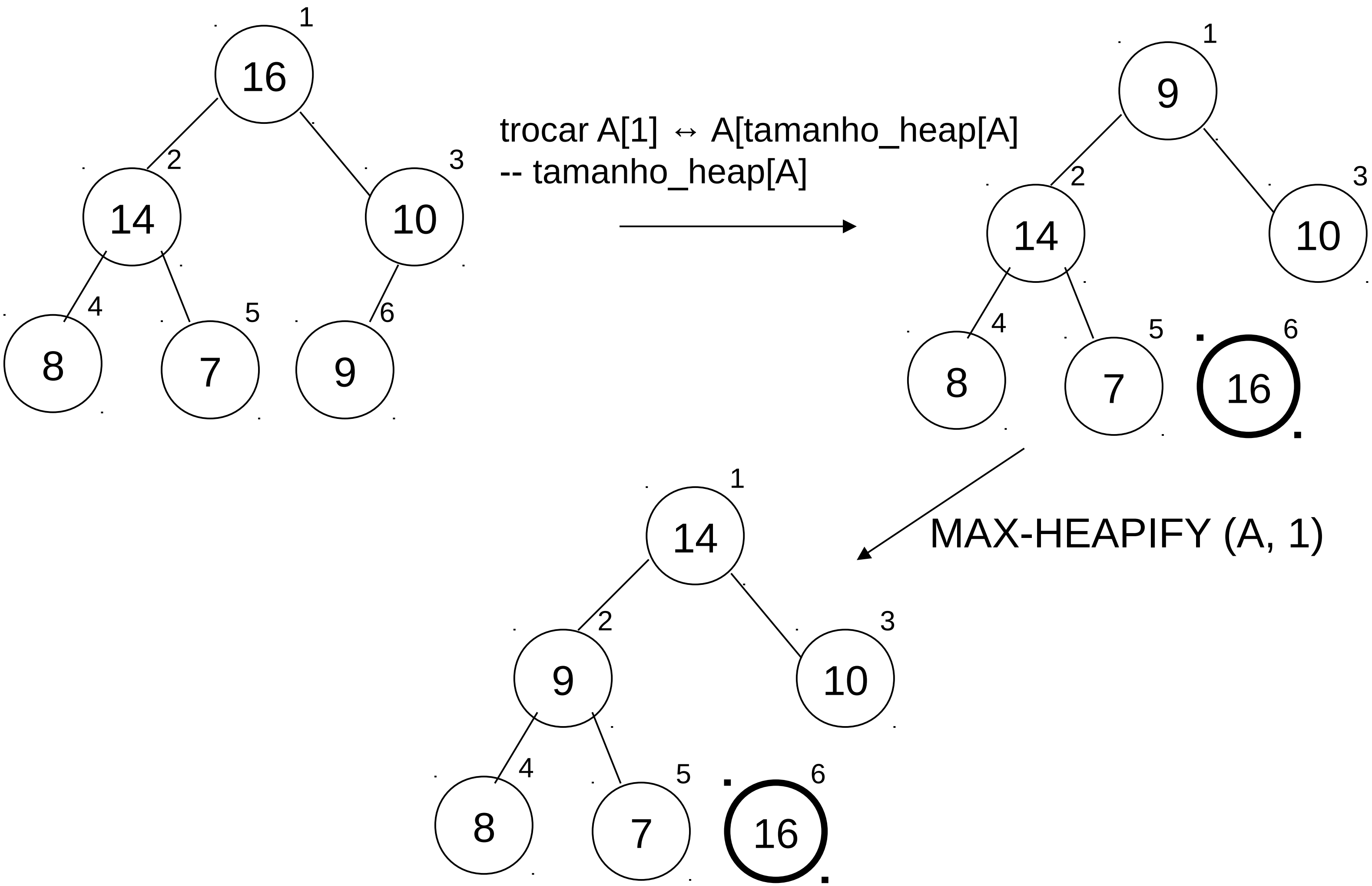
Algoritmo HeapSort

O procedimento HeapSort demora o tempo O(n*logn), pois a chamada a BUILD-MAX-HEAP demora o tempo O(n), e cada uma das n-1 chamadas MAX-HEAPIFY demora o tempo O(logn).

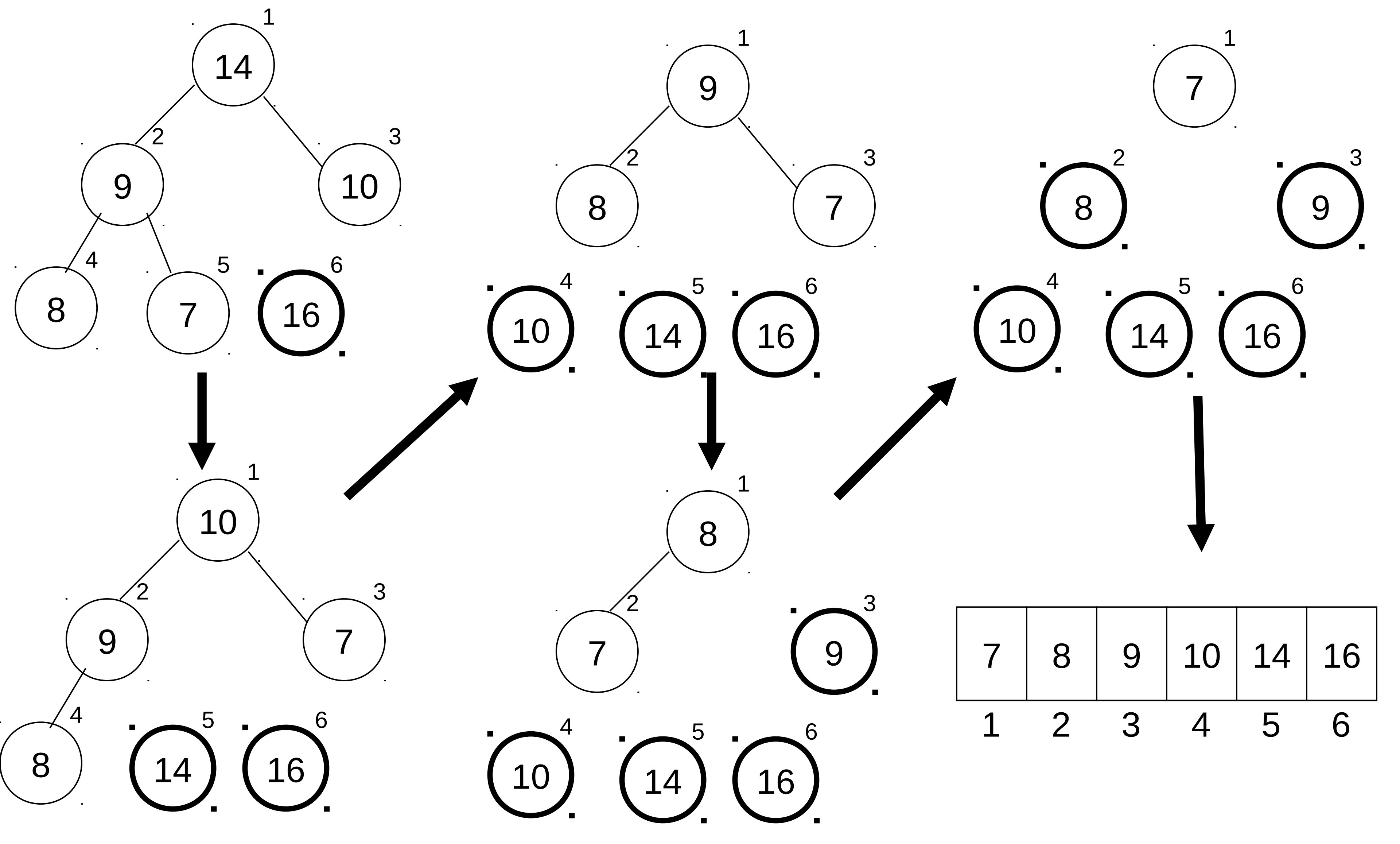
Algoritmo HeapSort



Algoritmo HeapSort



Algoritmo HeapSort



Exercício

Ilustre os movimentos que o algoritmo HeapSort realiza sobre o arranjo abaixo:

A = <3, 5, 2, 7, 13, 1, 15, 10, 4, 8>

Filas de Prioridade Máxima

- Como controlar o acesso a um recurso compartilhado por diversos processos ao mesmo tempo?
  - O uso de filas é recomendado: fila de impressão, fila de processamento e etc.
- E se houver a necessidade de priorizarmos o acesso para algumas tarefas?
  - Para isso, existe o conceito de Filas de Prioridade Máxima.
- Quando um trabalho termina ou é interrompido, o trabalho de prioridade mais alta é selecionado dentre os trabalhos pendentes.
  - E é exatamente um heap (máximo) a ser usado para implementar um fila de prioridade máxima.

Funções de uma Fila de Prioridade Máxima

Para que um heap máximo se torne uma Fila de Prioridade Máxima, o mesmo deve implementar as seguintes funções:

- **HEAP-MAXIMUN(A)**

  retorna o elemento de A com maior valor
- **MAX-HEAP-INSERT(A, x)**

  insere o elemento **x** no conjunto A.
- **HEAP-EXTRACT-MAX(A)**

  remove e retorna o elemento A de maior chave
- **HEAP-INCREASE-KEY(A, i, x)**

  aumenta o valor da chave do elemento **x** para o novo valor e que se presume ser maior que o valor atual de **x**.

Função HEAP-MAXIMUM

HEAP-MAXIMUM(A)

return A[1]

Este procedimento é executado no tempo O(1)

Função HEAP-EXTRACT-MAX

- Armazena o valor do elemento de maior chave;
- Troca o elemento de maior valor com o último elemento do heap;
- Decrementa o tamanho do heap;
- Chama o procedimento MAX-HEAPIFY para o primeiro elemento do heap, afim de reestabelecer o heap máximo.

Função HEAP-EXTRACT-MAX

```
HEAP-EXTRACT-MAX(A)
        if(tamanho_heap[A] < 1)
                return “heap underflow”;
        max ← A[1];
        A[1] ← A[tamanho_heap[A]];
        -- tamanho_heap[A];
        MAX-HEAPIFY(A, 1);
        return max;
```

Devido chamada a MAX-HEAPIFY o tempo de execução de HEAP-EXTRACT-MAX é O(log n)

Função HEAP-INCREASE-KEY

```
HEAP-INCREASE-KEY(A, i, chave)
    if(chave < A[i])
        return "nada a ser feito";
    A[i] ← chave;
    while (i > 1) e (A[PAI(i)] < A[i])
        trocar A[i] ↔ A[PAI(i)];
        i ← PAI(i);
```

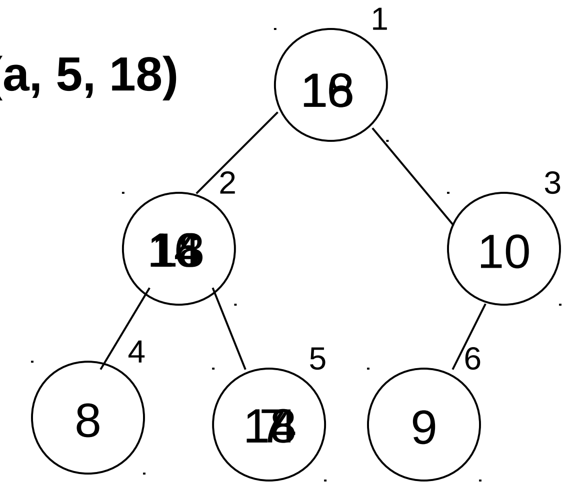
O tempo de execução sobre um heap de n elementos é O(log n). Que é a altura máxima de subida por um dos ramos da árvore.

Função HEAP-INCREASE-KEY

HEAP-INCREASE-KEY(a, 5, 18)

FIM!

A



Função MAX-HEAP-INSERT

- O procedimento expande o heap máximo;
- Adiciona à árvore uma nova folha cuja chave tem valor $-\infty$ (infinito negativo);
- Chama HEAP-INCREASE-KEY para garantir a propriedade de heap máximo.

Função MAX-HEAP-INSERT

MAX-HEAP-INSERT(A, chave)

++ *tamanho_heap[A];*

A[tamanho_heap[A]] $\leftarrow$ *-∞;*

HEAP-INCREASE_KEY (A, tamanho_heap[A], chave);

O tempo de execução de MAX-HEAP-INSERT sobre um heap de n elementos é O (log n).