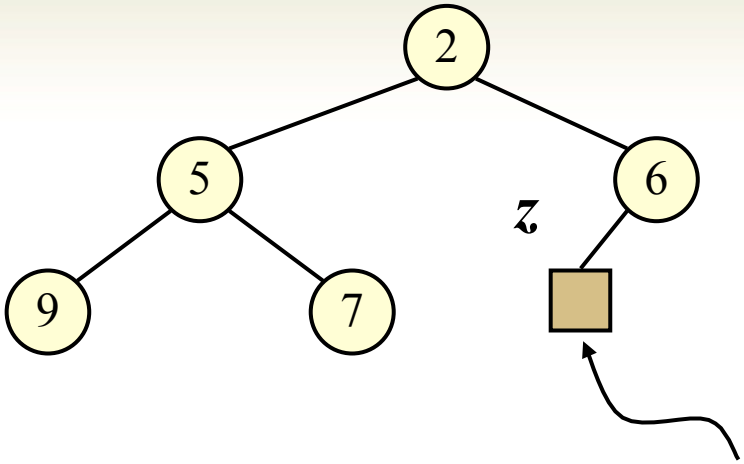
Ordenação: HeapSort

Prof. Túlio Toffolo
http://www.toffolo.com.br

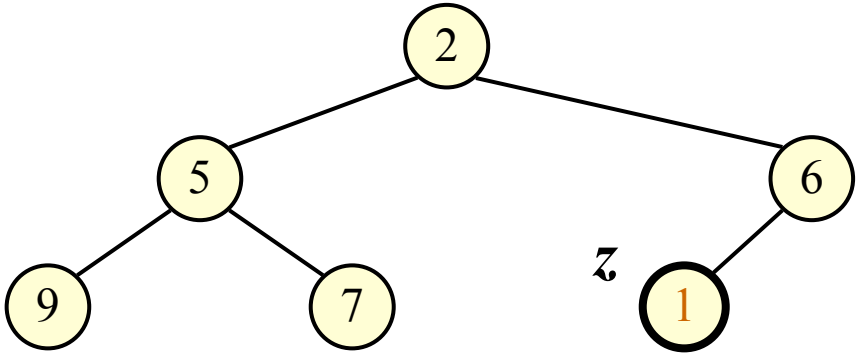
BCC202 – Aula 17
Algoritmos e Estruturas de Dados I

HEAP
OPERAÇÕES

Inserir um Nó no Heap



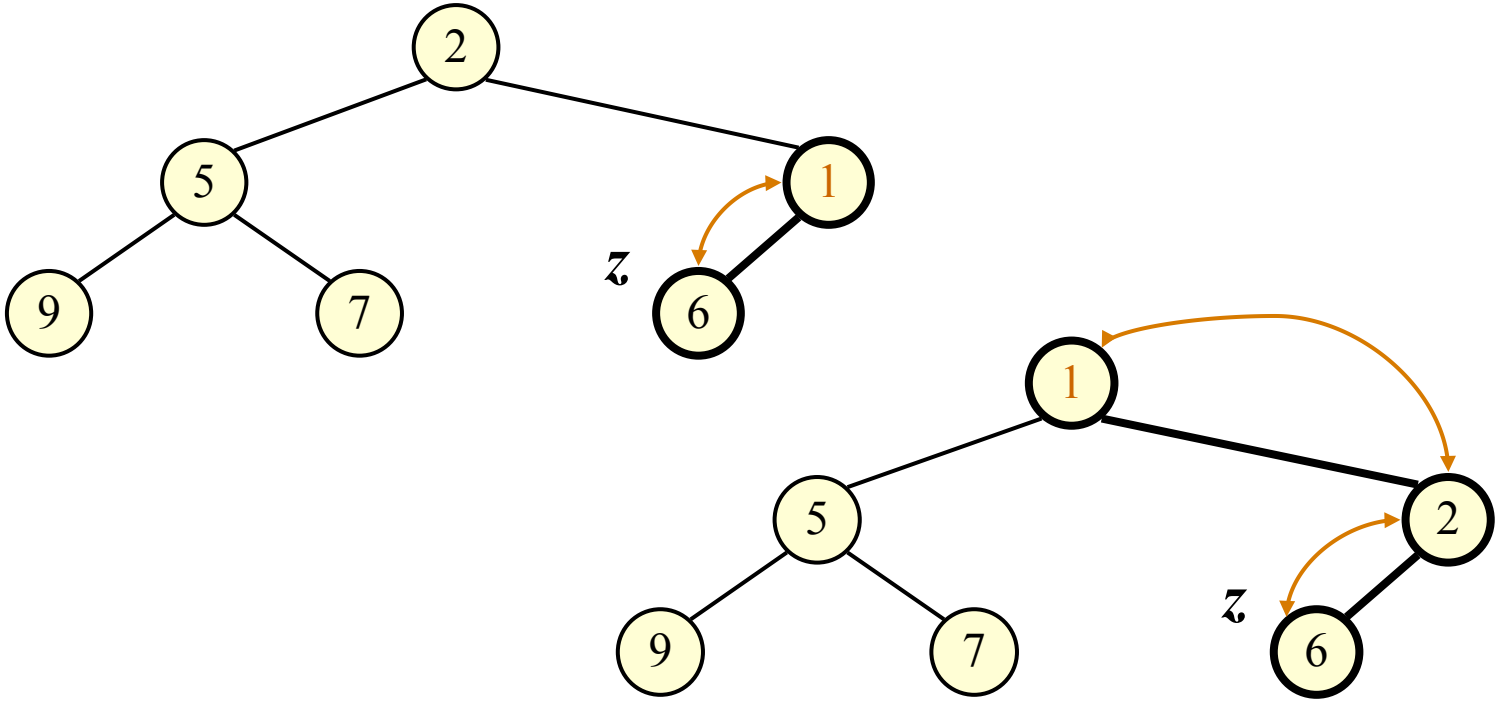
Nó sendo inserido



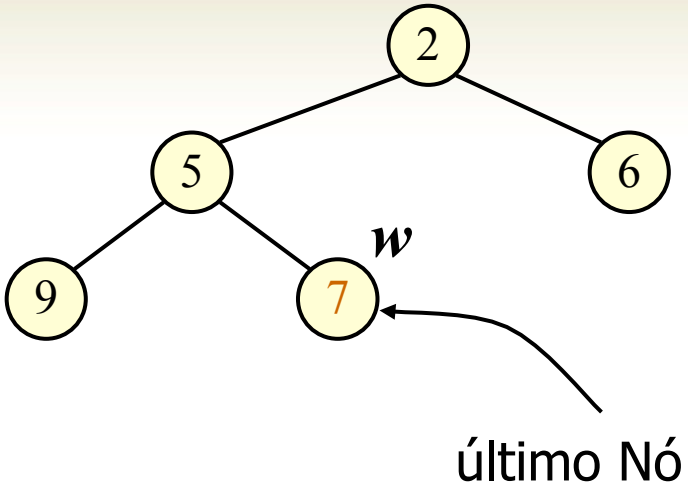
Comparar o nó inserido com os pais e trocar enquanto ele for menor que o pai ou até que ele seja o nó raiz

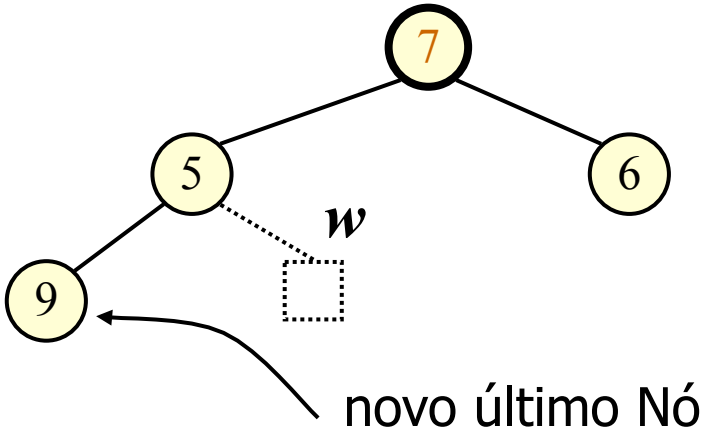
Inserir um Nó no Heap

- Na pior das hipóteses o custo de uma inserção será O(n log n), equivalente à altura da árvore



Remover um Nó do Heap

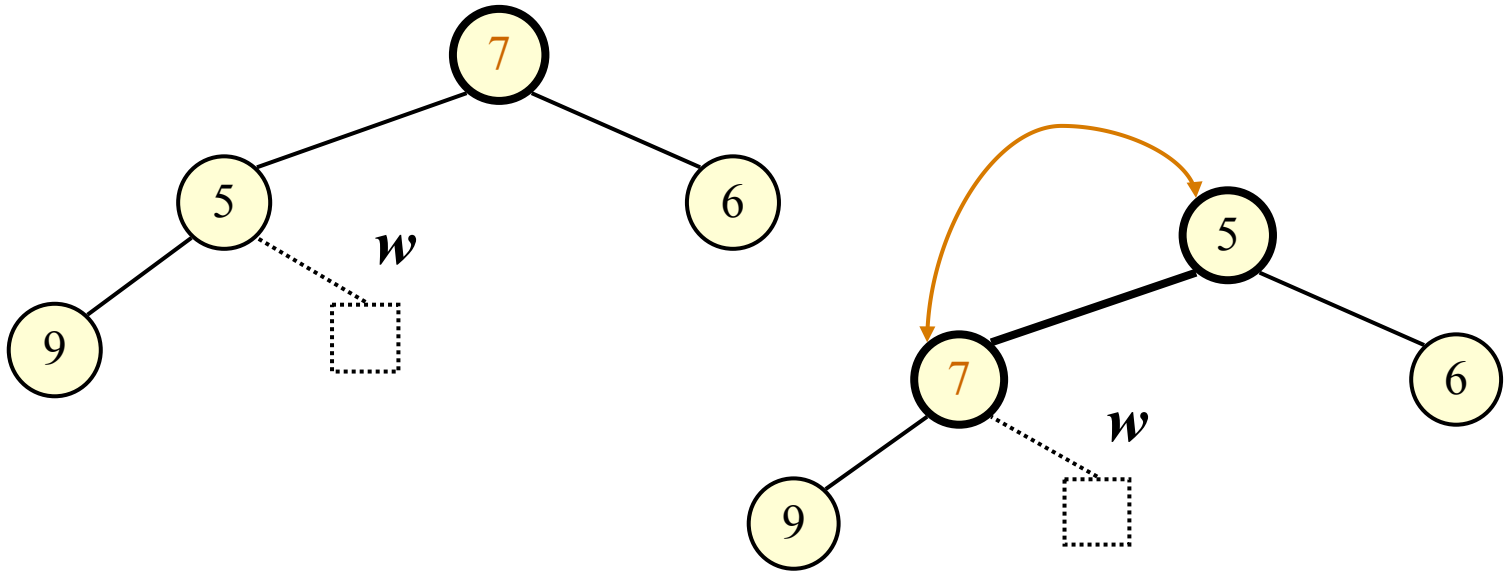




Trocar o nó raiz pelo último nó do heap e remover o último nó

Remover um Nó do Heap

- Refazer o heap!
- Na pior das hipóteses o custo de uma remoção será O(n log n), equivalente à altura da árvore



ORDENAÇÃO USANDO HEAP

HEAPSORT

Ordenação usando Filas de Prioridades

- As operações das filas de prioridades podem ser utilizadas para implementar algoritmos de ordenação.
- Basta utilizar repetidamente a operação Insere para construir a fila de prioridades.
- Em seguida, utilizar repetidamente a operação Retira para receber os itens na ordem correta.

Ordenação usando Filas de Prioridades

- O uso de listas lineares não ordenadas corresponde ao método **SelectSort**.
- O uso de listas lineares ordenadas corresponde ao método **InsertSort**.
- O uso de heaps corresponde ao método **HeapSort**.

Heapsort

- Para reduzir o número de operações, será utilizado um Heap em que o **maior** valor é armazenado na **raiz**
  - Teremos que o valor de um nó pai deverá ser sempre maior ou igual aos valores dos seus filhos

- Utilizaremos repetidamente a operação Insere para construir a fila de prioridades.
- Em seguida, utilizaremos repetidamente a operação Retira para receber os itens na ordem **inversa**.

HeapSort

- Algoritmo:
  1. Construir o *heap.*
  2. Troque o item na posição 1 do vetor (raiz do *heap) com o item da posição n-1.*
  3. Use o procedimento Refaz para reconstituir o *heap para os itens* A[0], A[1], ..., A[n - 2].
  4. Repita os passos 2 e 3 com os n - 1 itens restantes, depois com os n - 2, até que reste apenas um item.

HeapSort

- Exemplo de aplicação do Heapsort:

1	2	3	4	5	6	7
S	*R*	*O*	*E*	*N*	*A*	*D*
R	***N***	*O*	*E*	***D***	*A*	***S***
O	*N*	***A***	*E*	*D*	*R*	
N	***E***	*A*	***D***	*O*		
E	***D***	*A*	*N*			
D	***A***	*E*				
A	*D*					

HeapSort

- O caminho seguido pelo procedimento Refaz para reconstituir a condição do *heap está em* negrito.
- Por exemplo, após a troca dos itens S e D na segunda linha da Figura, o item D volta para a posição 5, após passar pelas posições 1 e 2.

HEAPSORT

IMPLEMENTAÇÃO

HeapSort

```
void heapRefaz(TItem *v, int esq, int dir) {
    int i = esq;
    int j = i*2 + 1; // j = primeiro filho de i

    TItem aux = v[i]; // aux = no i (pai de j)

    while (j <= dir) {
        if (j < dir && v[j].chave < v[j+1].chave)
            j++; // j recebe o outro filho de i
        if (aux.chave >= v[j].chave)
            break; // heap foi refeito corretamente
        v[i] = v[j];
        i = j;
        j = i*2 + 1; // j = primeiro filho de i
    }
    v[i] = aux;
}
```

HeapSort

```
void heapConstroi(TItem *v, int n) {
    int esq;
    esq = (n / 2) - 1; // esq = primeiro no folha do heap
    while (esq >= 0) {
        heapRefaz(v, esq, n-1);
        esq--;
    }
}
```

HeapSort


```
void heapSort(TItem *v, int n) {
    TItem aux;
    heapConstroi(v, n);
    while (n > 1) {
        aux = v[n-1];
        v[n-1] = v[0];
        v[0] = aux;
        n--;
        heapRefaz(v, 0, n-1); // refaz o heap
    }
}
```

MERGESORT

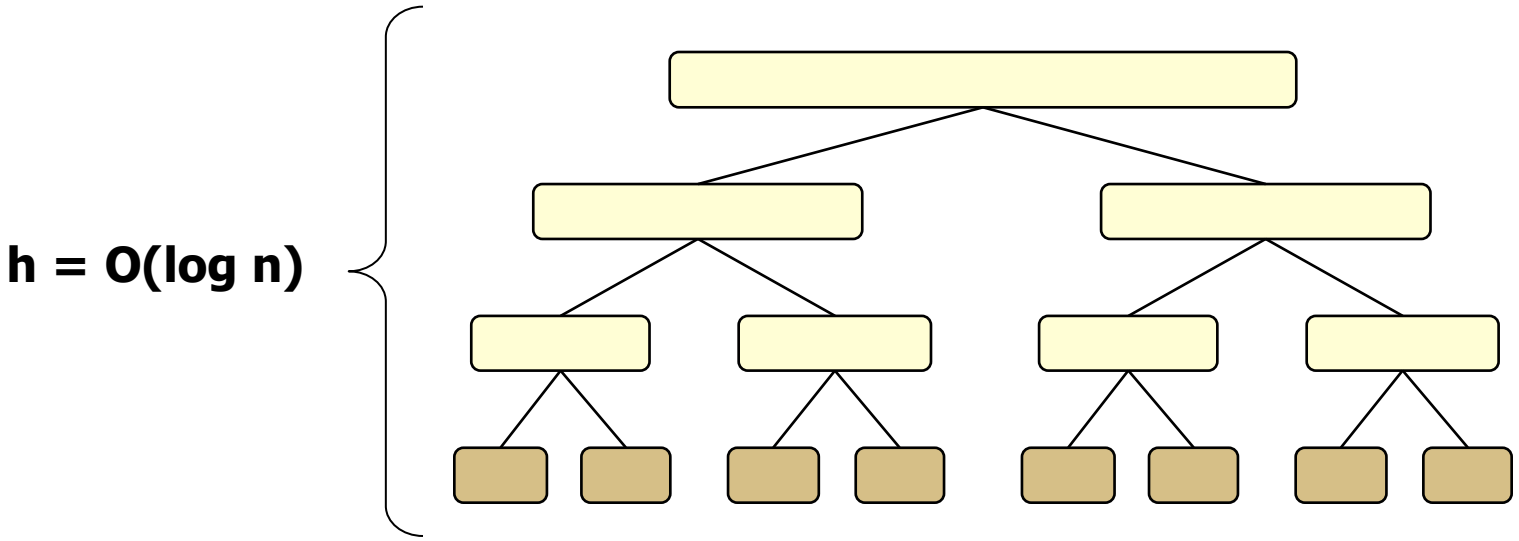
ANÁLISE DO ALGORITMO

Heapsort

- O procedimento Refaz gasta cerca de log n operações, no pior caso.
- Constroi – executa **O(n) x Refaz**
- Loop interno – executa **O(n) x Refaz**
- Logo, Heapsort gasta um tempo de execução **O(n log n)** no pior caso.

Análise do HeapSort

- A altura **h** da árvore de execução é **O(log n)**
- Refazer o heap, na pior das hipóteses, tem custo igual à altura da árvore. Construir o heap custa O(n log n)
- Logo: algoritmo é **O(n log n)**



MERGESORT

VANTAGENS/DESVANTAGENS

Quando usar o Heapsort?

- O HeapSort é recomendado:
  - Para aplicações que não podem tolerar eventualmente um caso desfavorável.
  - Não é recomendado para arquivos com poucos registros, por causa do tempo necessário para construir o heap.

Heapsort

- Vantagens:
  - O comportamento do Heapsort é sempre O(n log n), qualquer que seja a entrada.
- Desvantagens:
  - O anel interno do algoritmo é bastante complexo se comparado com o do Quicksort.
  - O Heapsort não é estável.

Perguntas?

HEAPSORT

EXERCÍCIO

Exercício

- Dada a sequência de números:

3 4 9 2 5 1 8

Ordene em ordem crescente utilizando o algoritmo **HeapSort**, apresentado a sequência dos números e explicando cada passo do algoritmo.