

Engenharia de Computação

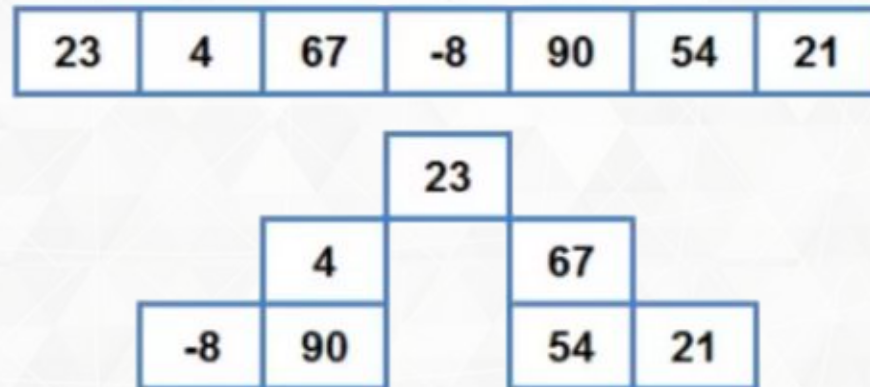
Pesquisa e Classificação de Dados

Aula 7 – HeapSort

Prof. Muriel de Souza Godoi
muriel@utfpr.edu.br

HeapSort

- Ordenação usando Heap (monte)
 - Heap: Vetor que simula uma árvore binária completa (exceção do último nível)
 - Todo elemento pai possui dois elementos como filhos
 - $\text{Pai}[i] \Rightarrow \text{FilhoEsq}: [2*i + 1] / \text{FilhoDir} [2*i + 2]$



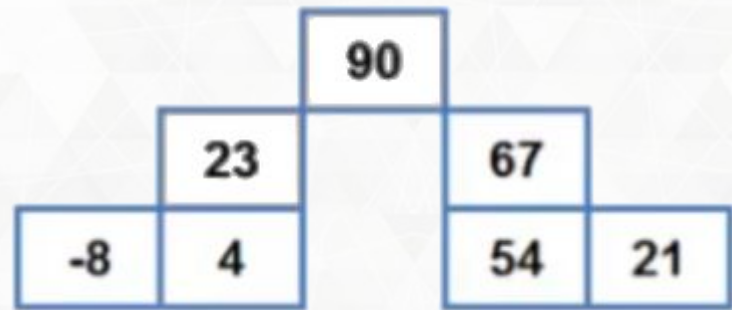
HeapSort - MaxHeap

- Um Heap é um MaxHeap se:
 - Todos os nós pais são maiores que seus nós filhos
 - **Consequência:** O maior elemento está na raiz da árvore

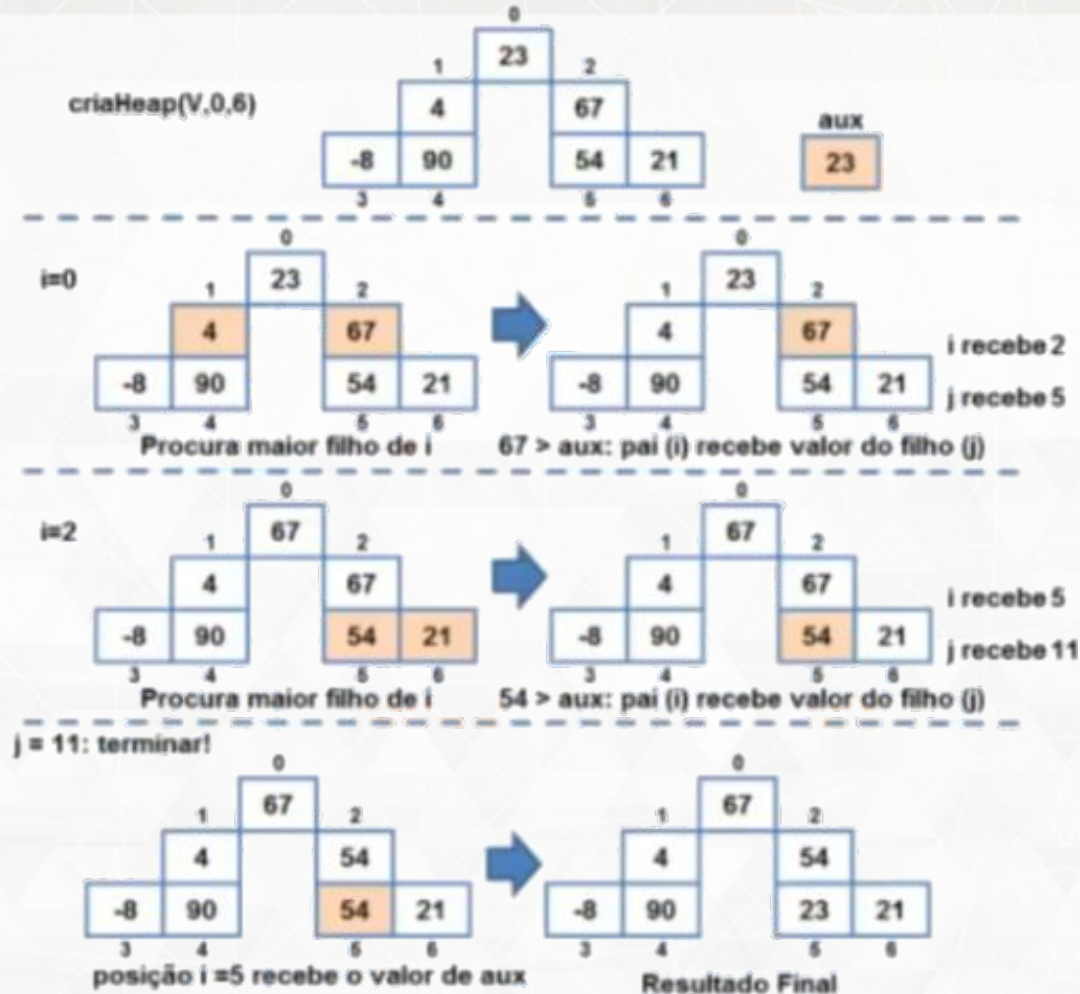
Heap



MaxHeap



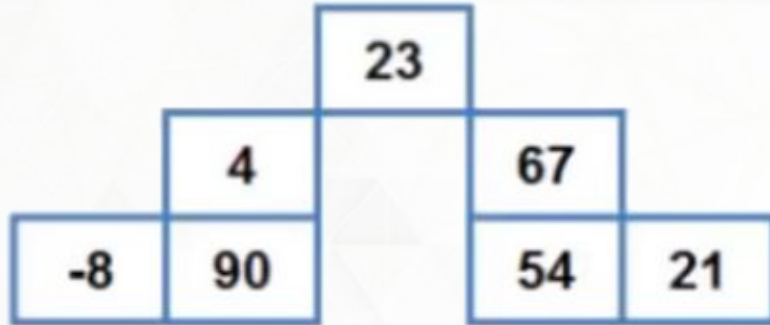
Heap Sort - criaHeap



Heap Sort - criaHeap

1º comando for: criaHeap(V,i,6)
Elemento Pai fica maior que os Filhos

i=3



i=2



i=1



i=0



HeapSort

```
44
45 void heapSort(int *vet, int N) {
46     int i, aux;
47     for(i=(N - 1)/2; i >= 0; i--) {
48         criaHeap(vet, i, N-1);
49     }
```

} Criar heap a partir dos dados

23	4	67	-8	90	54	21
----	---	----	----	----	----	----

1º comando for: criaHeap(V,i,6)
Elemento Pai fica maior que os Filhos

Heap Sort

- Como usar um Max Heap para ordenar?
 - **Passo 1:**
 - Transformar o heap em um max-heap
 - **Passo 2:**
 - Move raiz (maior valor) para o final(ultima posicao)
 - Reconstrói o heap do vetor restante

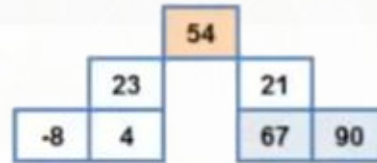
HeapSort

2º comando for: move maior elemento para o final e acha o maior do vetor restante

$i-1=5$ 67 23 54 -8 4 21 90



$i-1=4$ 54 23 21 -8 4 67 90



$i-1=3$ 23 4 21 -8 54 67 90



$i-1=2$ 21 4 -8 23 54 67 90



$i-1=1$ 4 -8 21 23 54 67 90



$i-1=0$ -8 4 21 23 54 67 90



Ordenado

HeapSort - Pseudocódigo

- O algoritmo usa **2 funções**
 - **heapSort**: divide os dados em vetores cada vez menores
 - **criaHeap**: transforma um **Heap** em um **MaxHeap**

QuickSort - Pseudocódigo

- **heapSort(v, inicio, fim)**
 - Se $\text{inicio} < \text{fim}$ então:
 - $\text{pivo} \leftarrow \text{particiona}(v, \text{inicio}, \text{fim})$
 - $\text{quickSort}(v, \text{inicio}, \text{pivo}-1)$
 - $\text{quickSort}(v, \text{pivo}+1, \text{fim})$

QuickSort - Pseudocódigo

- **criaHeap(v, inicio, fim)**
 - $esq \leftarrow inicio$
 - $dir \leftarrow fim$
 - $pivo \leftarrow v[inicio]$
 - **Enquanto** $esq < dir$ **faça:**
 - **Enquanto** $v[esq] \leq pivo$ **faça:**
 - incrementa esq
 - **Enquanto** $v[dir] > pivo$ **faça:**
 - decrementa esq
 - **Se** $esq < dir$ **então:**
 - troca $v[esq]$ e $v[dir]$
 - troca $v[dir]$ com $v[inicio]$

Heap Sort - Complexidade

- Considerando um vetor com n elementos, o tempo de execução é sempre:
 - $O(n \log n)$;
- Na prática, o HeapSort é mais lento que o QuickSort, exceto no pior caso.

Exercício

- Implemente o **HeapSort** em C considerando as seguintes assinaturas de função
 - **Função 1: heapSort**

```
/**  
 * \brief Ordena o vetor usando HeapSort  
 *  
 * \param v vetor a ser ordenado  
 * \param n tamanho do vetor  
 *  
 * Ordena o vetor usando o método HeapSort  
 */  
void heapSort(int *v, int n);
```



Exercício

• Função 2: **criaHeap**

```
/**  
 * \brief Transforma um heap em maxheap  
 *  
 * \param v vetor a ser particionado  
 * \param pai índice do nó raiz do heap  
 * \param fim índice do último elemento do heap  
 *  
 * ??????  
 */  
void criaHeap(int *v, int pai, int fim);
```