

Ordenação: Heapsort

Estruturas de Dados e Algoritmos – Ciência da Computação



Prof. Daniel Saad Nogueira
Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

1 Heapsort



Heapsort

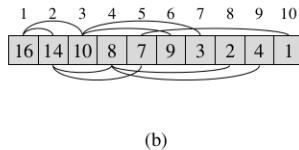
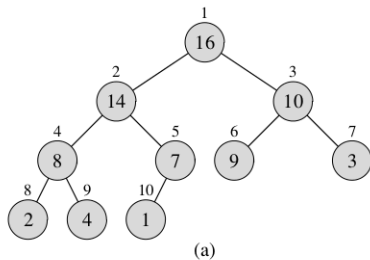
Heap

A chave do heapsort é uma estrutura denominada heap. Uma heap binária é uma estrutura de natureza recursiva e tem as seguintes propriedades:

- i) O elemento pai é \geq do que os seus filhos.
- ii) O filho da esquerda é uma heap.
- iii) O filho da direita também é uma heap.



Heapsort

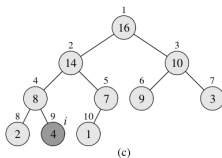
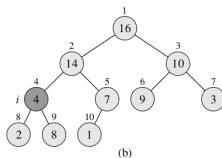
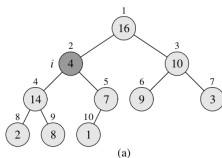




Heapsort

Heapify

Para construir uma Heap, devemos aplicar o procedimento de **heapify** nos nós que não apresentam a propriedade de Heap.

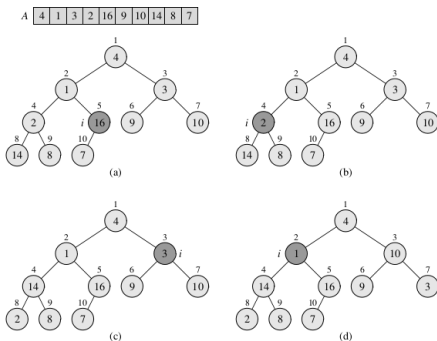




Heapsort

Heap

Note que os nós folha, já são heaps (por vacuidade). Logo, o **heapify** só necessita ser aplicado aos nós acima dos nós folhas.





Heapsort

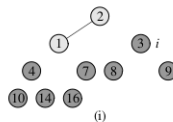
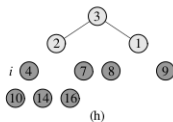
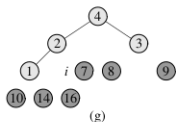
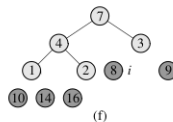
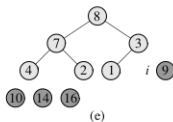
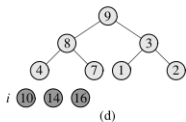
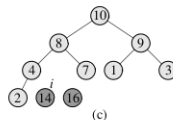
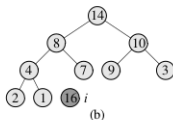
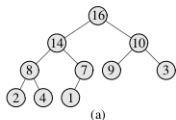
Heapsort

- Uma vez que a Heap está contruída, sabemos que o elemento raiz (primeiro elemento) é o maior de todos, logo podemos retirá-lo e colocá-lo no fim da sequência.
- Escolhemos o último nó folha para ser a raiz (primeiro elemento da sequência) e aplicamos **heapify** para manter a estrutura da heap.
- O procedimento é repetido até que tenhamos a sequência ordenada.



Heapsort

Exemplo





Heapsort

Function Heapsort

Input: V

Output: $V, \quad V[i] < V[i + 1], 0 \leq i < n - 1$

```
1 MAKEHEAP( $V$ )
2 for(  $i \leftarrow V.SIZE() - 1; i > 0; i --$  )
3   SWAP( $V[0], V[i]$ )
4   HEAPIFY( $V, 0, i$ )
```



Heapsort

Function MakeHeap

Input: V

Output: V , com propriedade de **Heap**

```
1 for(  $i \leftarrow V.SIZE()/2; i \geq 0; i--$  )  
2   | HEAPIFY( $V, i, V.size()$ )
```



Heapsort

Function Heapify

Input: $V, i, heapSize$

```
1  $l \leftarrow 2 \cdot i + 1$ 
2  $r \leftarrow 2 \cdot i + 2$ 
3  $largest \leftarrow i$ 
4 if(  $l < heapSize \wedge V[l] > V[i]$  )
5    $largest \leftarrow l$ 
6 if(  $r < heapSize \wedge V[r] > V[largest]$  )
7    $largest \leftarrow r$ 
8 if(  $largest \neq i$  )
9    $SWAP(V[i], V[largest])$ 
10   $HEAPIFY(V, largest, heapSize)$ 
```



Sumário

2 Análise



Heapsort

Análise

- Para construir a Heap, leva-se tempo $O(n \lg n)$, uma vez que é necessário manter a propriedade de Heap para todos os nós, e cada nó tem altura $O(\lg n)$.
- Apesar de ser um limite superior, uma análise mais detalhada mostra que a construção da Heap é feita em tempo $\Theta(n)$.
- Uma vez que a Heap é construída, a retira do nó raiz e a manutenção da propriedade da Heap levam tempo $\Theta(\lg n)$.
- Como esse procedimento é repetido para todos os nós, temos que o Heapsort leva tempo $\Theta(n \lg n)$.



Heapsort

In-place	Estável
✓	✗