Heapsort

Análise de Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília, Campus Taguatinga

Sumário

Sumário

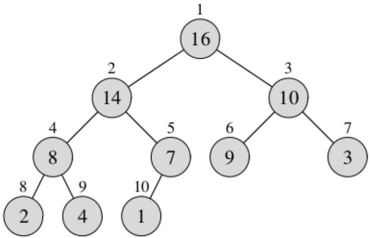
Heapsort

Heap

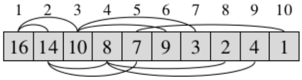
A chave do heapsort é uma estrutura denominada heap. Uma heap binária é uma estrutura de natureza recursiva e tem as seguinte propriedades:

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



(a)

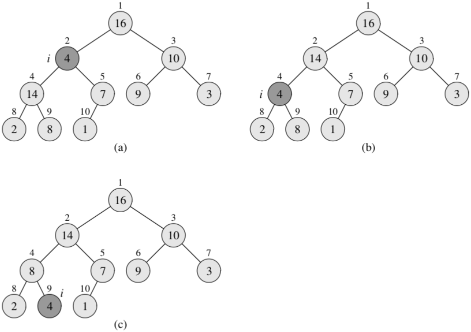


(b)

Heapsort

Heapify

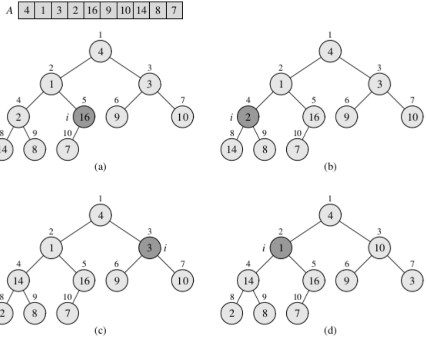
Para construir uma Heap, devemos aplicar o procedimento de **heapify** nos nós que não apresentam a propriedade de Heap.



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.



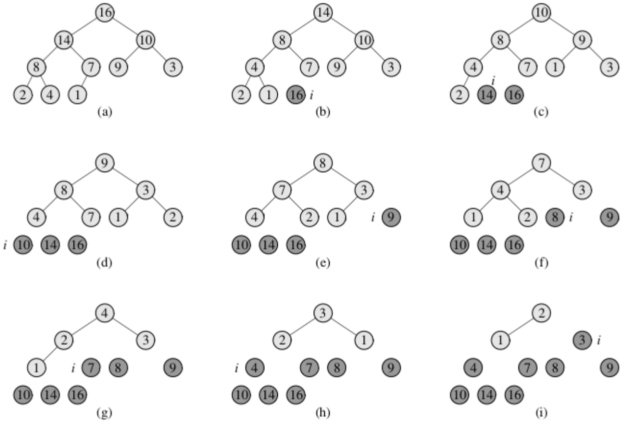
Heapsort

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

$\textsc{MakeHeap}(V)$

for($i \leftarrow V.\textsc{size}() - 1; i > 0; i--$)

 $\textsc{Swap}(V[0], V[i])$

 $\textsc{Heapify}(V, 0, i)$

Heapsort

Function MakeHeap

Input: V

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

for($i \leftarrow V.\textsc{size}()/2; i \geq 0; i--$)

$\quad \textsc{Heapify}(V, i, V.size())$

Heapsort

Function Heapify

Input: $V, i, heapSize$

$l \leftarrow 2 \cdot i + 1$

$r \leftarrow 2 \cdot i + 2$

$largest \leftarrow i$

if($l < heapSize \wedge V[l] > V[i]$)

  $largest \leftarrow l$

if($r < heapSize \wedge V[r] > V[largest]$)

  $largest \leftarrow r$

if($largest \neq i$)

  $\textsc{Swap}(V[i], V[largest])$

  $\textsc{Heapify}(V, largest, heapSize)$

Heapsort

Function Heapify

Input: $V, i, heapSize$

```
while i < heapSize do
    l ← 2 · i + 1
    r ← 2 · i + 2
    largest ← i
    if( l < heapSize ∧ V[l] > V[i] )
        largest ← l
    if( r < heapSize ∧ V[r] > V[largest] )
        largest ← r
    if( largest = i )
        break;
    Swap(V[i], V[largest])
    i ← largest
```

Sumário

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
✓	✗

Heapsort

Teorema

$\textsc{MakeHeap}(V)$ leva tempo $\Theta(n)$.

Heapsort

Demonstração

O procedimento $\textsc{Heapify}()$ quando chamado de um nó de altura h leva tempo $O(h)$.

Logo, $\textsc{MakeHeap}(\text{V})$ leva tempo:

$$
\begin{array}{lll}
\displaystyle\sum_{h=0}^{\lfloor \lg n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil O(h) & \in & \diamond \text{ cada nível de altura h tem essa quantidade de folhas} \\
O\left(n \displaystyle\sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) & \leq & \diamond \text{ Isola o termo } n \\
O\left(n \displaystyle\sum_{h=0}^{\infty} \frac{h}{2^h}\right) & = & \diamond \text{ Majoração} \\
O\left(n \dfrac{1/2}{(1-1/2)^2}\right) & = & \diamond \text{ Equivalência} \\
O\left(2n\right) & \in O(n) &
\end{array}
$$

□