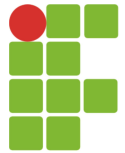
Heapsort

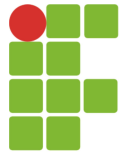
Análise de Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

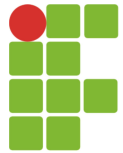
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



Sumário



Heapsort

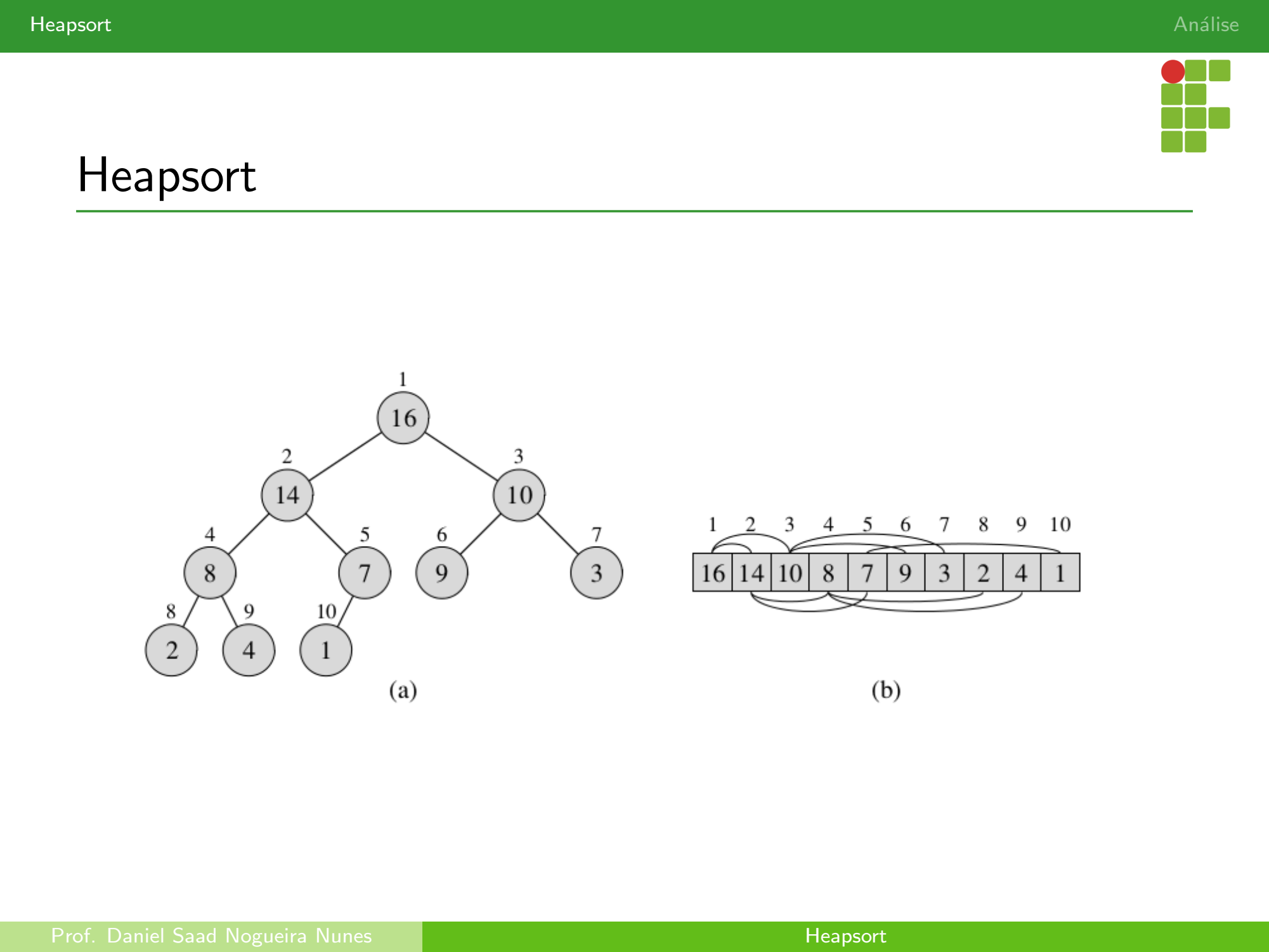
Heap

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.

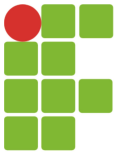
iii) O filho da direita também é uma heap.



Heapsort

(a)

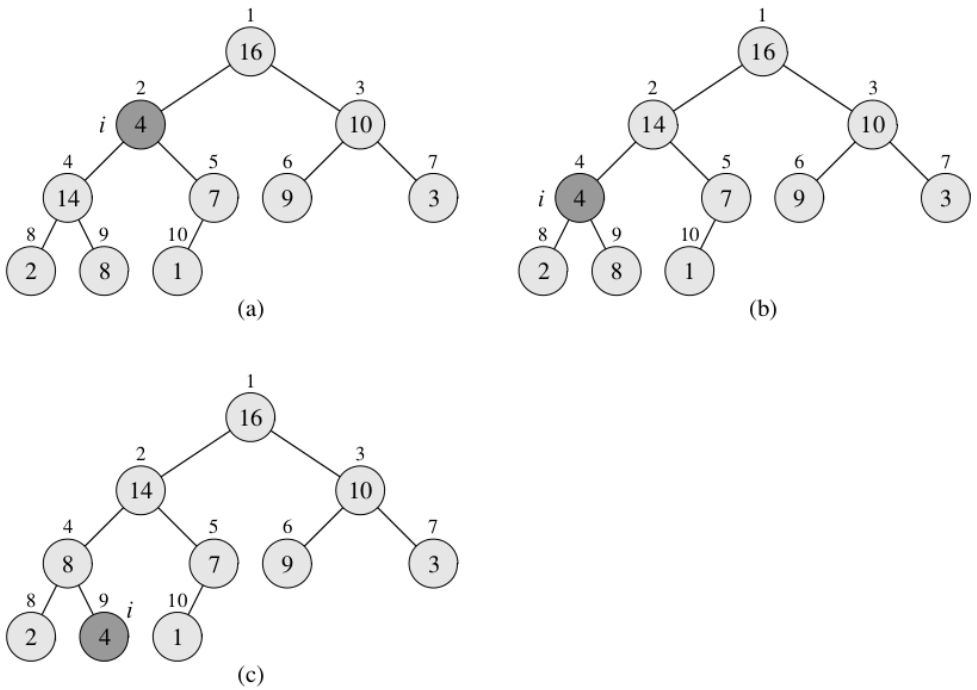
(b)



Heapsort

Heapify

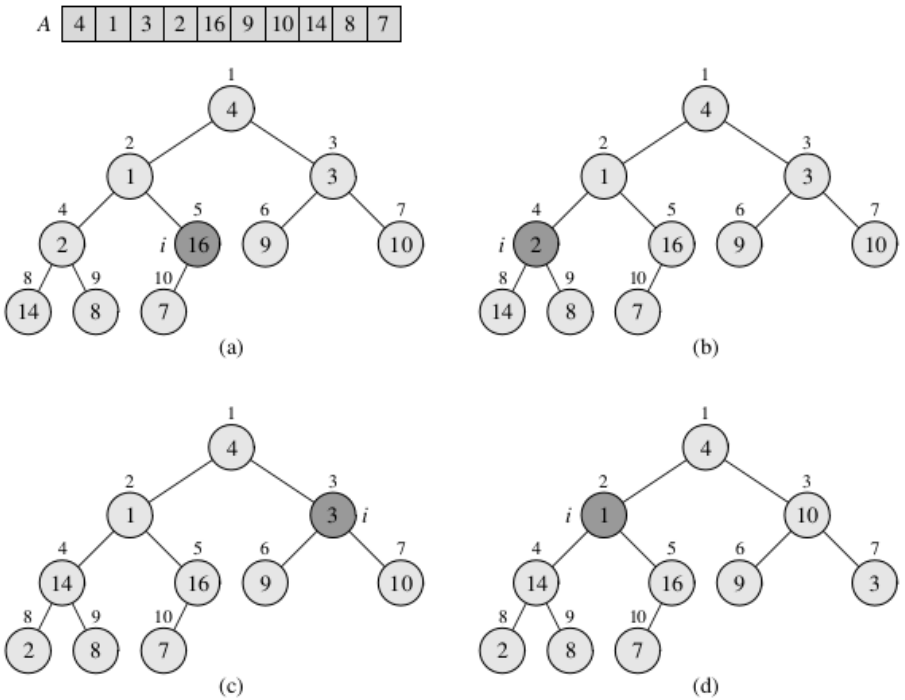
Para construir uma Heap, devemos aplicar o procedimento de **heapify** nos nós que não apresentam a propriedade de Heap.

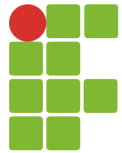


Heapsort

Heap

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





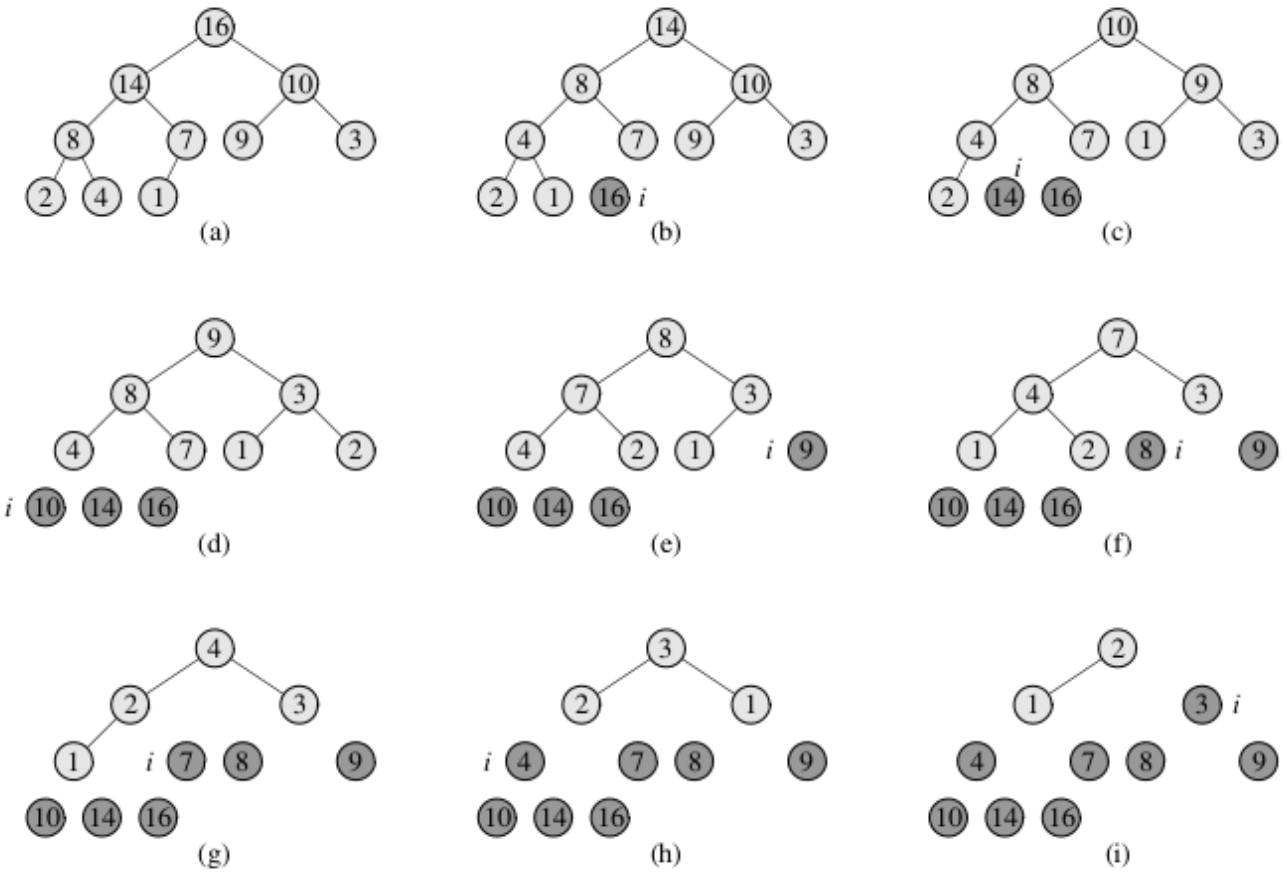
Heapsort

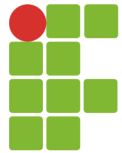
Heapsort

- 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





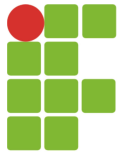
Heapsort

Function Heapsort

Input: V

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

```
1 MakeHeap(V)
2 for( i ← V.size() − 1; i > 0; i − − )
3 |   Swap(V[0], V[i])
4 |   Heapify(V, 0, i)
```



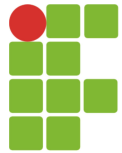
Heapsort

Function MakeHeap

Input: V

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

```
1 for( i ← V.SIZE()/2; i ≥ 0; i − − )
2   HEAPIFY(V, i, V.size())
```

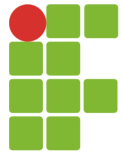


Heapsort

Function Heapify

Input: $V, i, heapSize$

```
1  l ← 2 · i + 1
2  r ← 2 · i + 2
3  largest ← i
4  if( l < heapSize ∧ V[l] > V[i] )
5      largest ← l
6  if( r < heapSize ∧ V[r] > V[largest] )
7      largest ← r
8  if( largest ≠ i )
9      SWAP(V[i], V[largest])
10     HEAPIFY(V, largest, heapSize)
```

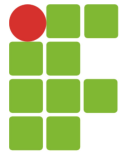


Heapsort

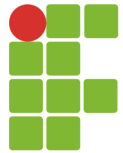
Function Heapify

Input: $V, i, heapSize$

```
1  while i < heapSize do
2      l ← 2 · i + 1
3      r ← 2 · i + 2
4      largest ← i
5      if( l < heapSize ∧ V[l] > V[i] )
6          largest ← l
7      if( r < heapSize ∧ V[r] > V[largest] )
8          largest ← r
9      if( largest = i )
10         break;
11     SWAP(V[i], V[largest])
12     i ← largest
```



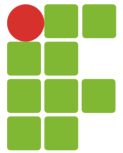
Sumário



Heapsort

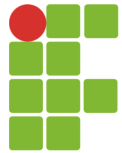
Análise

- 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

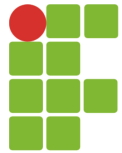
In-place	Estável
✓	✗



Heapsort

Teorema

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



Heapsort

Demonstração

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

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

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil O(h) \quad \in \quad \diamond \text{ cada nível de altura h tem essa quantidade de folhas}$$

$$O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right) \quad \leq \quad \diamond \text{ Isola o termo } n$$

$$O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h} \right) \quad = \quad \diamond \text{ Majoração}$$

$$O\left(n \frac{1/2}{(1-1/2)^2} \right) \quad = \quad \diamond \text{ Equivalência}$$

$$O\left(2n\right) \quad \in O(n)$$

□