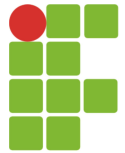
Ordenação: Heapsort

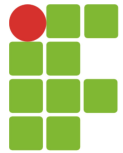
Estruturas de Dados e Algoritmos – Ciência da Computação

Prof. Daniel Saad Nogueira Nunes

IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



Heapsort

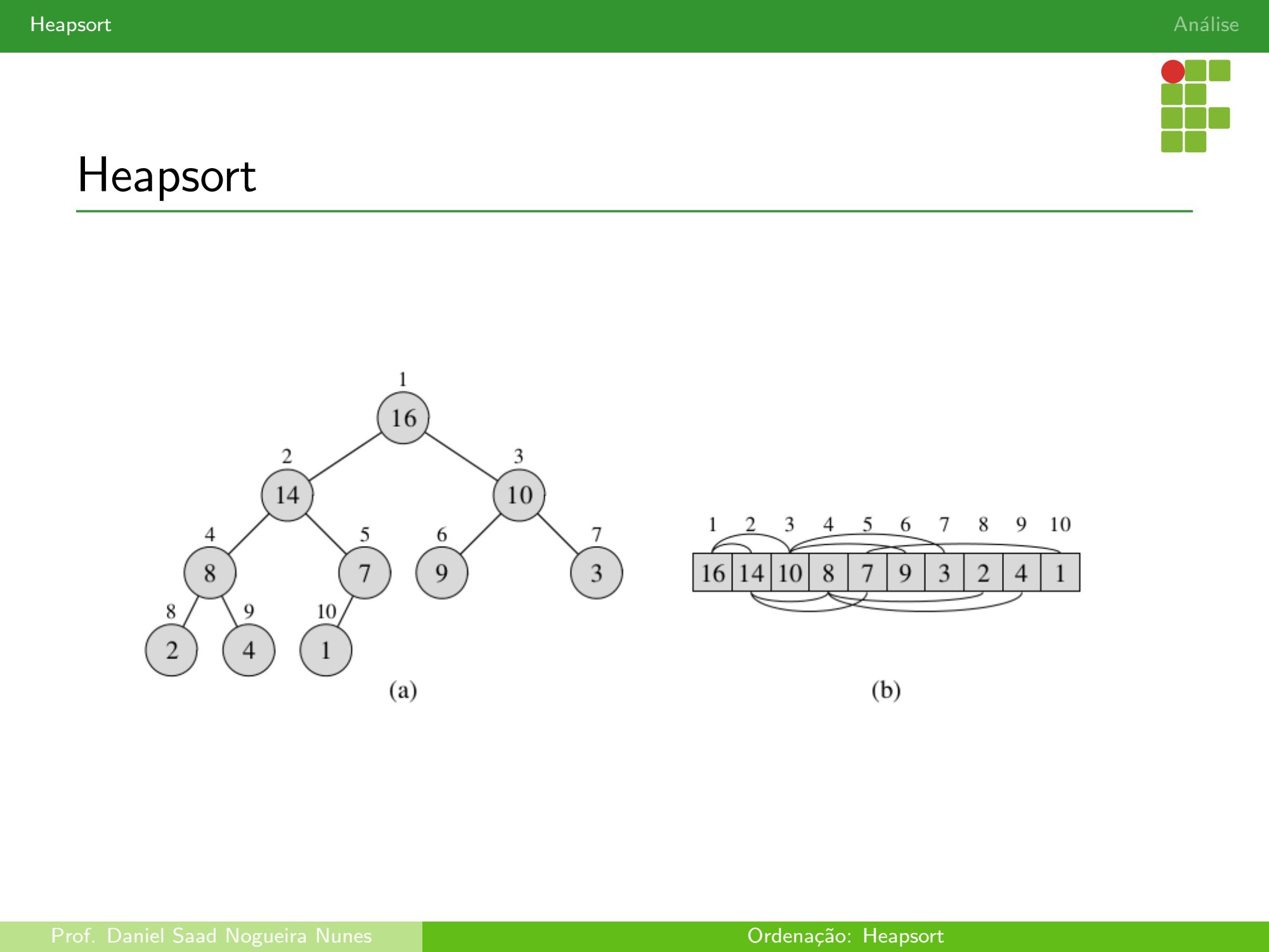
Heap

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.

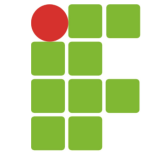
iii) O filho da direita também é uma heap.



Heapsort

(a)

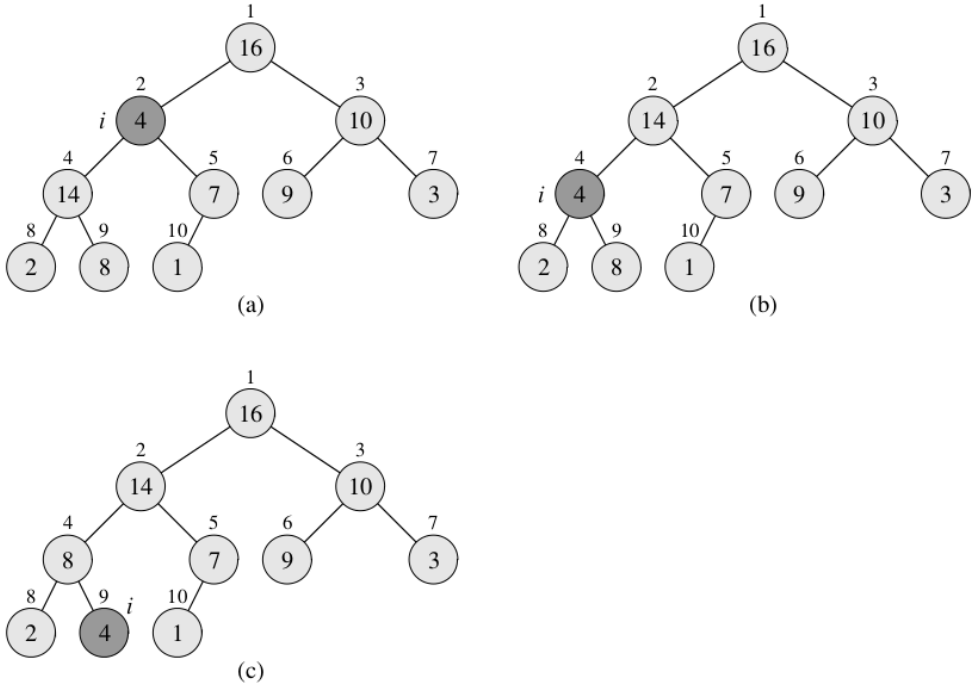
(b)

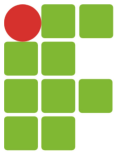


Heapsort

Heapify

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

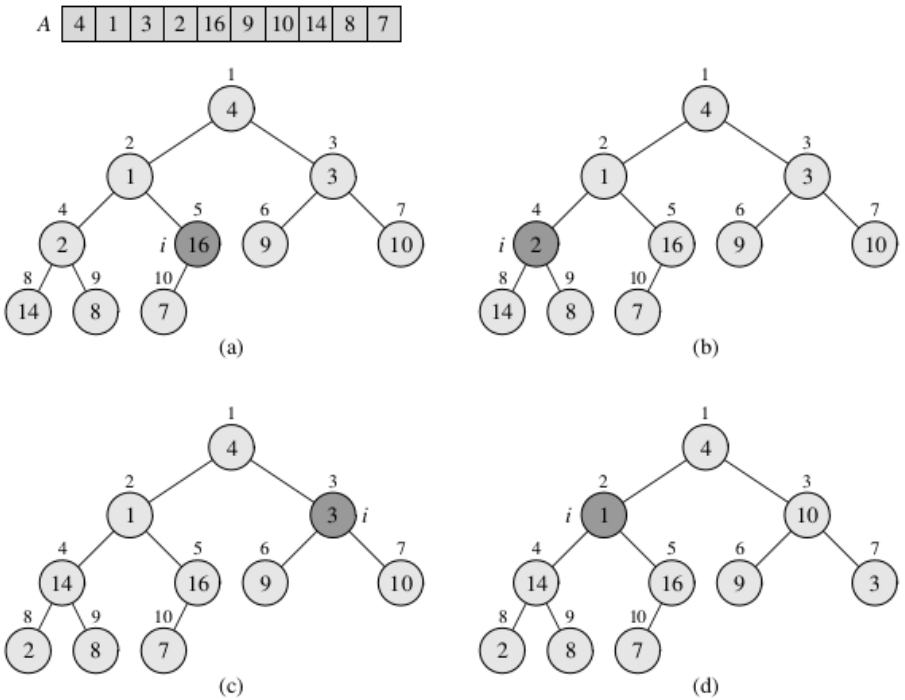


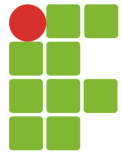


Heapsort

Heap

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





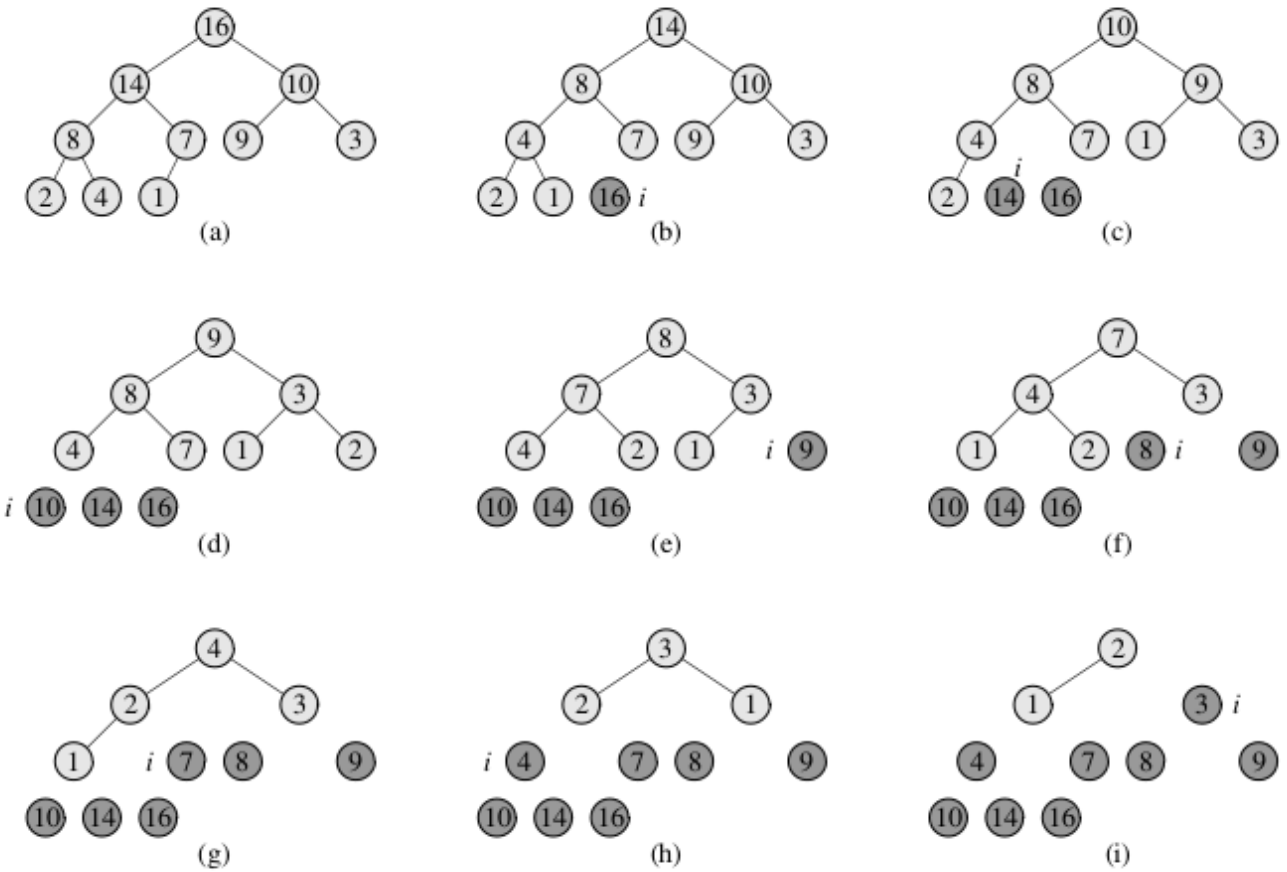
Heapsort

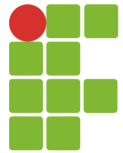
Heapsort

- 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

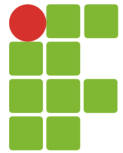
Exemplo





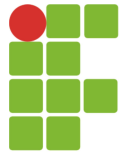
Heapsort

```
1  static void heapify(int *v, size_t i, size_t size) {
2      int left;
3      int right;
4      int largest;
5      while (i < size) {
6          left = (i * 2) + 1;
7          right = (i * 2) + 2;
8          largest = i;
9          if (left < size && v[left] > v[largest]) {
10             largest = left;
11         }
12         if (right < size && v[right] > v[largest]) {
13             largest = right;
14         }
15         if (i == largest) {
16             break;
17         }
18         int swp = v[i];
19         v[i] = v[largest];
20         v[largest] = swp;
21         i = largest;
22     }
```

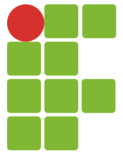


Heapsort

```
25  static void make_heap(int *v, size_t size) {
26      int i;
27      for (i = size / 2; i >= 0; i--) {
28          heapify(v, i, size);
29      }
30  }
31
32  void heap_sort(int *v, size_t size) {
33      make_heap(v, size);
34      for (int i = size - 1; i > 0; i--) {
35          int swp = v[i];
36          v[i] = v[0];
37          v[0] = swp;
38          heapify(v, 0, i);
39      }
40  }
```



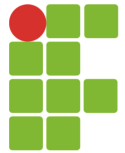
Sumário



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
✓	✗