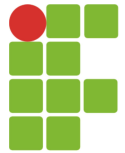
Ordenação: Heapsort

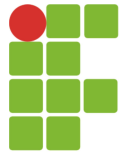
Estruturas de Dados e Algoritmos

Prof. Daniel Saad Nogueira Nunes

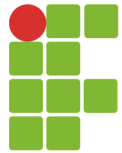
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário



Sumário



Heapsort

Heap

A conceito central do heapsort é uma estrutura denominada **heap**. Uma heap binária é uma árvore, estrutura de natureza recursiva, e tem as seguinte propriedades:

i) O elemento pai é $\geq$ do que os seus filhos (caso existam).

ii) O filho da esquerda é uma heap.

iii) O filho da direita também é uma heap.

Heapsort

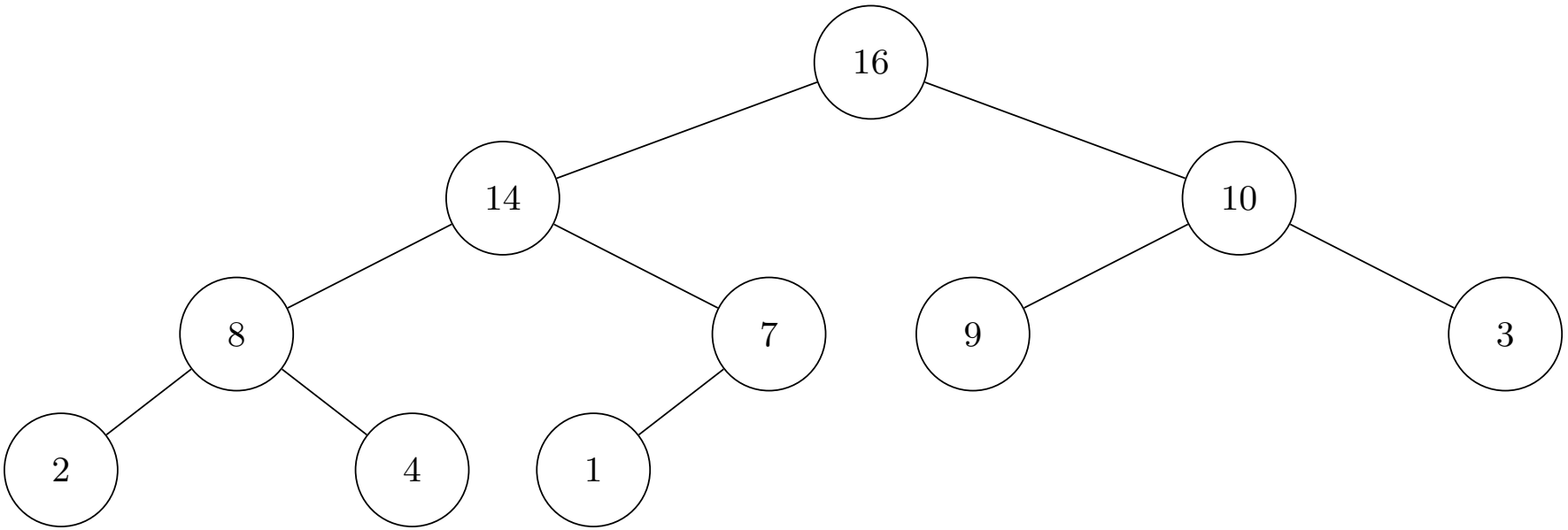
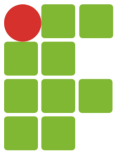
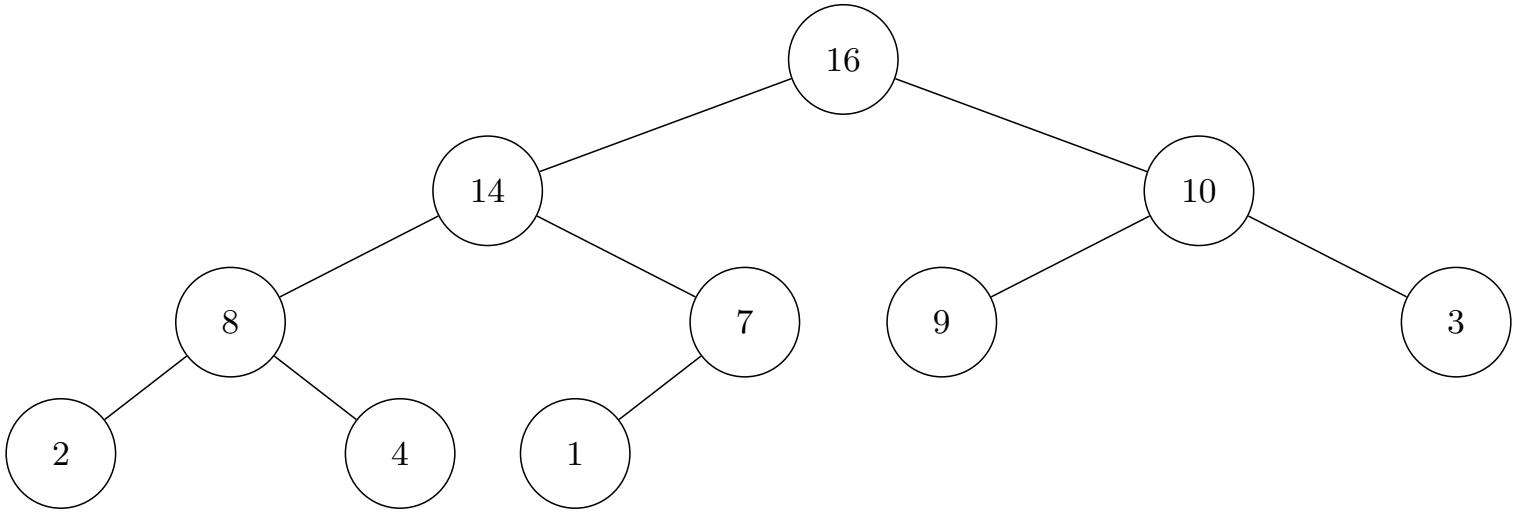


Figura: Heap binária.

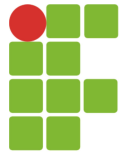


Heapsort

- Contudo estamos interessados em ordenar vetores.
- É possível representar a árvore em um vetor.



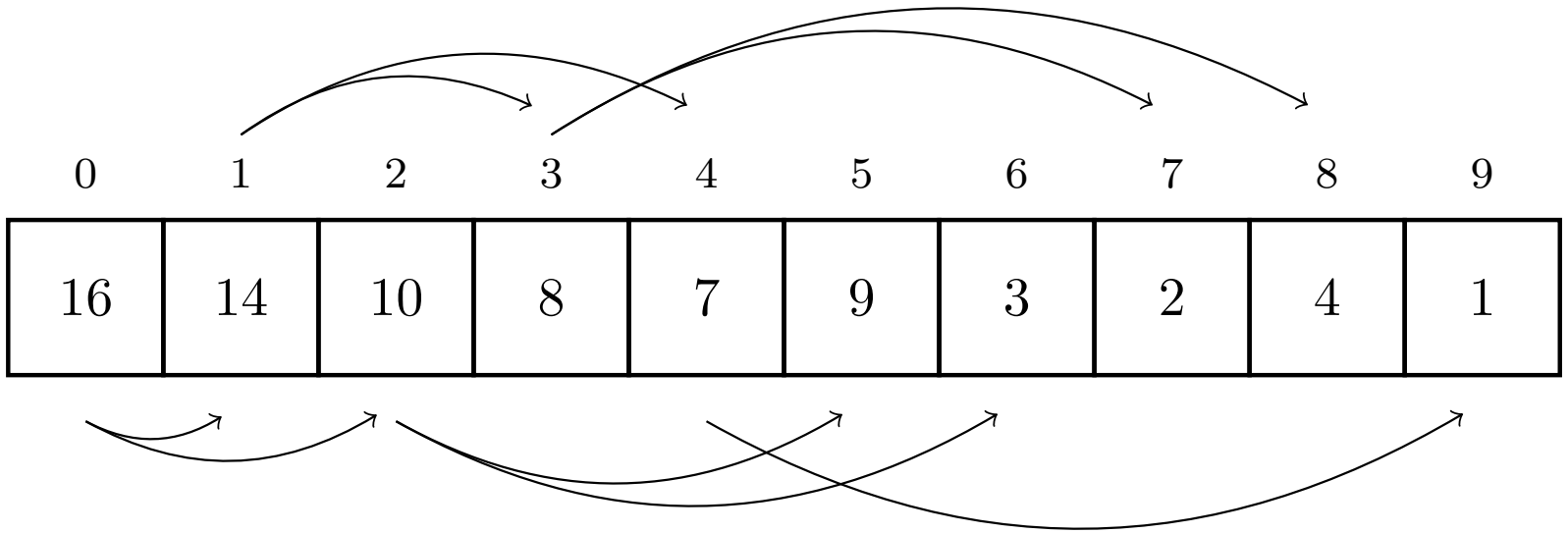
0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

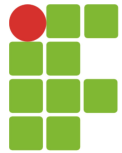


Heapsort

- Como navegar na árvore se temos apenas a representação de vetor? Não perdemos informação?
- **Não!** Navegar na árvore é simples. Para um elemento na posição i:
  - O filho da esquerda está na posição $2i + 1$.
  - O filho da direita está na posição $2i + 2$.

Heapsort

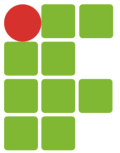




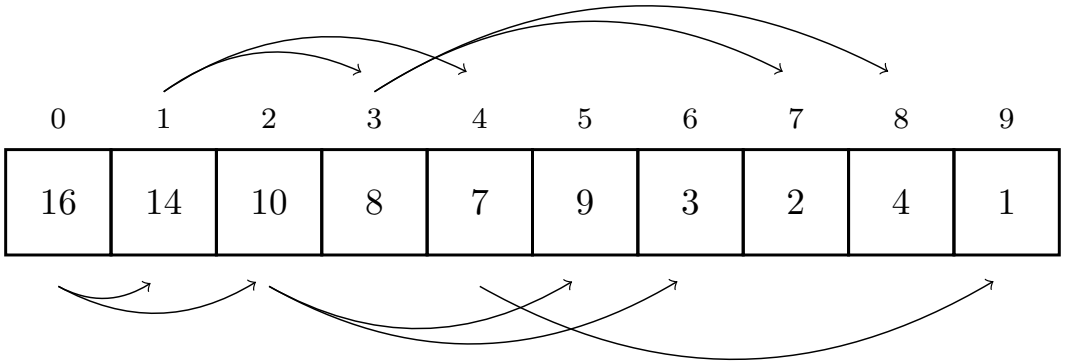
Heapsort

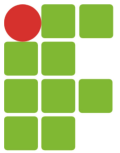
Heap

- Em uma heap, o maior elemento sempre se encontra na posição 0.
- A chave do Heapsort é colocar o maior elemento ao final do vetor e reestruturar a Heap, para que o próximo maior elemento vá para a posição 0.
- A operação que reestrutura a heap é chamada de **heapify**
- Ela consiste em trocar o elemento com o maior dos seus filhos, até que não seja mais possível.

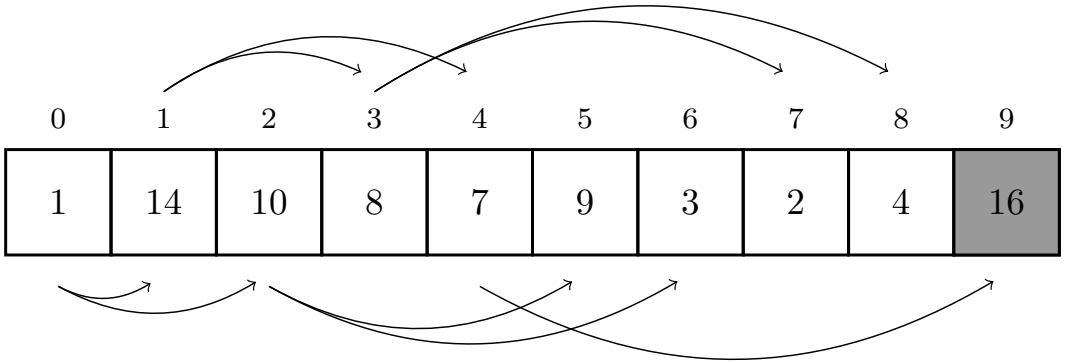


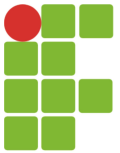
Heapsort



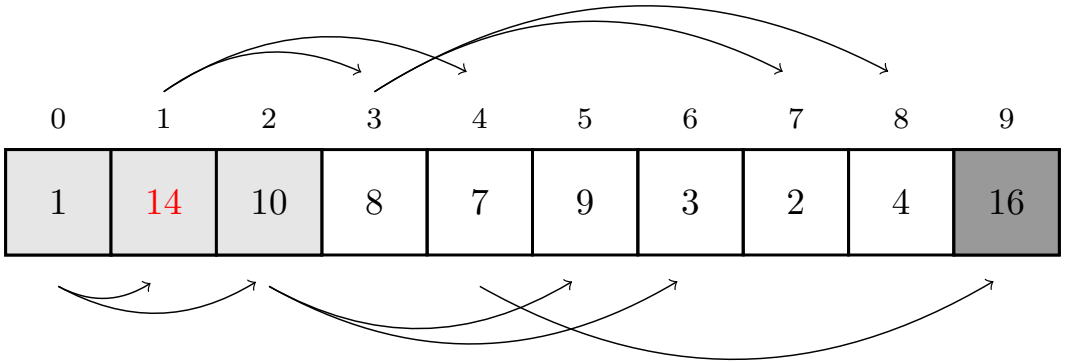


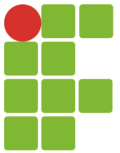
Heapsort



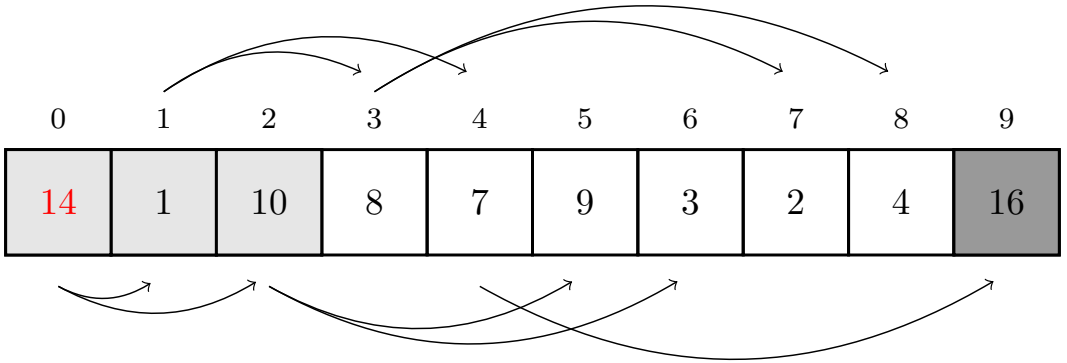


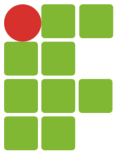
Heapsort



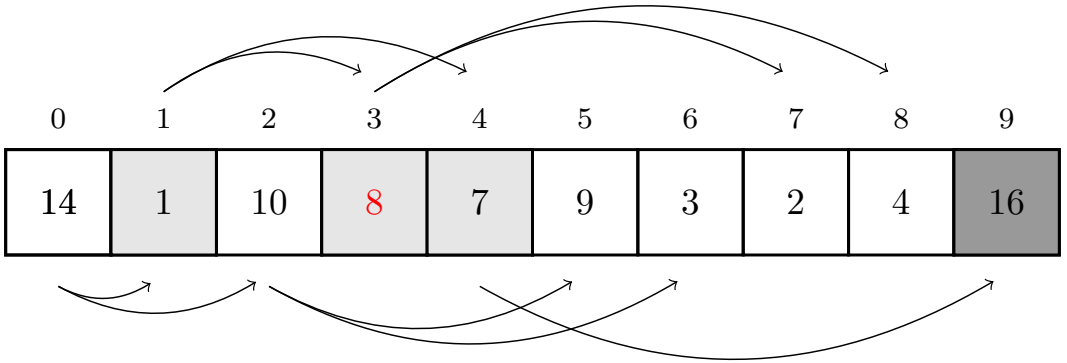


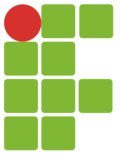
Heapsort



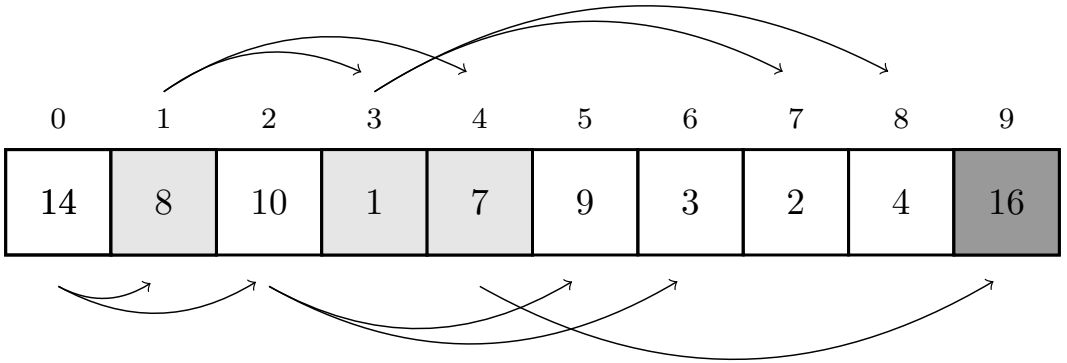


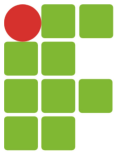
Heapsort



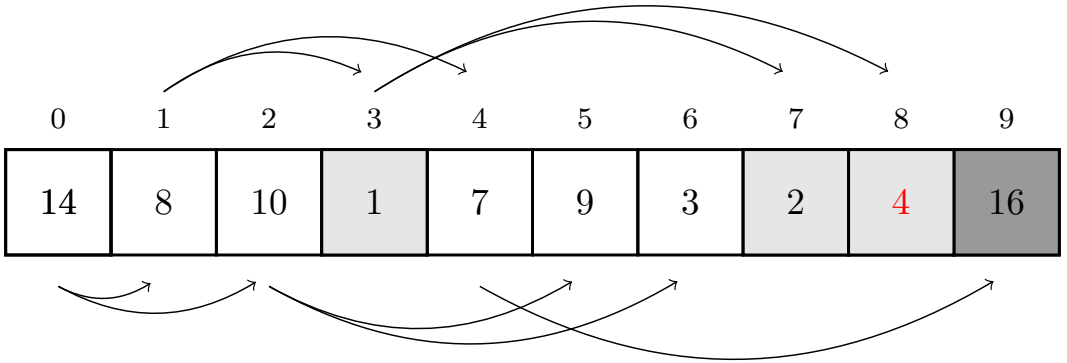


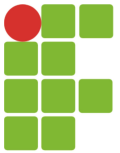
Heapsort



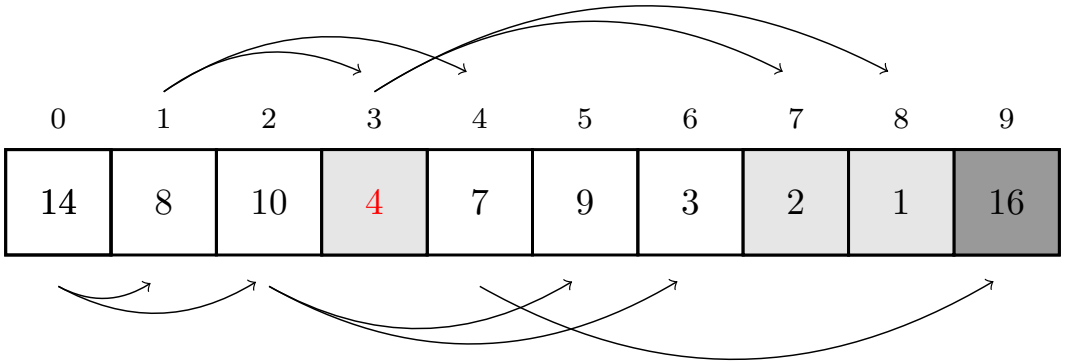


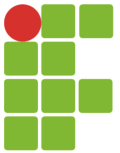
Heapsort



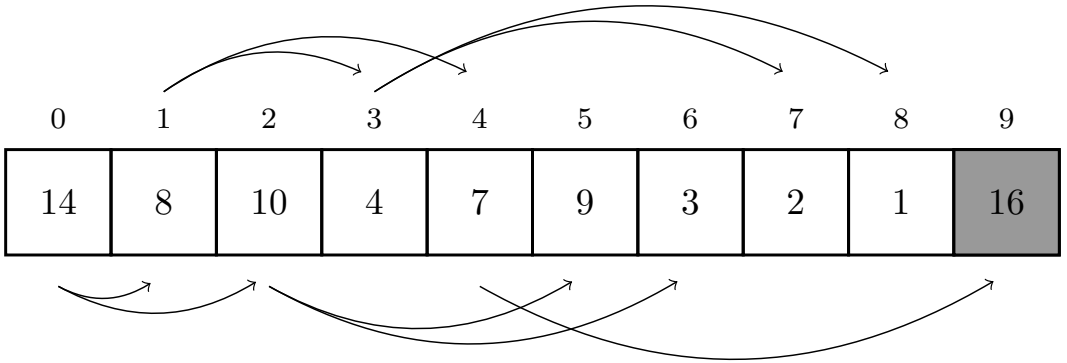


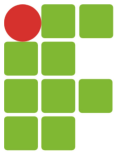
Heapsort



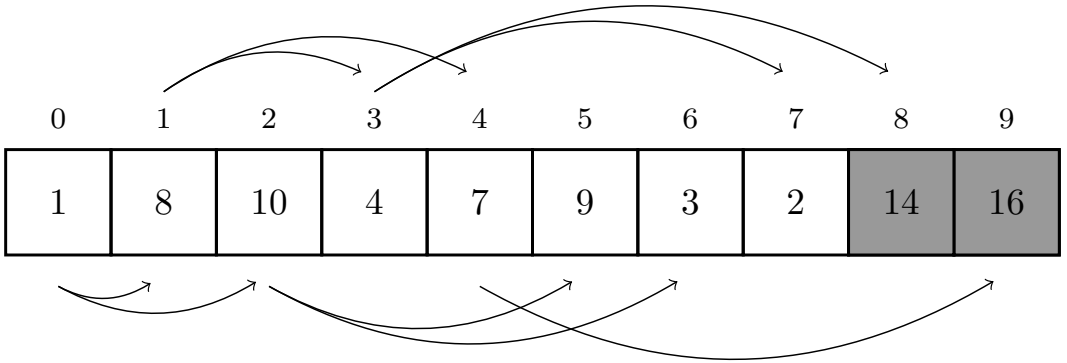


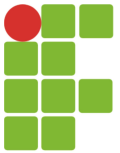
Heapsort



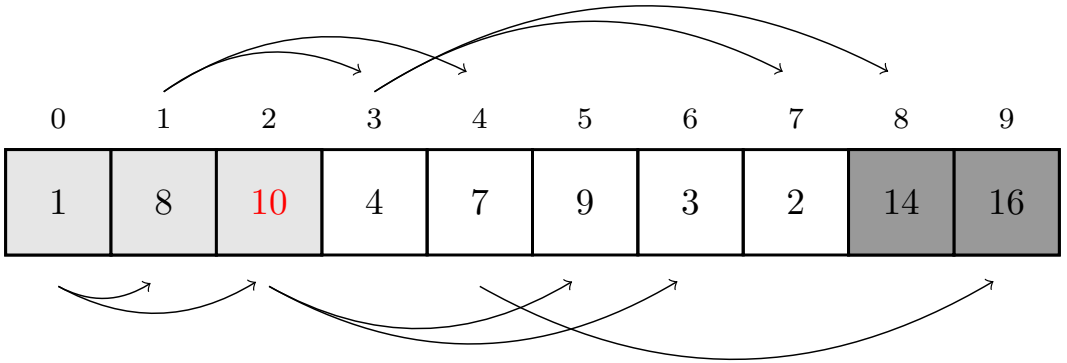


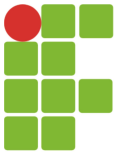
Heapsort



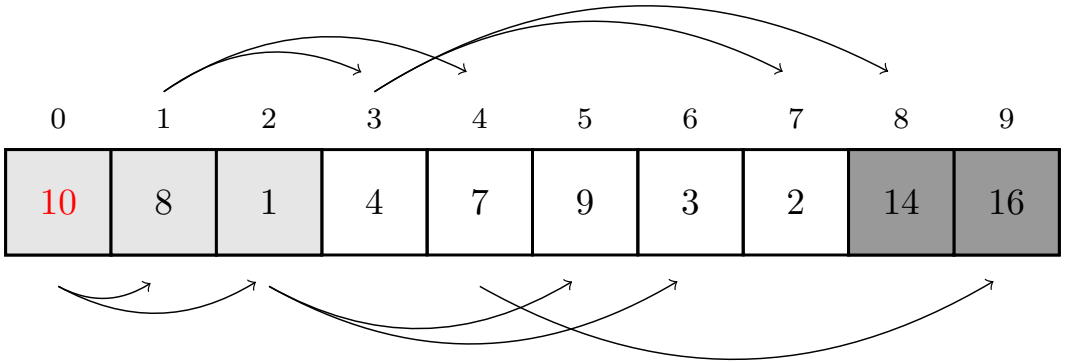


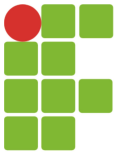
Heapsort



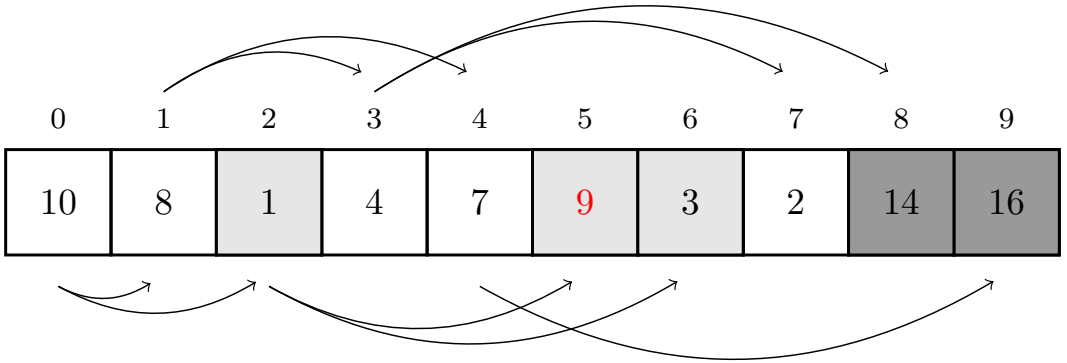


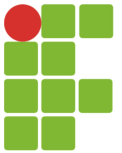
Heapsort



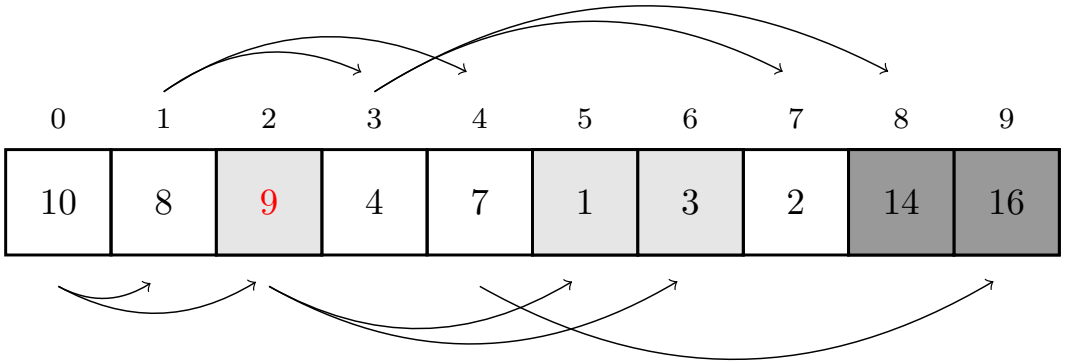


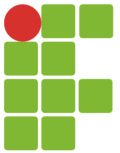
Heapsort



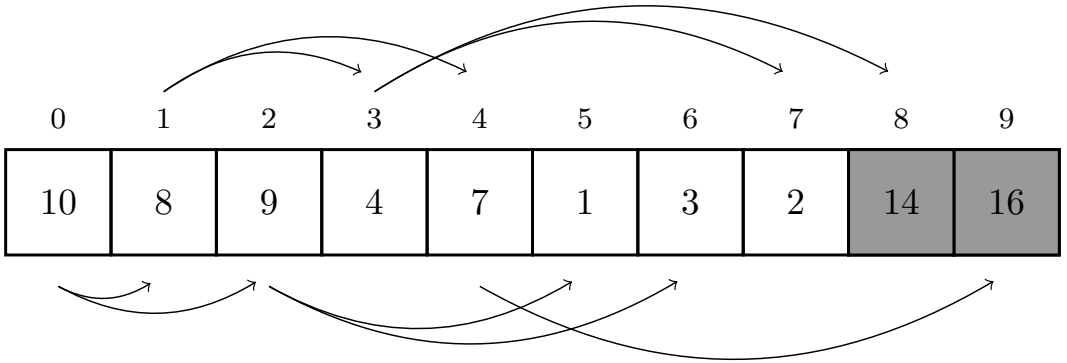


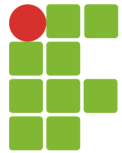
Heapsort





Heapsort



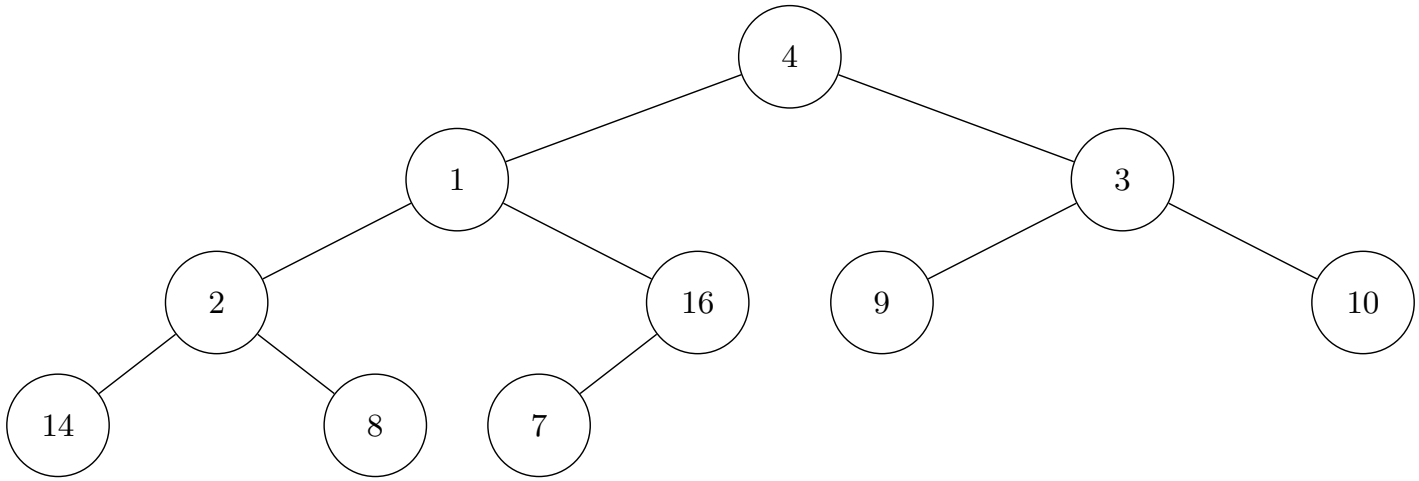


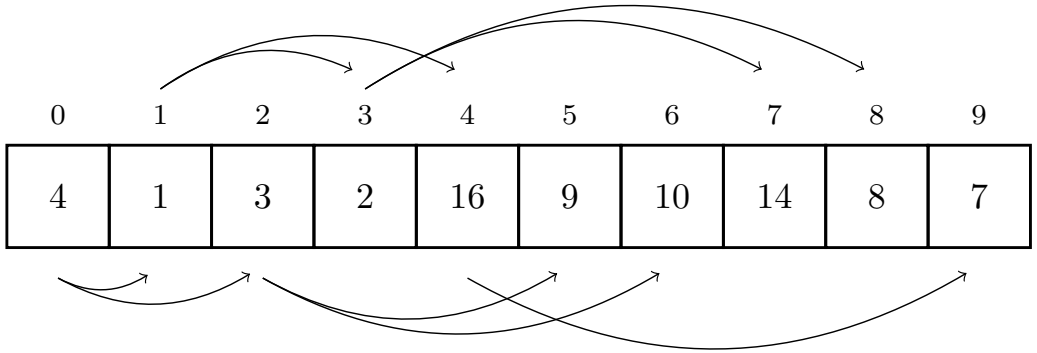
Heapsort

Makeheap

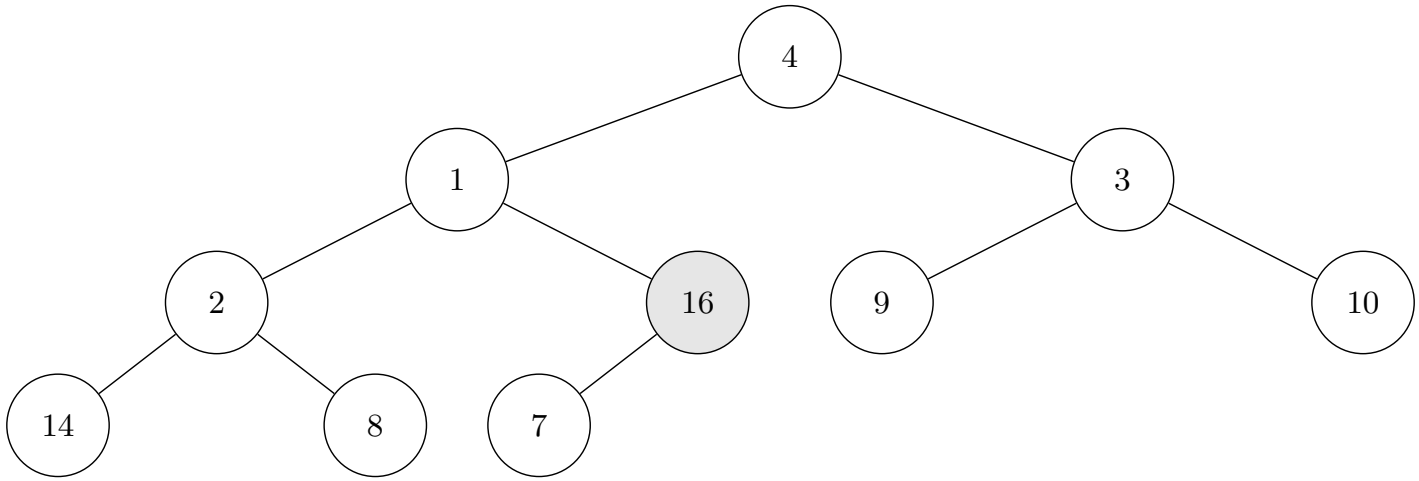
- Com uma heap, conseguimos construir um vetor? Mas como obter uma heap a princípio?
- Para construir uma Heap, devemos aplicar o procedimento de **heapify** nos nós que não apresentam a propriedade de Heap.
- O primeiro nó que devemos verificar é o que está na metade do vetor, pois é um nó que possui filhos.
- Varremos o vetor da direita para a esquerda.

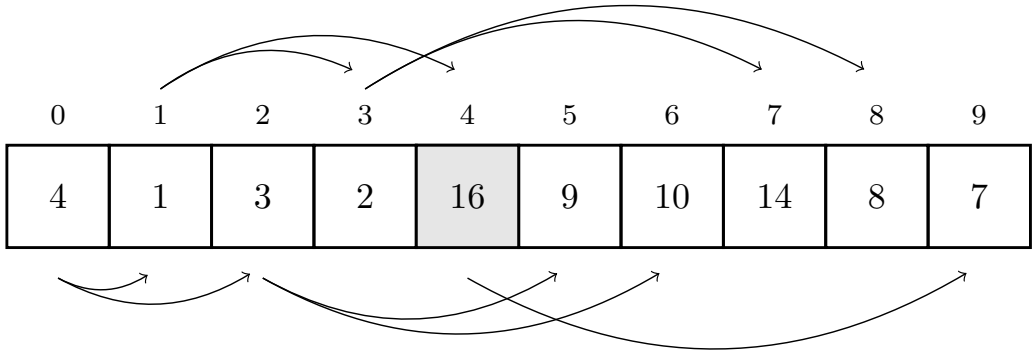
Heapsort



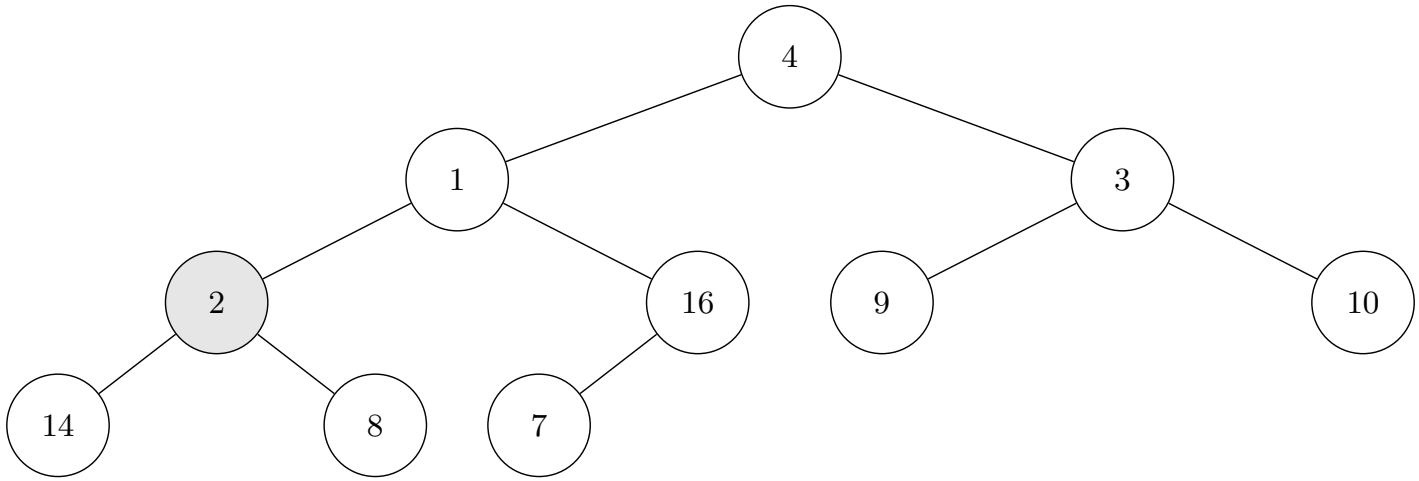


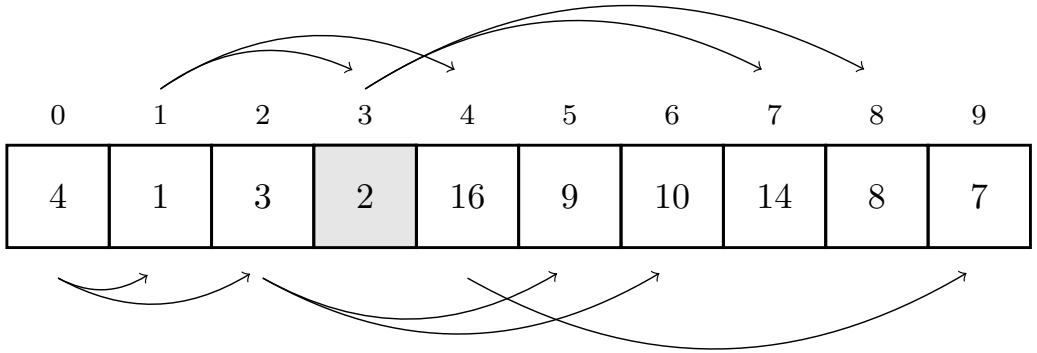
Heapsort



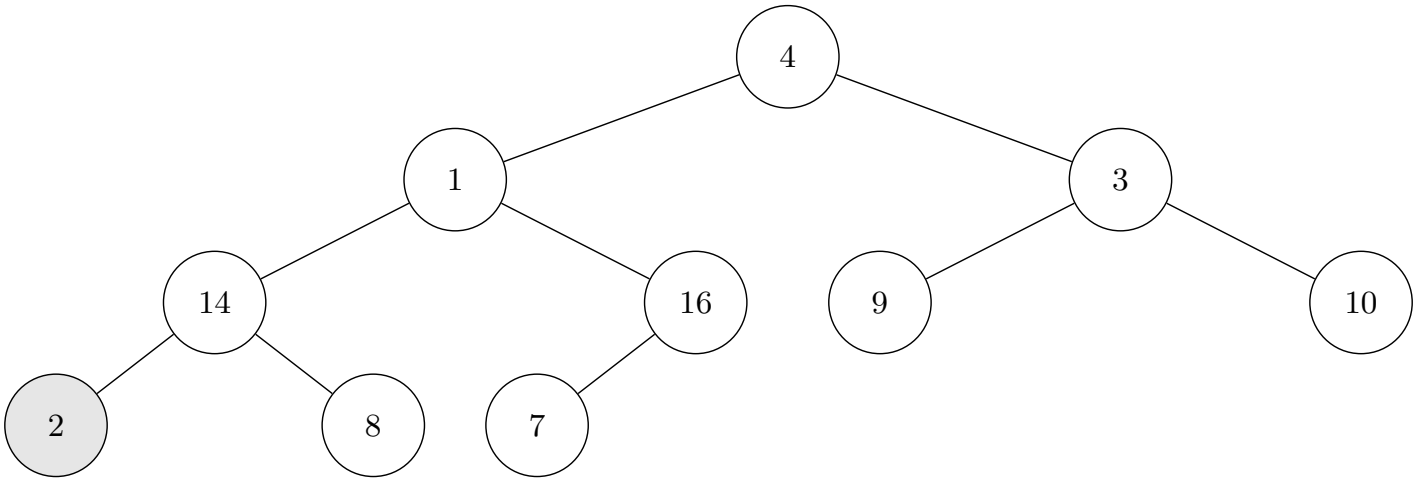


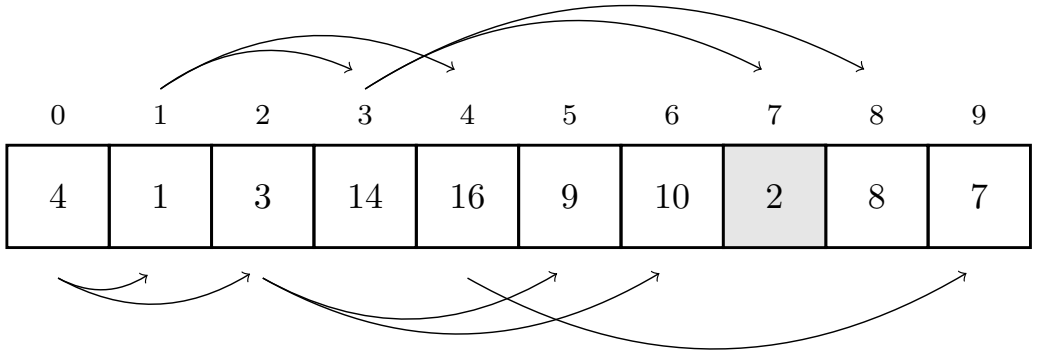
Heapsort



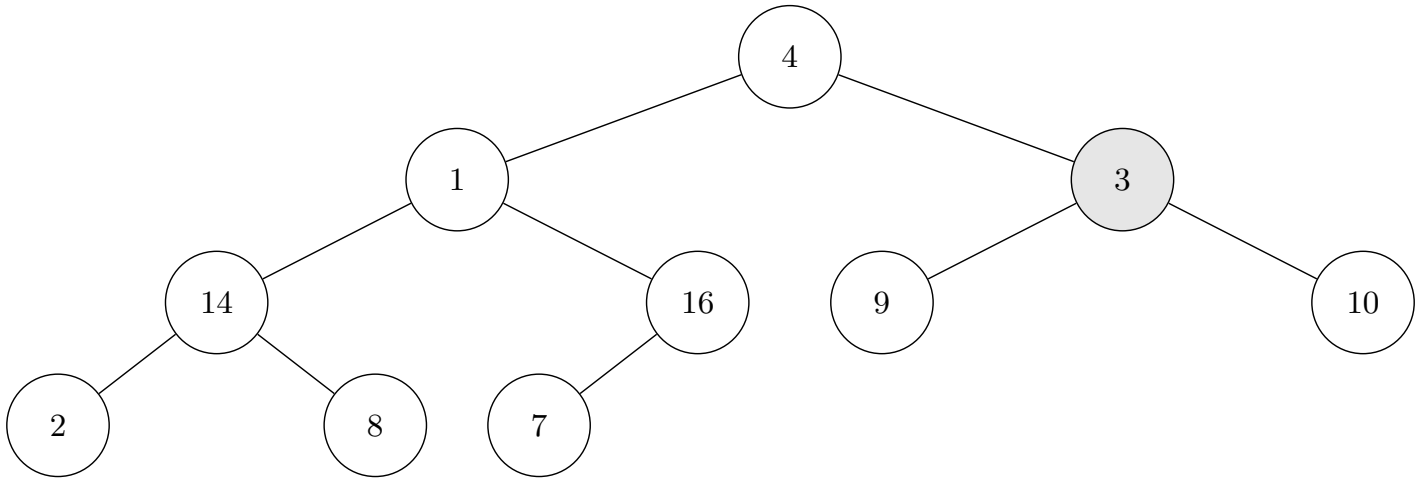


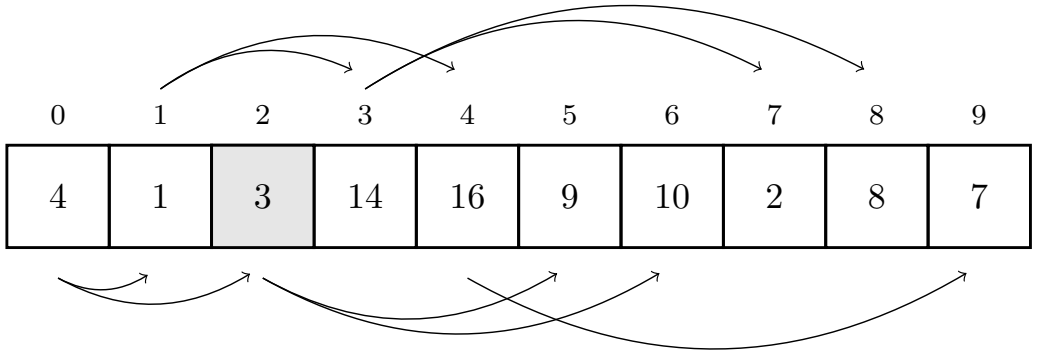
Heapsort

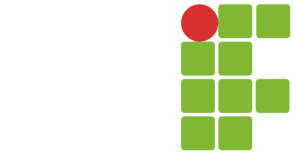




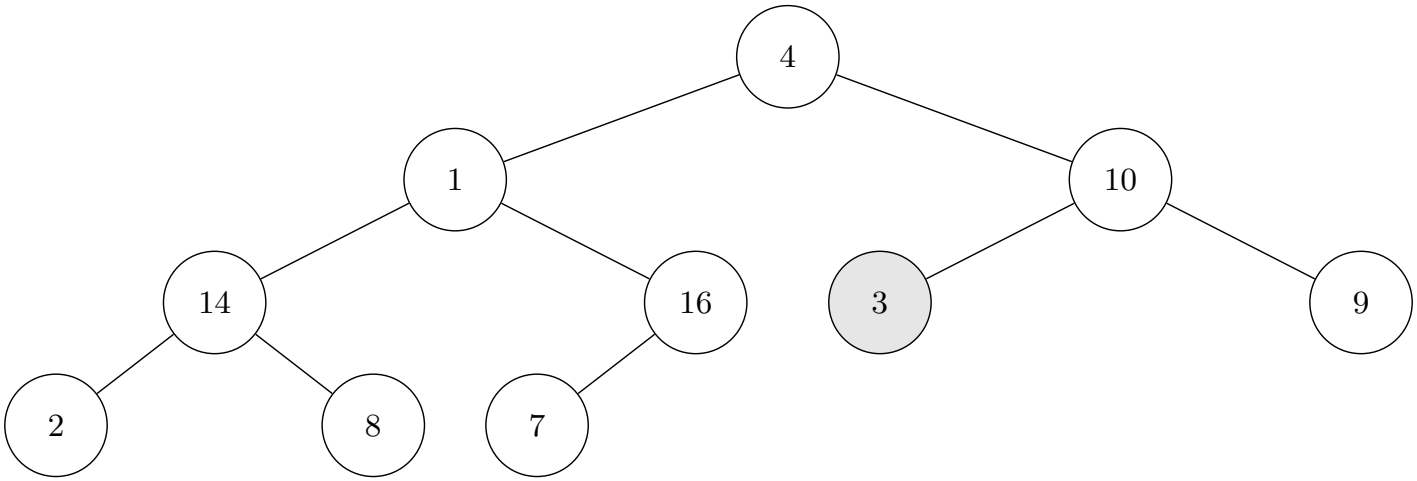
Heapsort

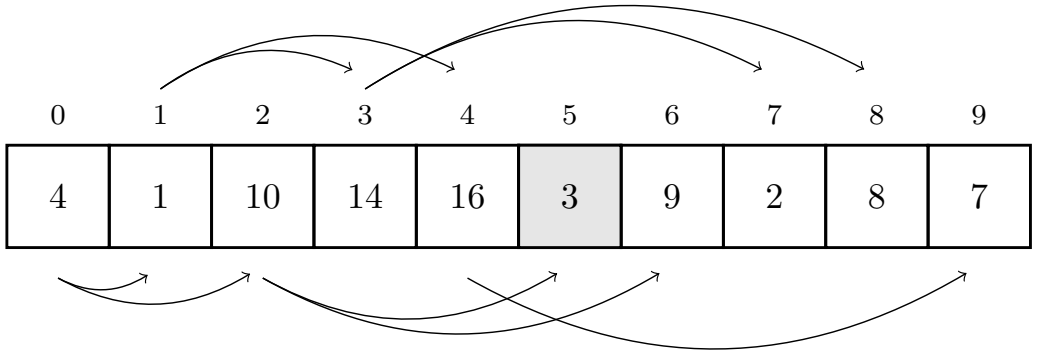




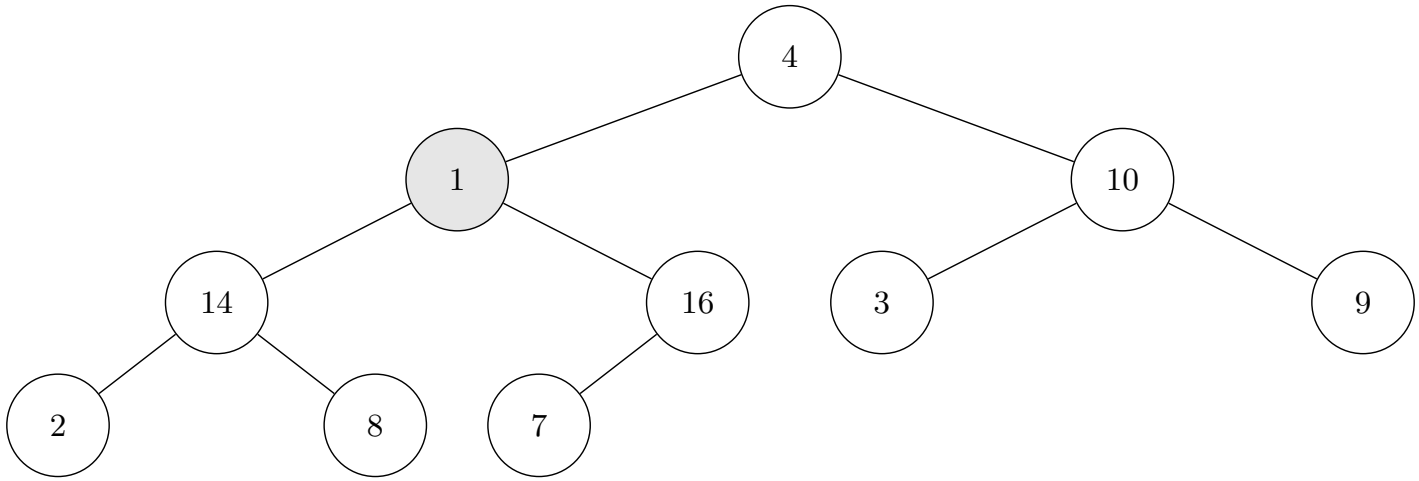


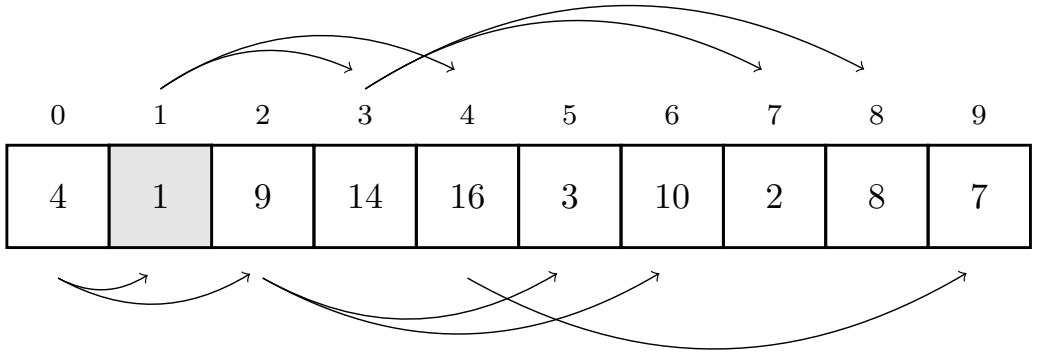
Heapsort



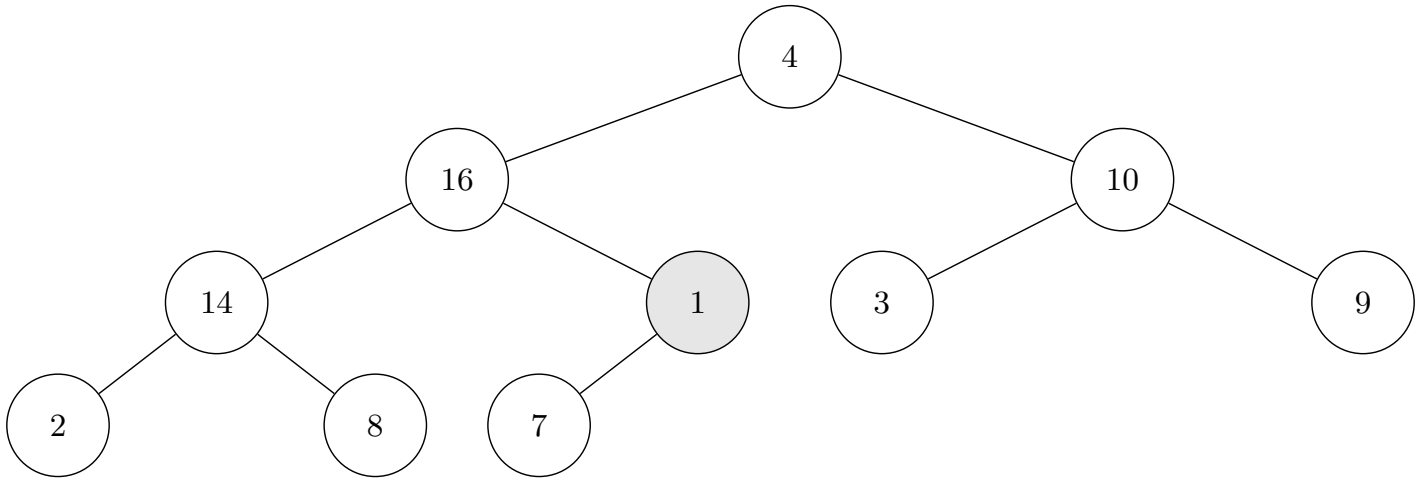


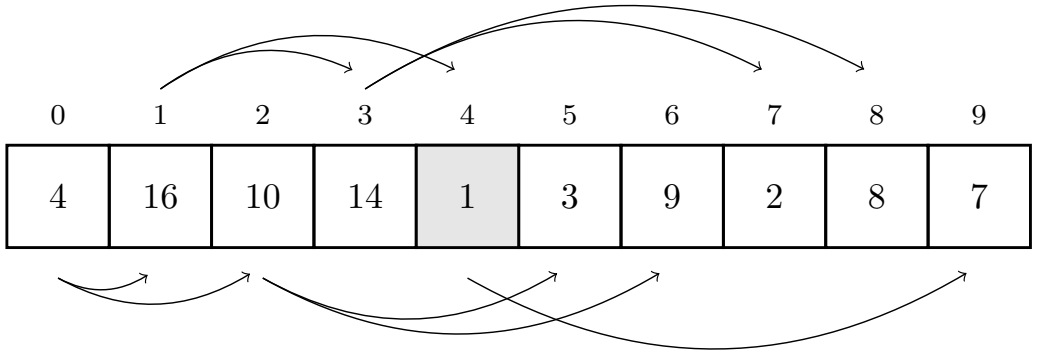
Heapsort



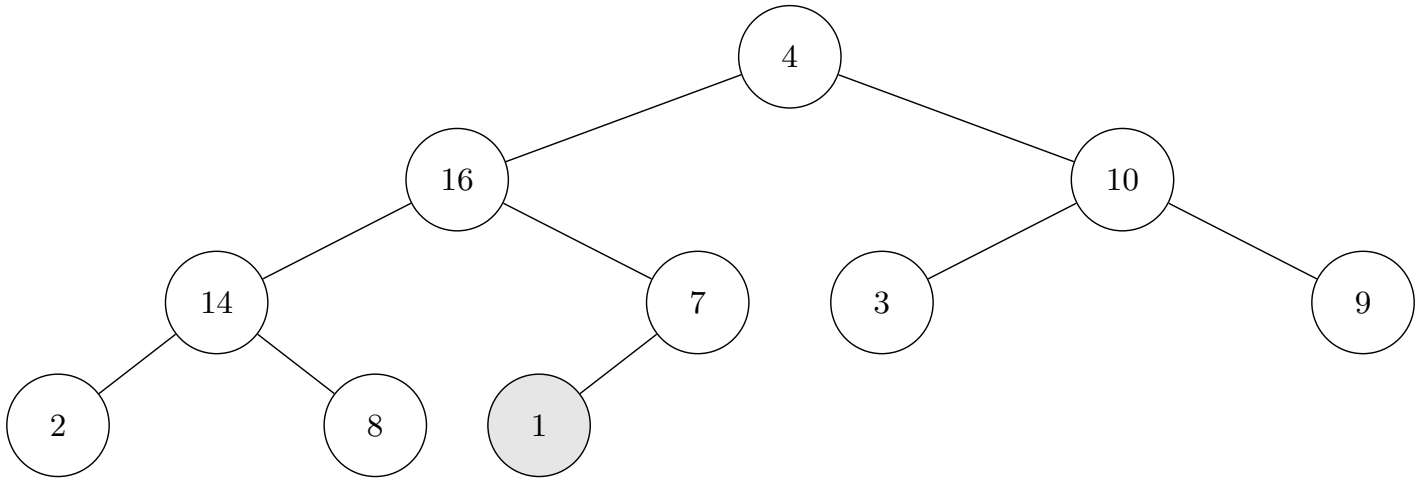


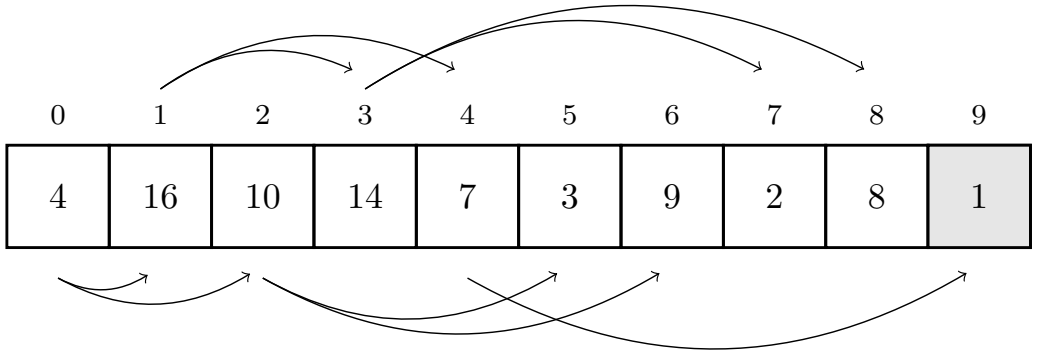
Heapsort



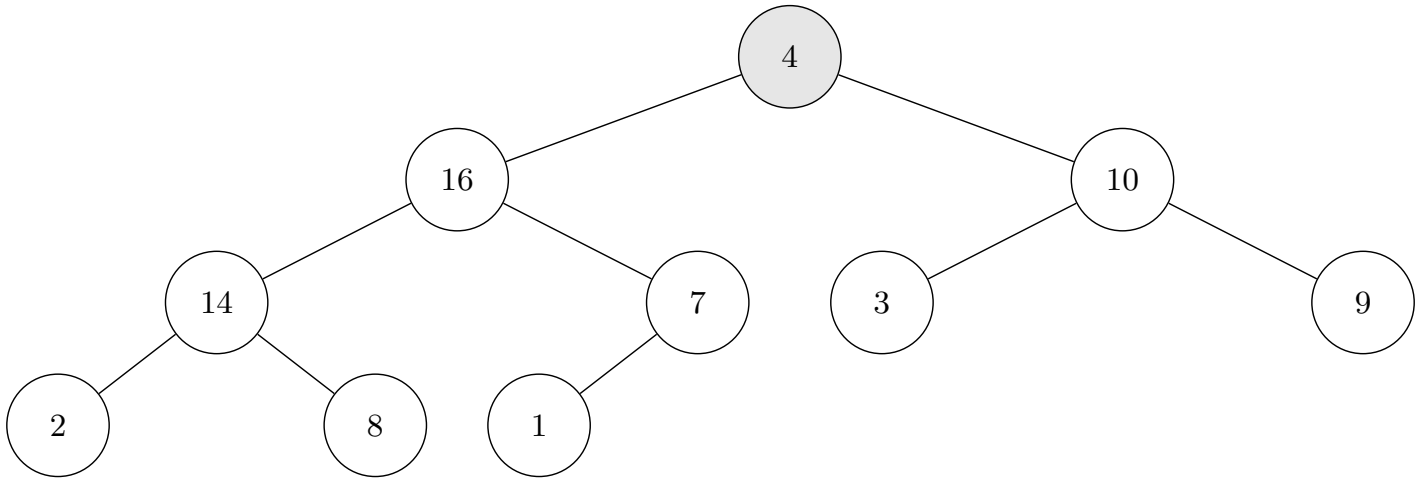


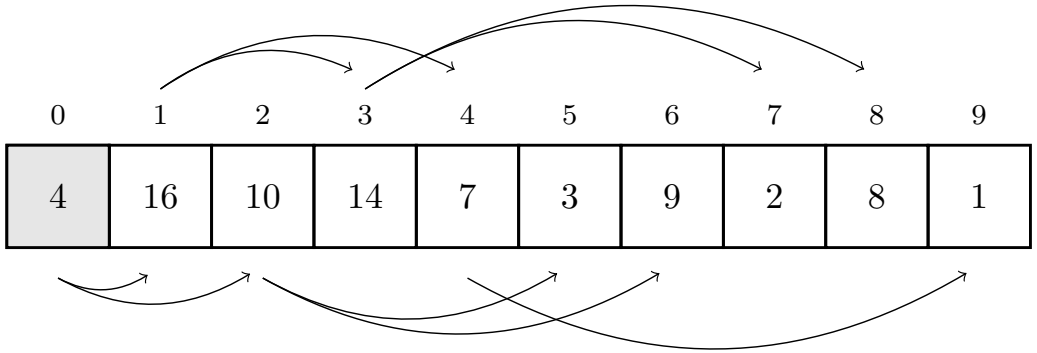
Heapsort



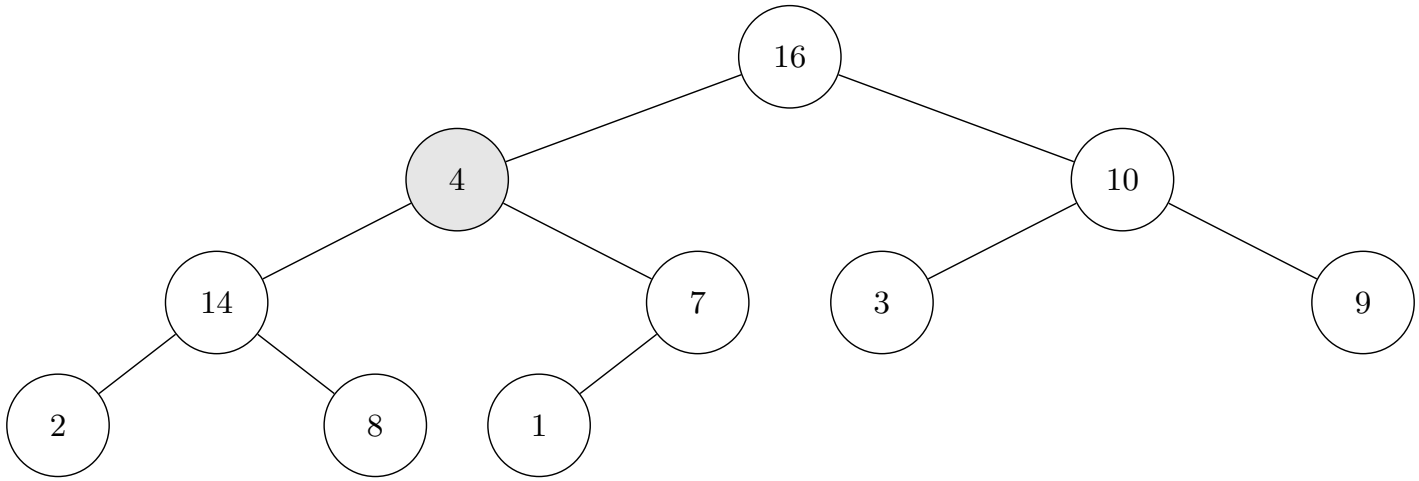


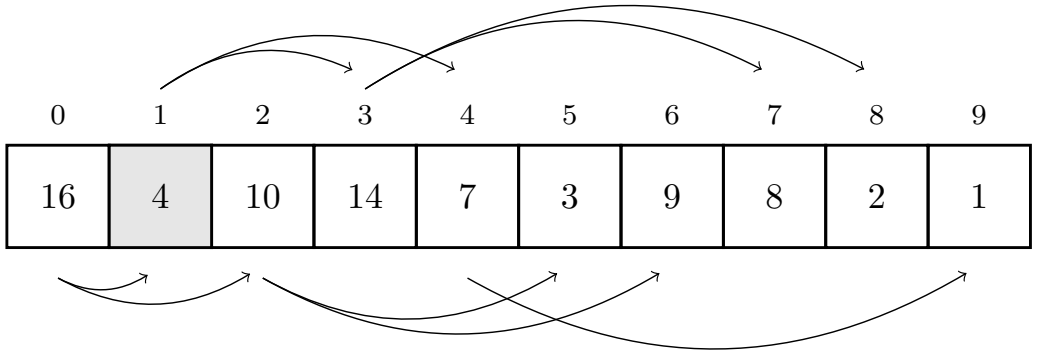
Heapsort



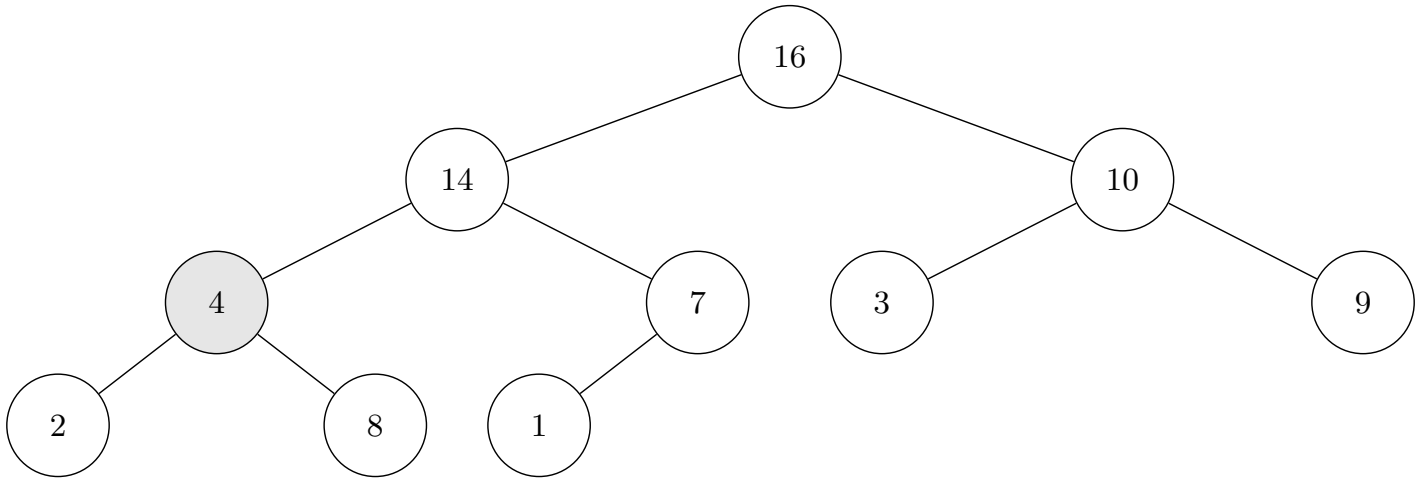


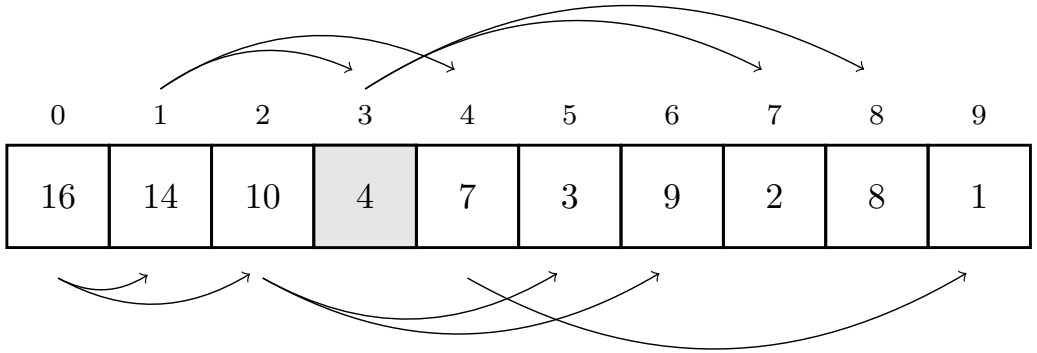
Heapsort

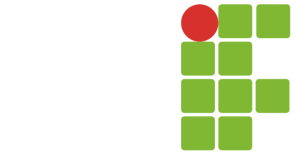




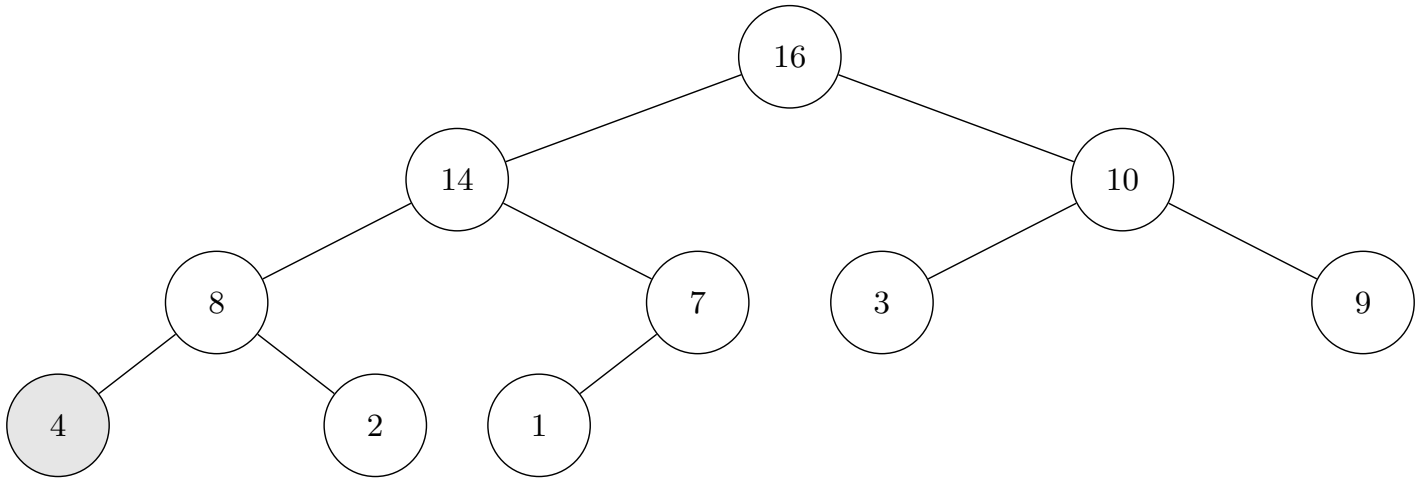
Heapsort

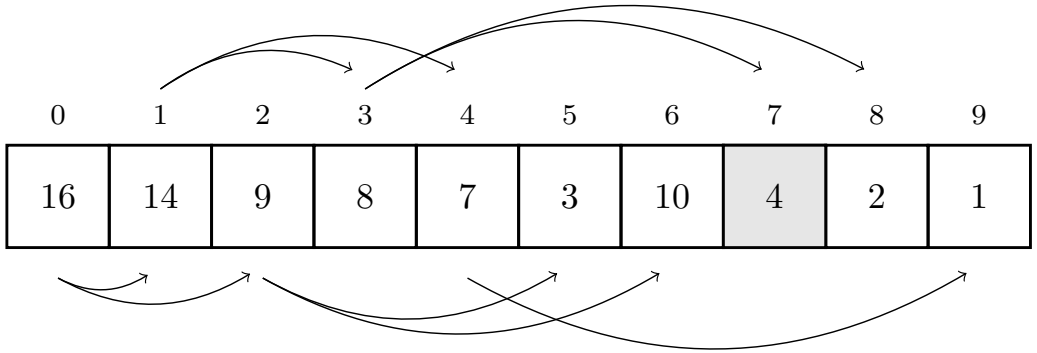


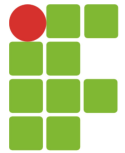




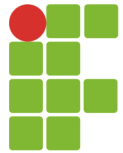
Heapsort





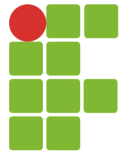


Sumário



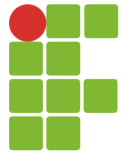
Heapsort

```
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     }
```

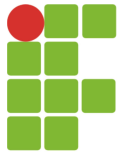


Heapsort

```
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  }
```



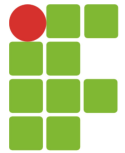
Sumário



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
✓	✗