Algoritmos - Aula 5

Fernando Raposo

Vamos ver

- Heap sort
- Análise Comparativa das Ordenações
- Conceitos Algorítmicos

Heap Sort

- Algoritmo de ordenação baseado em comparações, similar ao Selection Sort. (procuramos o menor elemento e colocamos no início). Deve-se considerar no processo uma estrutura de Heap Binário (**Binary Heap**).

Heap Binário o que é?

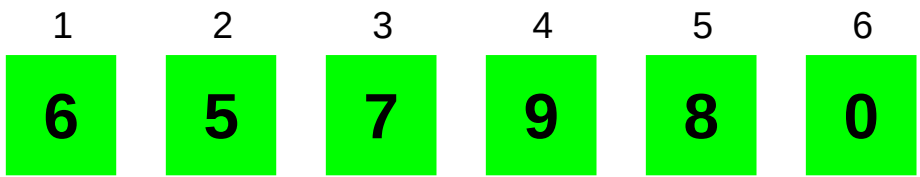
1. Uma árvore;
2. Esta árvore pode ser representada por um array;
3. Cada nó filho é menor que seu nó pai (ou o nó pai é maior que seus nós filhos).

Heap Sort

- Enxergando um Heap Binário em um array.
  - raiz: array[1]
  1. Pai de índice f é f / 2;
  2. O filho esquerdo do índice p é 2p;
  3. O filho direito do índice p é 2p+1.

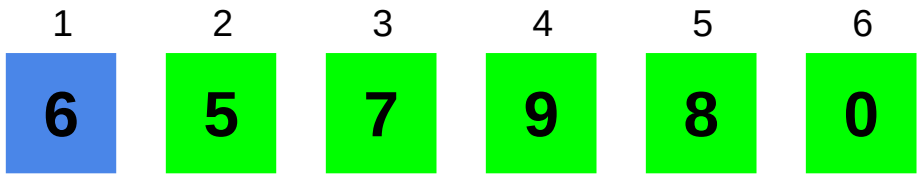
Heap Sort

- Vamos treinar (Obs: para simplificar retiramos o índice zero do array)
  - Qual a raiz?



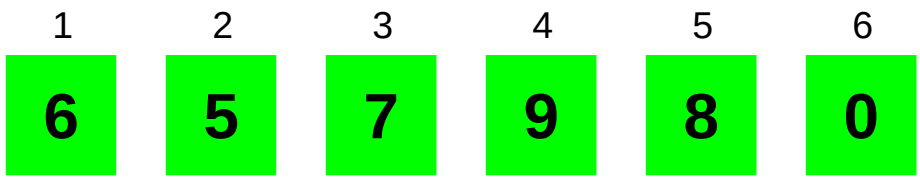
Heap Sort

- Vamos treinar (Obs: para simplificar retiramos o índice zero do array)
  - Qual a raiz?
  - R: array[1]



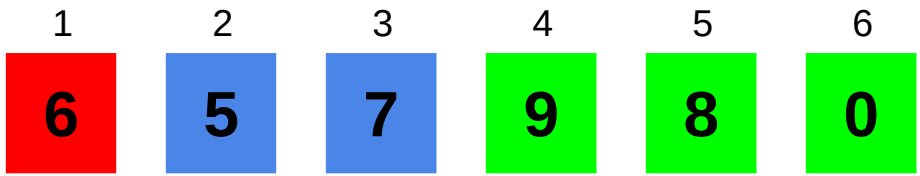
Heap Sort

- Vamos treinar (Obs: para simplificar retiramos o índice zero do array)
  - Quais os filhos da raiz array[1] ?



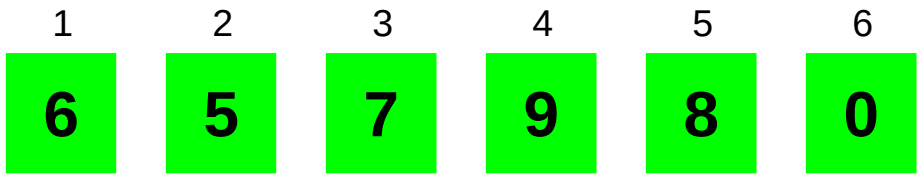
Heap Sort

- Vamos treinar (Obs: para simplificar retiramos o índice zero do array)
  - Quais os filhos da raiz array[1] ?
  - R: Filho esquerdo = 2p = 2*1 = 2
  - R: Filho direito = 2p+1 = 2*1 + 1 = 3



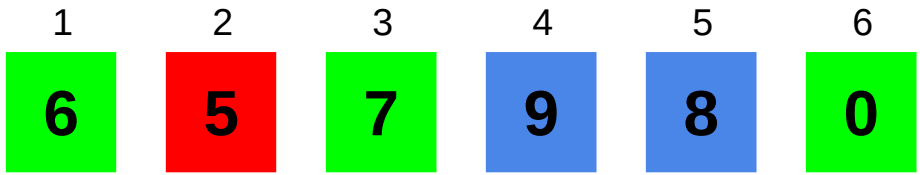
Heap Sort

- Vamos treinar (Obs: para simplificar retiramos o índice zero do array)
  - Quais os filhos da raiz array[2] = 2 ?
  - R: Filho esquerdo = 2p = 2*2 = 4
  - R: Filho direito = 2p+1 = 2*2 + 1 = 5



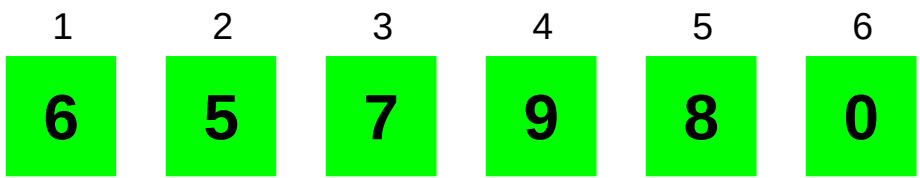
Heap Sort

- Vamos treinar (Obs: para simplificar retiramos o índice zero do array)
  - Quais os filhos da raiz array[2] = 2 ?
  - R: Filho esquerdo = 2p = 2*2 = 4
  - R: Filho direito = 2p+1 = 2*2 + 1 = 5



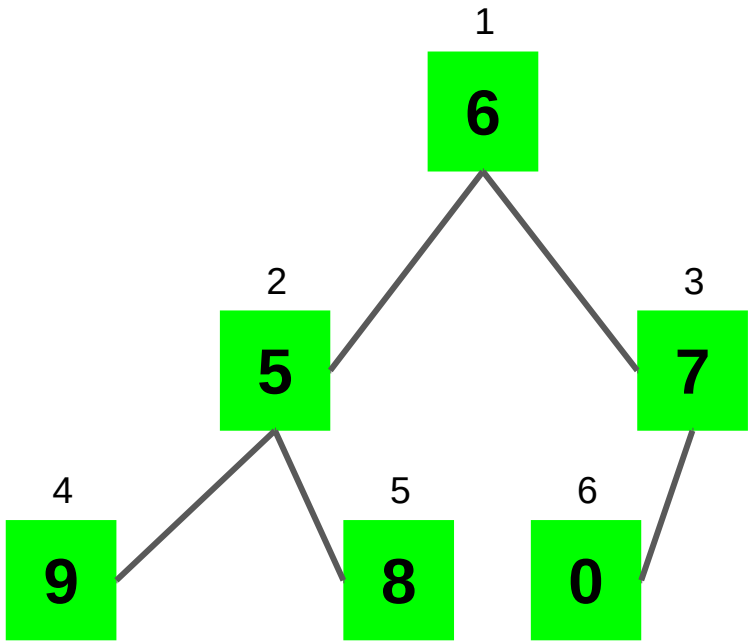
Heap Sort

- Fazendo as contas e montando o Heap…
- ANTES:



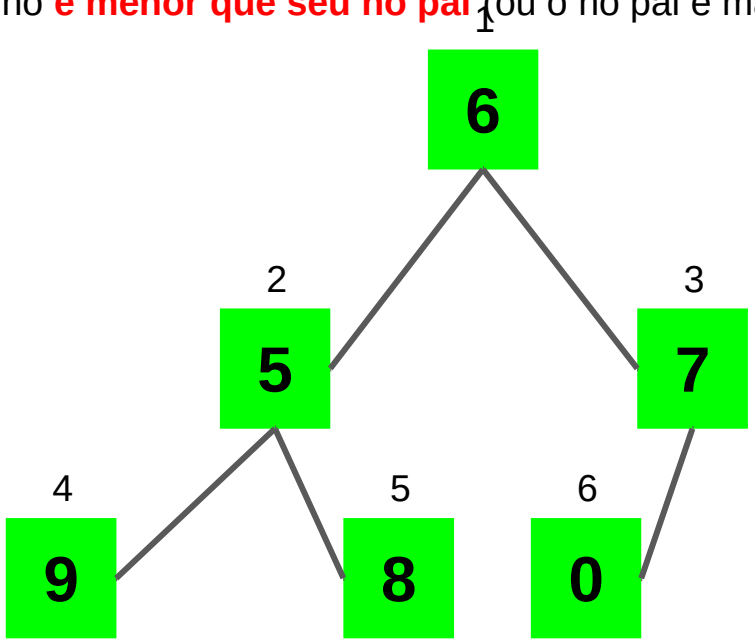
Heap Sort

- Depois… (mas ainda falta...)



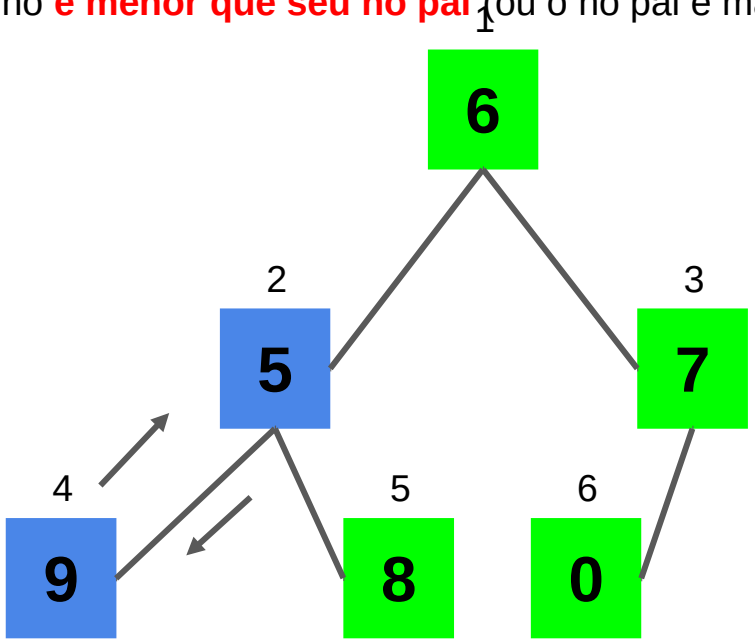
Heap Sort

- Ainda Falta?
- Sim! Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



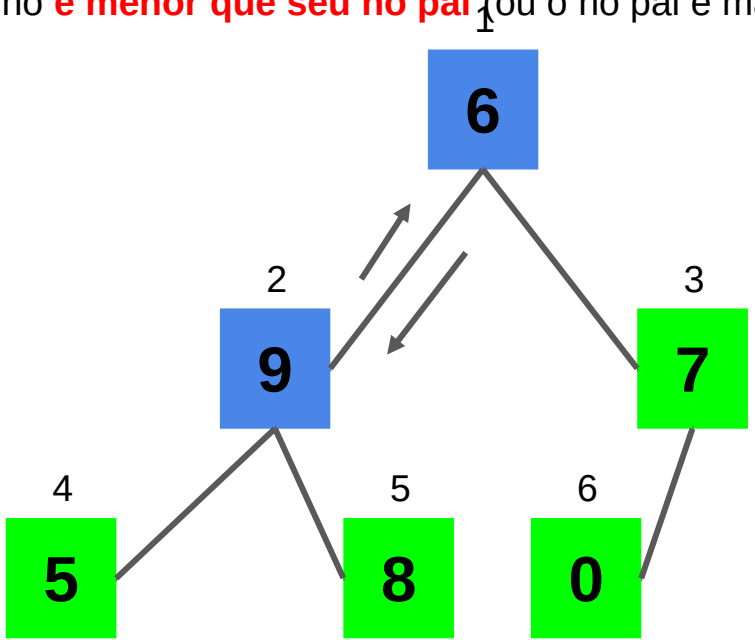
Heap Sort

- Ainda Falta?
- Sim! Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



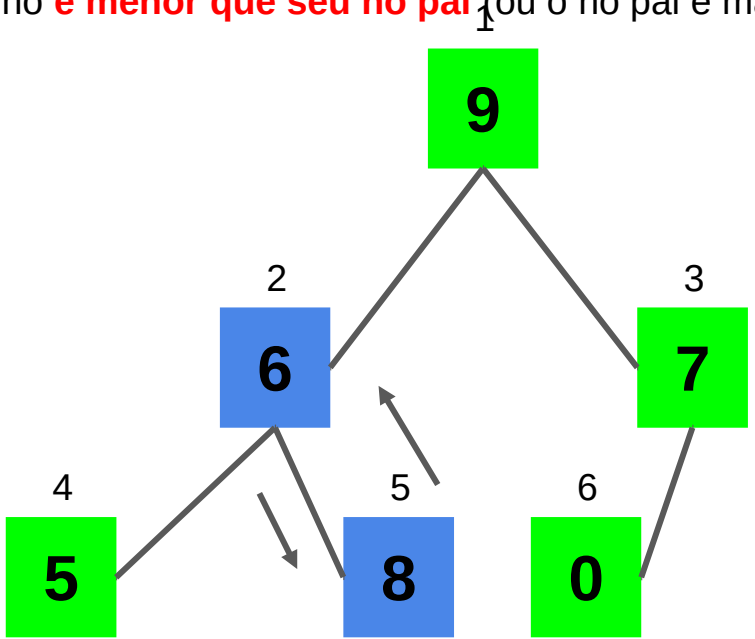
Heap Sort

- Ainda Falta?
- Sim! Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



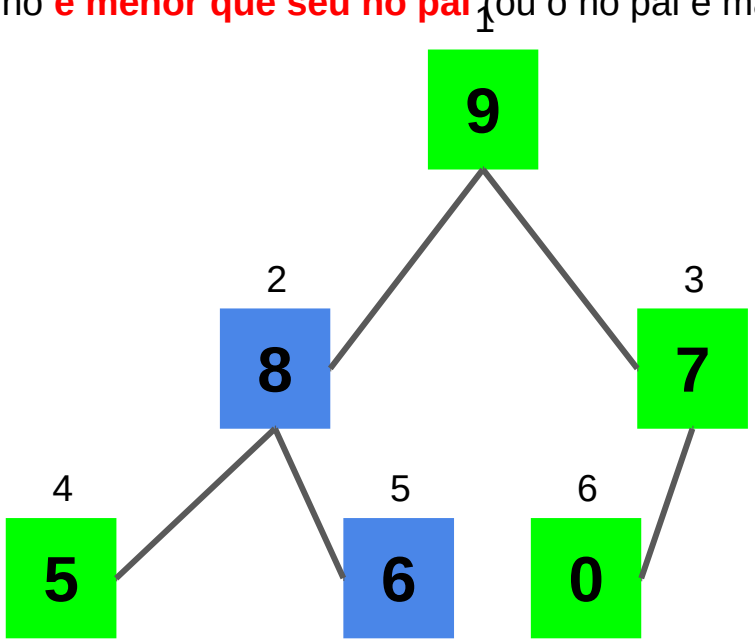
Heap Sort

- Ainda Falta?
- Sim! Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Sort

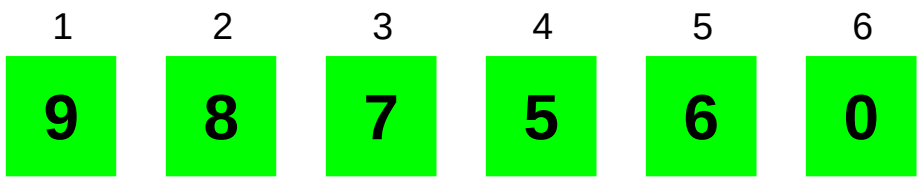
- Ainda Falta?
- Sim! Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Binário!

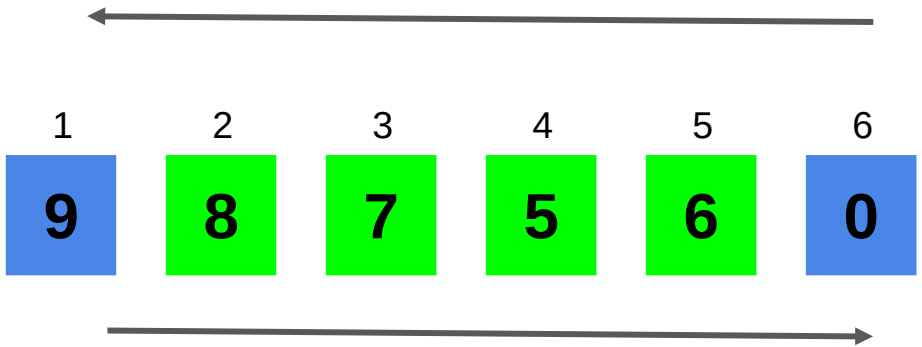
Heap Sort

- Temos então:
- … E ainda não acabou...



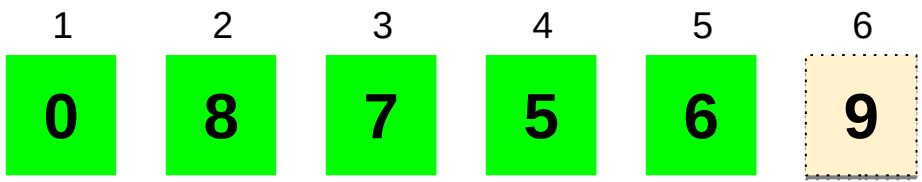
Heap Sort

- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



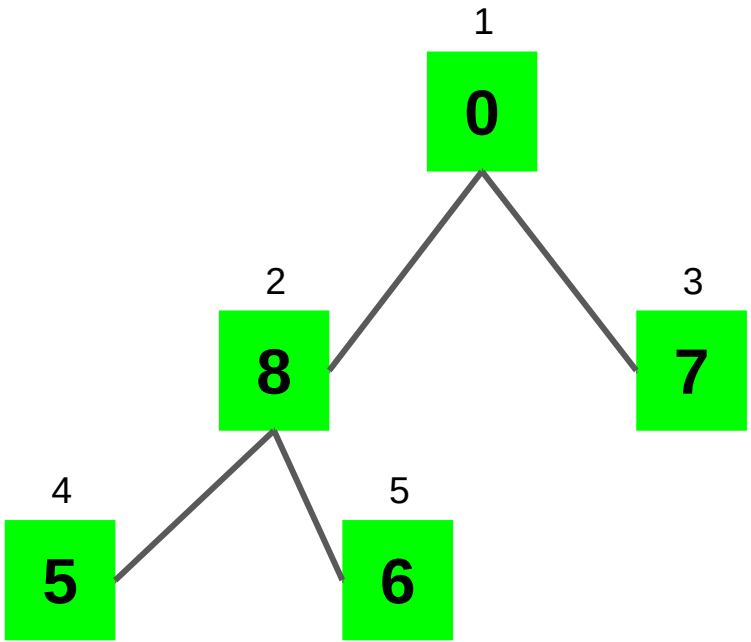
Heap Sort

- Trocado!



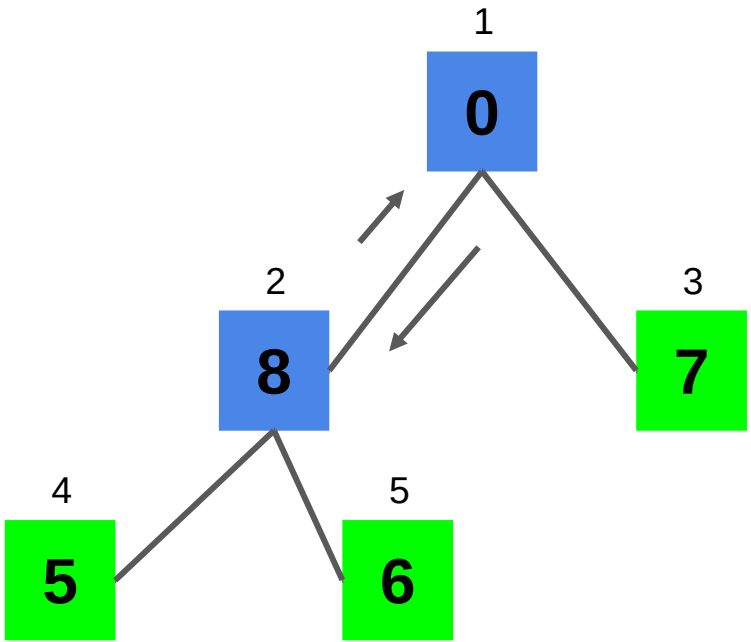
Heap Sort

- Como ficou o Heap



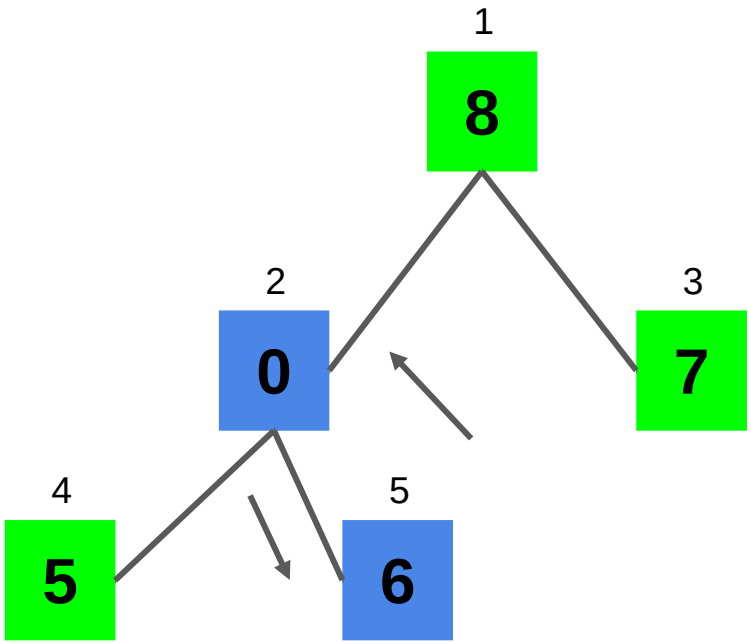
Heap Sort

- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



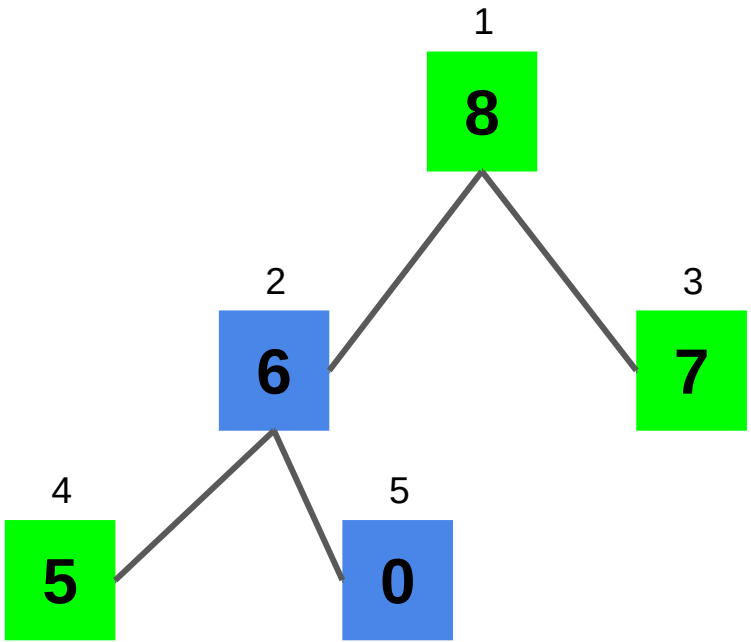
Heap Sort

- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Sort

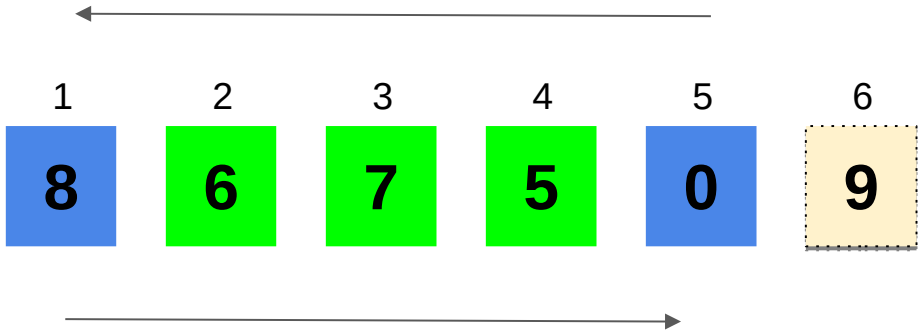
- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Binário!

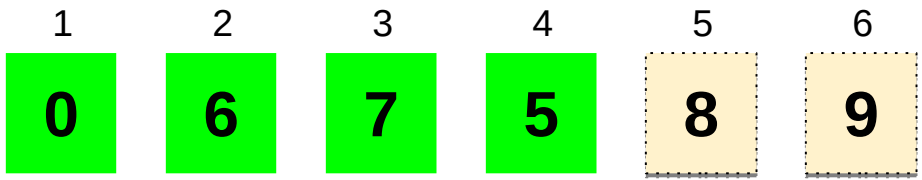
Heap Sort

- Temos no array…
- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



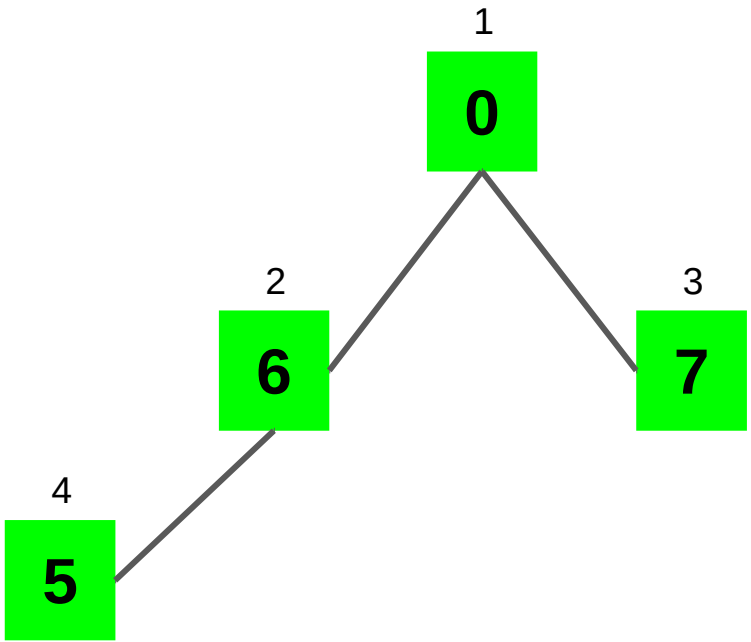
Heap Sort

- Temos no array…
- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



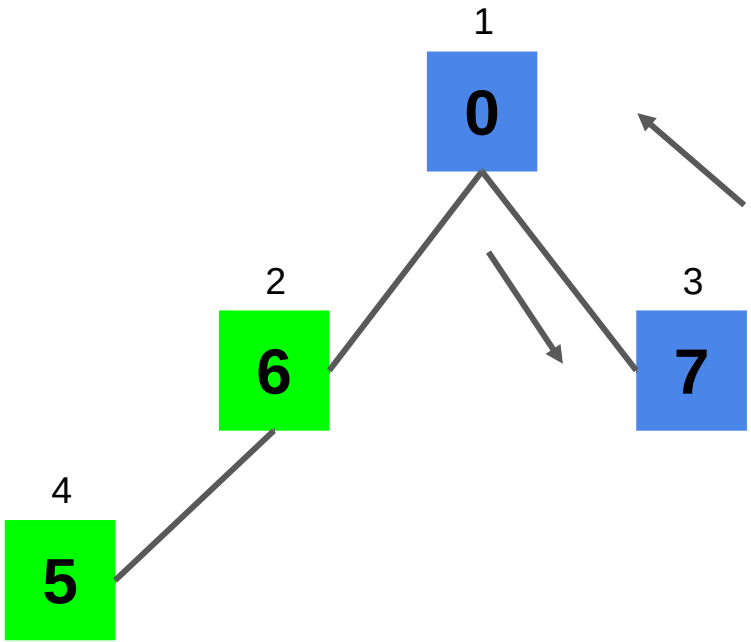
Heap Sort

- Temos no Heap



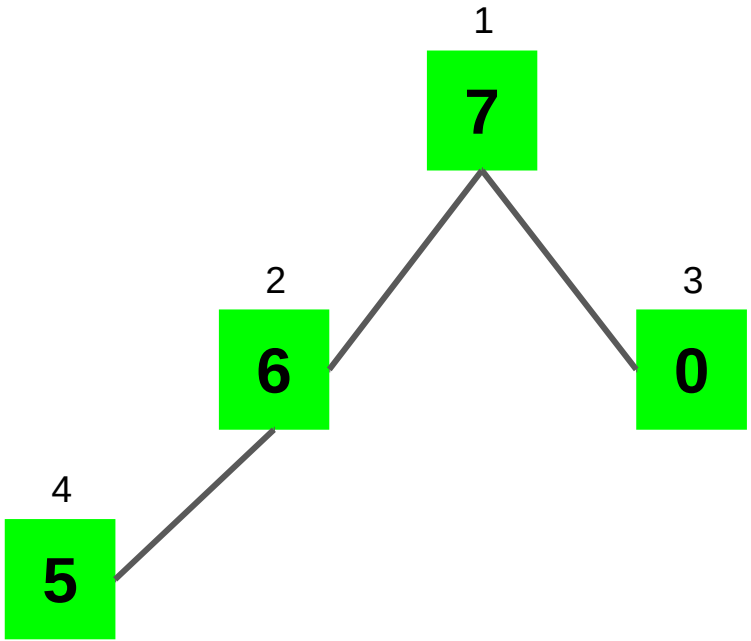
Heap Sort

- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Sort

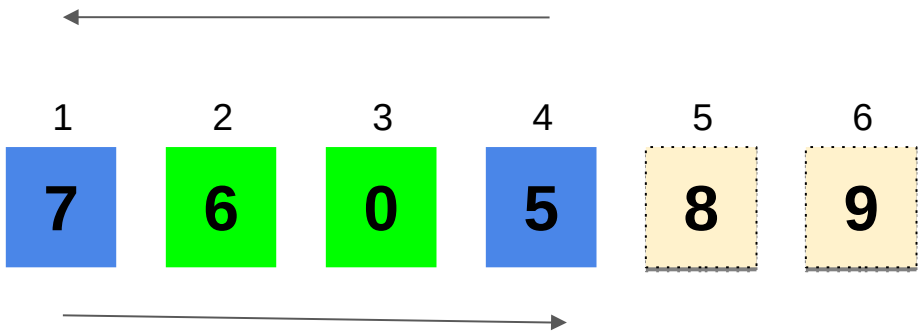
- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Binário!

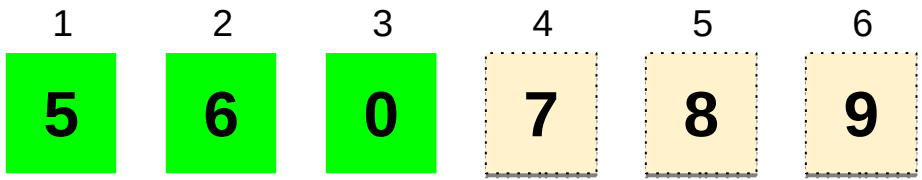
Heap Sort

- Temos no array…
- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



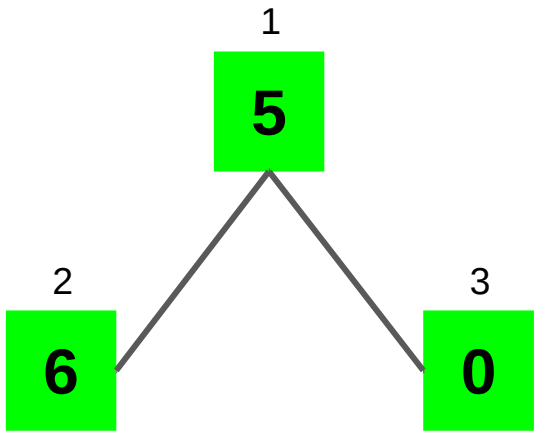
Heap Sort

- Temos no array…
- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



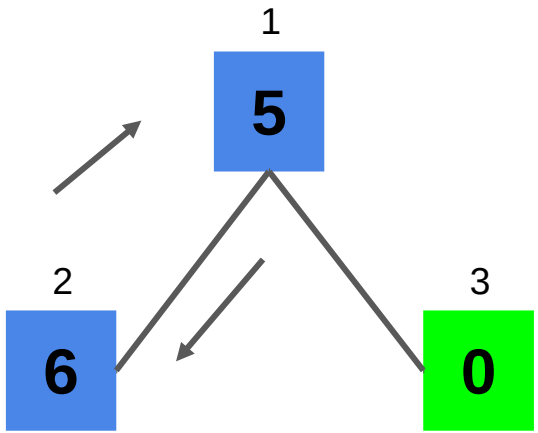
Heap Sort

- Temos no array



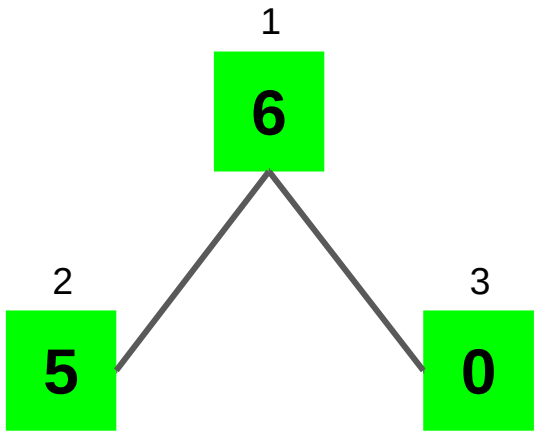
Heap Sort

- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Sort

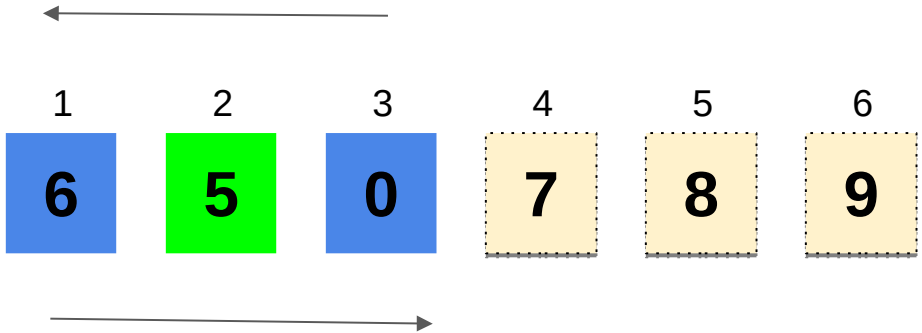
- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Binário!

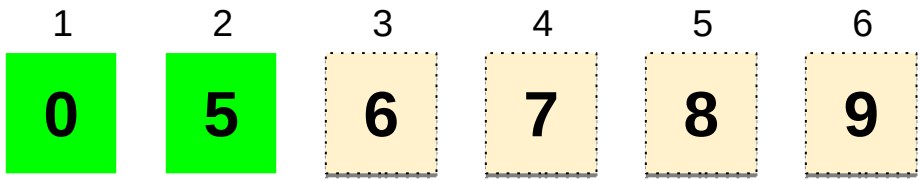
Heap Sort

- Temos no array…
- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



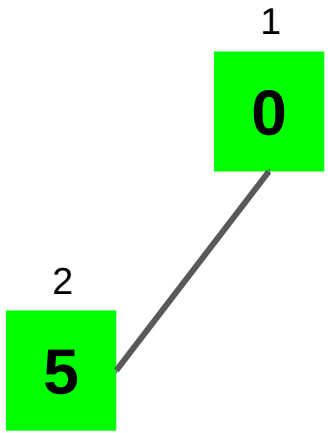
Heap Sort

- Temos no array…
- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



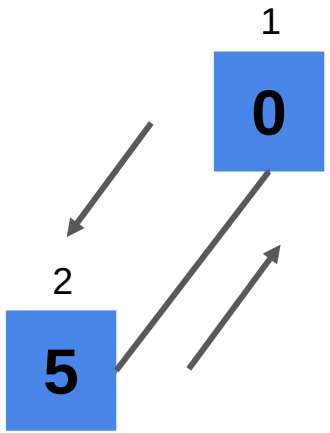
Heap Sort

- Temos no array



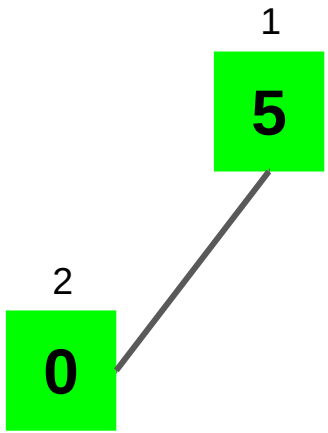
Heap Sort

- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Sort

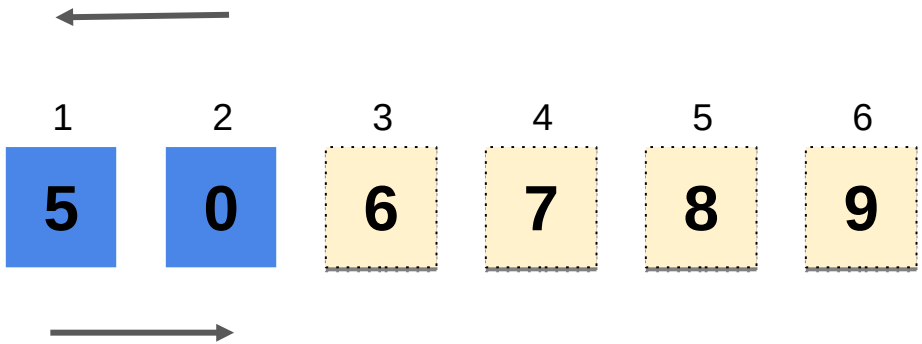
- Precisamos ser um Heap Binário (terceira regra)
  - Cada nó filho **é menor que seu nó pai** (ou o nó pai é maior que seus nós filhos).



Heap Binário!

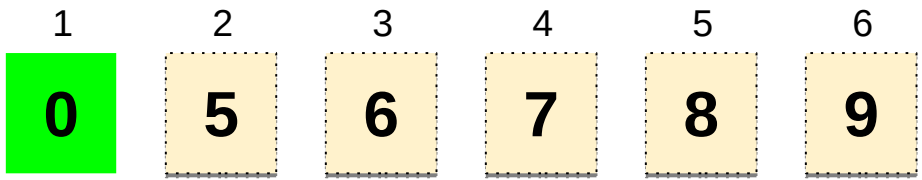
Heap Sort

- Temos no array…
- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



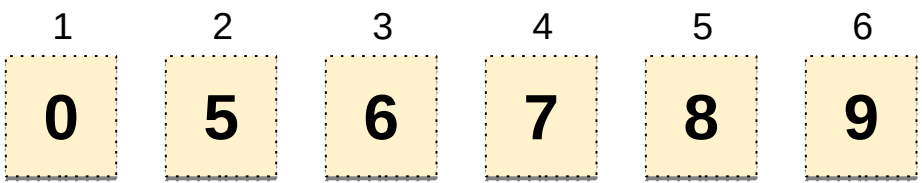
Heap Sort

- Temos no array…
- Troque o primeiro elemento e o último elemento de posições, e “apague” o último elemento do Heap.



Heap Sort

- O algoritmo termina quando há só um elemento no Heap.
- O array está ordenado!



Heap Sort: Análise

- Complexidade: O(nLogn) em todos os casos
- Vantagens:
  - Espaço: Não necessita de vetor auxiliar para a ordenação;
- Desvantagem:
  - Na prática, Quicksort e Mergesort são melhores que ele;

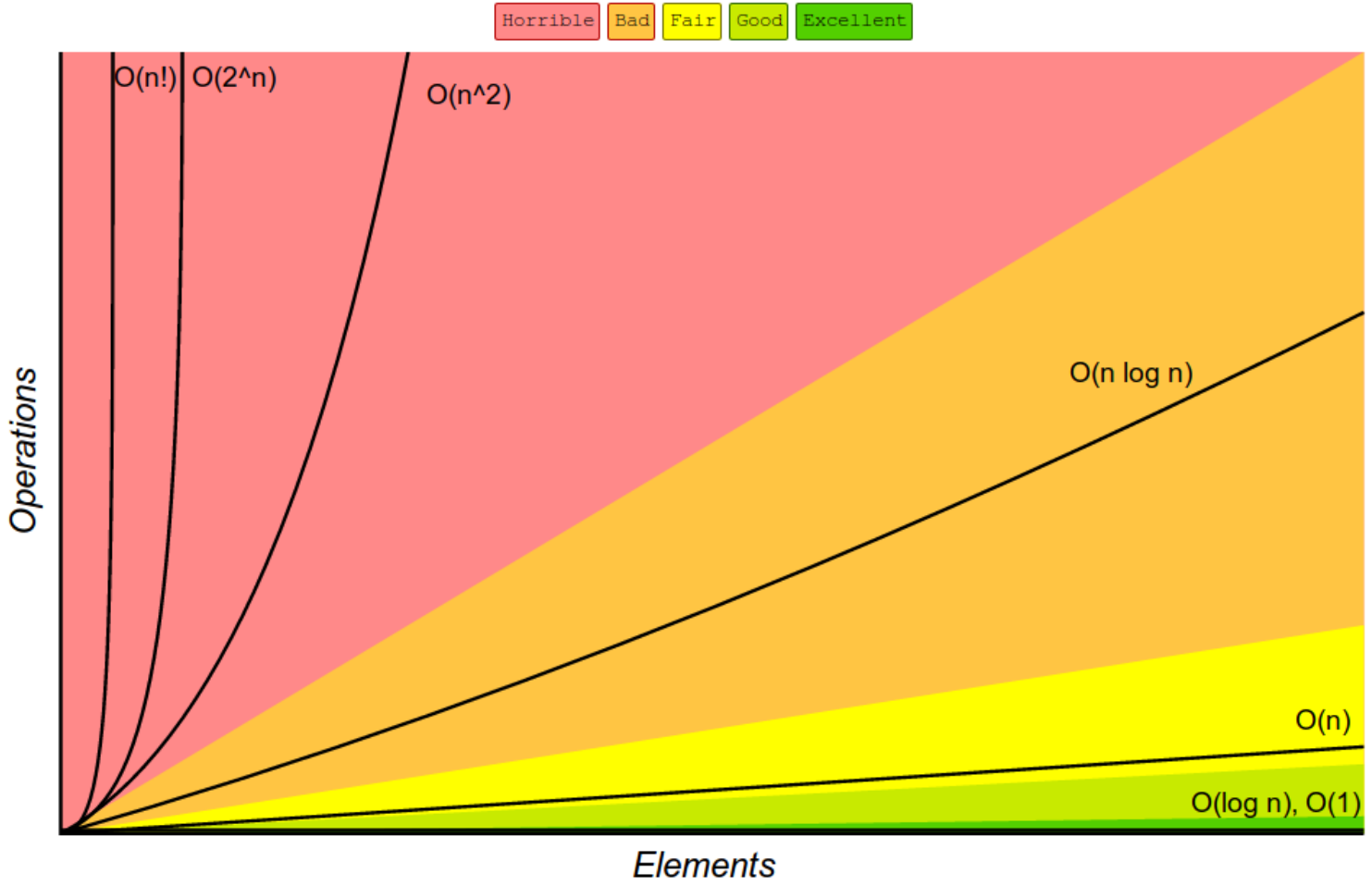
Análise

- É importante termos noção da complexidade dos diversos Algoritmos e suas particularidades.

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

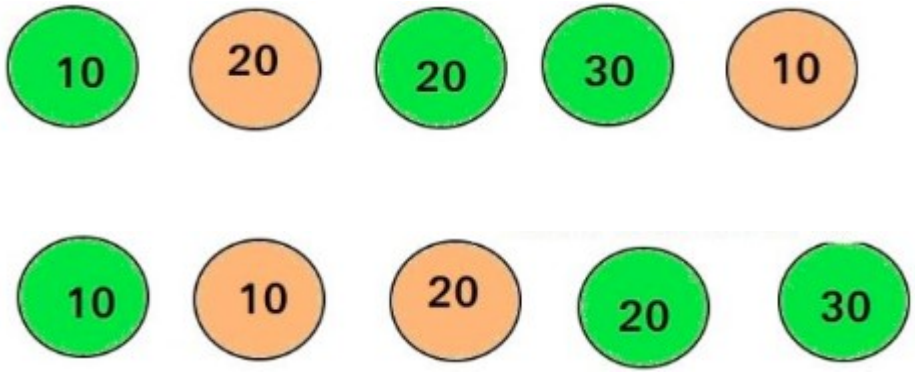
fonte

Análise


Horrible
Bad
Fair
Good
Excellent
O(n!)
O(2^n)
O(n^2)
O(n log n)
O(n)
O(log n), O(1)
Operations
Elements

Estabilidade de um Algoritmo

- Algebricamente
  - Um algoritmo é dito estável se: i < j E A[i] ≡ B[j] $\rightarrow \pi$[i] < π[j], onde π é a permutação de ordenação;
- Possível tradução
  - Elementos equivalentes mantém suas posições relativas após ordenação.



Estabilidade de um Algoritmo

(Dave, A)
(Alice, B)
(Ken, A)
(Eric, B)
(Carol, A)

Por nome

(Alice, B)
(Carol, A)
(Dave, A)
(Eric, B)
(Ken, A)

Seções fora

Por seção (Hipótese)

(Carol, A)
(Dave, A)
(Ken, A)
(Eric, B)
(Alice, B)

Se fosse Estável...

(Carol, A)
(Dave, A)
(Ken, A)
(Alice, B)
(Eric, B)

BEFORE	
Name	**Grade**
Dave	C
Earl	B
Fabian	B
Gill	B
Greg	A
Harry	A

Estável

AFTER	
Name	**Grade**
Greg	A
Harry	A
Earl	B
Fabian	B
Gill	B
Dave	C

BEFORE	
Name	**Grade**
Dave	C
Earl	B
Fabian	B
Gill	B
Greg	A
Harry	A

Instável

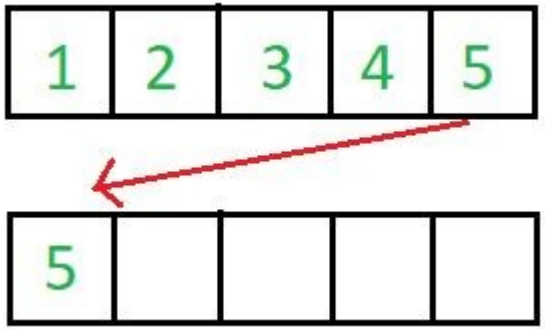
AFTER	
Name	**Grade**
Greg	A
Harry	A
Gill	B
Fabian	B
Earl	B
Dave	C

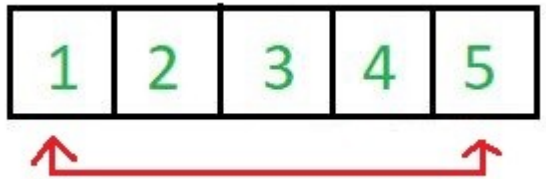
Estabilidade de um Algoritmo

- Estáveis por natureza: MergeSort, BubbleSort, InsertionSort
- Instáveis: Heapsort, Quicksort (mas podem ter implementações estáveis)

In Place

- Um algoritmo é dito "In-place" quando não é usado espaço extra para sua execução.
  - Obs: Pequena tolerância: Espaço para constantes de controle pode ser utilizados sem violar a característica In-place.






Thank You!