Quicksort, Mergesort e Heapsort

Quicksort

Charles Antony Richard Hoare - 1960

Apresenta a estratégia de dividir e conquistar, dividir o problema em subproblemas menores que possam ser resolvidos mais rapidamente.

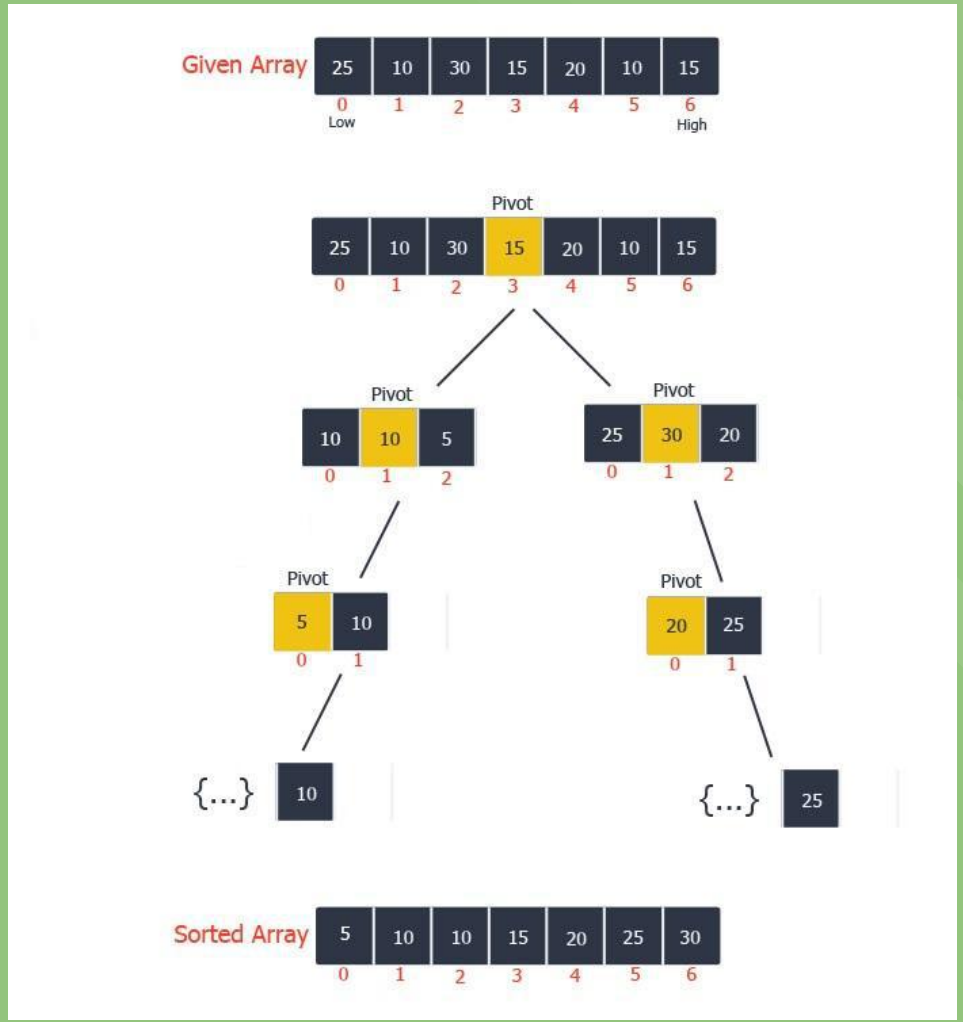
- Uso de um pivô

Sua complexidade média e mínima é $O(n \log n)$
Sua complexidade máxima é $O(n^2)$

Passo 1: Escolher um pivô, na metade do vetor.

Passo 2: No sentido das pontas até o pivô, verificar se existem números que são menores ou maiores que o pivô e trocá-los.

Passo 3: Depois que a análise passar pelo pivô, recursivamente fazer a mesma coisa com as duas metades.


Given Array
25 10 30 15 20 10 15
0 1 2 3 4 5 6
Low
High
Pivot
25 10 30 15 20 10 15
0 1 2 3 4 5 6
Pivot
10 10 5
0 1 2
Pivot
25 30 20
0 1 2
Pivot
5 10
0 1
Pivot
20 25
0 1
{...}
10
{...}
25
Sorted Array
5 10 10 15 20 25 30
0 1 2 3 4 5 6

6 5 3 1 8 7 2 4

Vantagens:

- Contido em si mesmo
- Não usa muita memória

Desvantagens:

- No pior caso, tem a pior performance

Mergesort

John Von Neumann - 1945

Também apresenta a estratégia de dividir e conquistar.

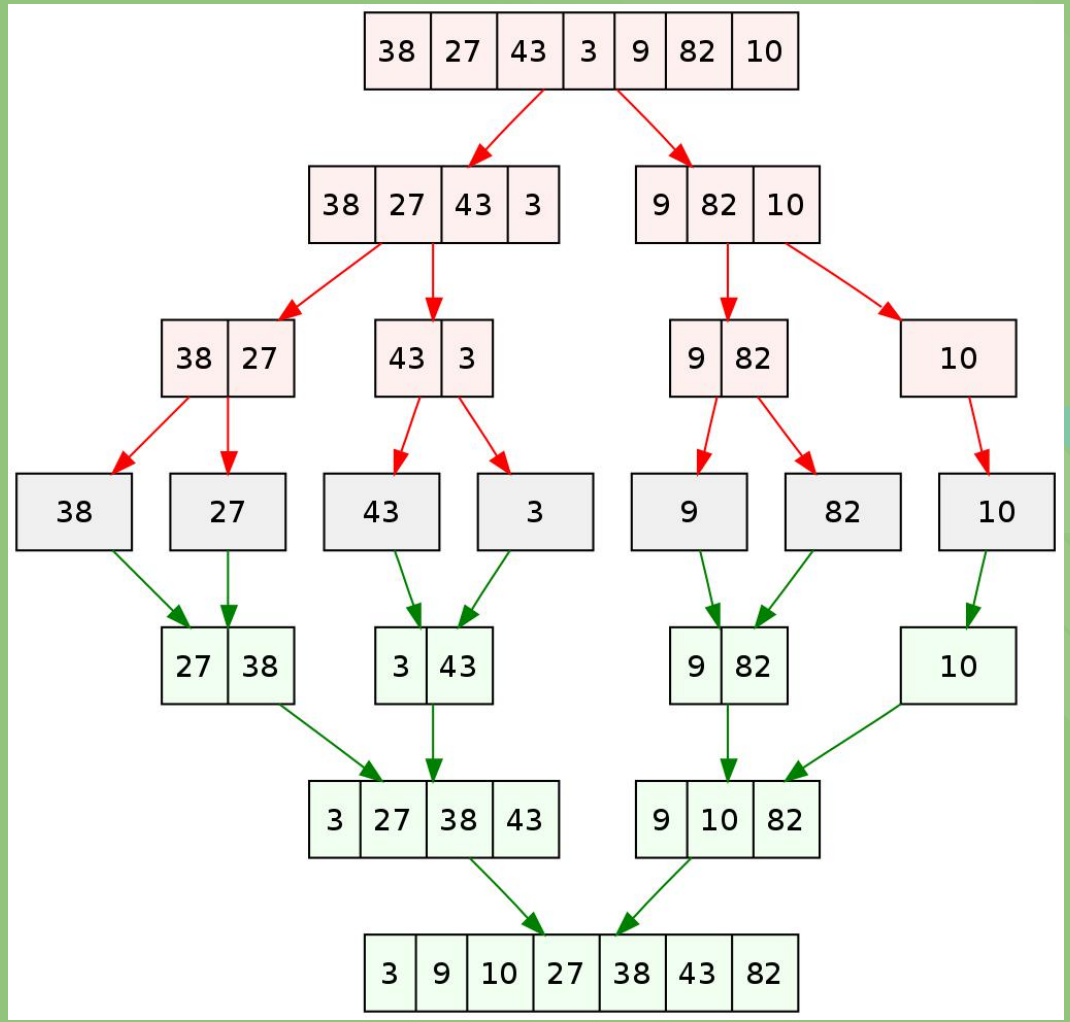
- Não usa um pivô

- Sua complexidade mínima é O(n)
  Sua complexidade média é O(n log n)
  Sua complexidade máxima é O(n log n)

Passo 1: Recursivamente dividir o vetor até sobrar unidades;

Passo 2: Supõe-se que quando sobram apenas unidades as mesmas estão ordenadas;

Passo 3: Após isso realizar a ordenação com a unidade adjacente (merge).


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

6 5 3 1 8 7 2 4

Vantagens:

- Sua complexidade não é das piores
- Geralmente é rápido, mas pode demorar em casos piores.

Desvantagens:

- Usa recursão extensivamente, o que consome muita memória

Heapsort

Robert W. Floyd e J.W.J Williams - 1964

Apresenta a estratégia de diminuir o espaço não organizado. É um algoritmo de seleção e não de comparação que nem os outros

Sua complexidade média é O(n log n)

Passo 1: Tratar o vetor trabalhado como um heap/árvore
Passo 2: Através de um algoritmo chamado HEAPFY, os maiores números da árvore flutuarão para cima da árvore(esquerda do vetor)
Passo 3: O maior número(que está à esquerda) será trocado com o último
Passo 4: Diminui-se o vetor trabalhado em 1(pela direita)
Passo 5: Voltar para o passo 2 até que o passo 4 seja impossível.

No vetor/Heap

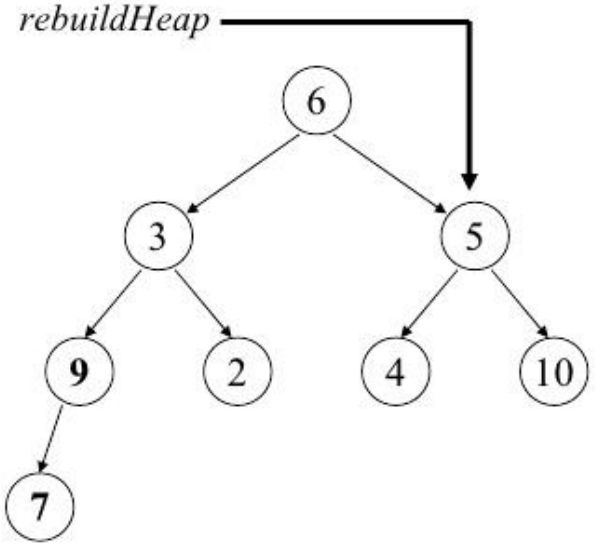
Nó: n

Filho esquerdo: n*2 + 1

Filho direito: n*2 + 2

Transform an Array Into a Heap: *Example*

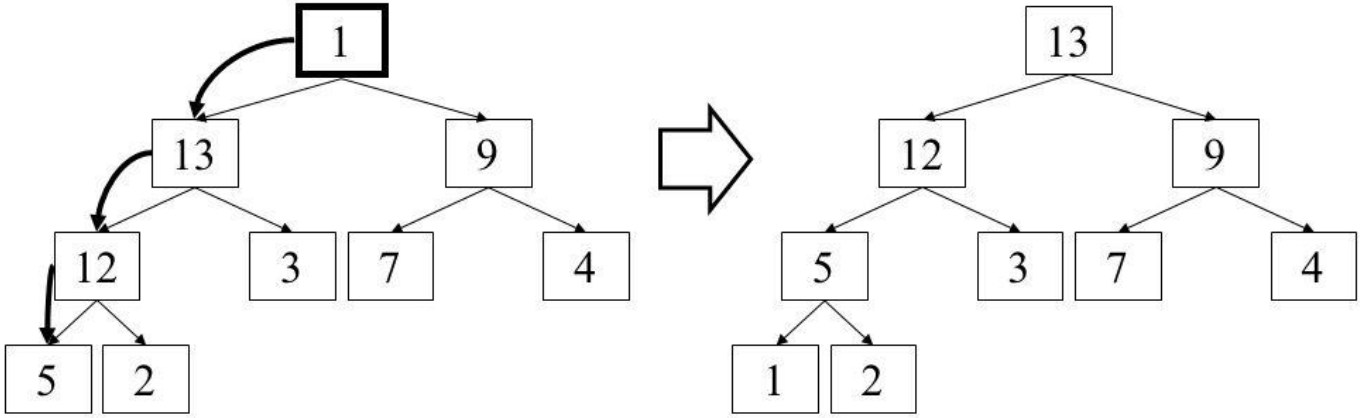
6	3	5	**9**	2	4	10	**7**
0	1	2	3	4	5	6	7



- Note that nodes 2, 4, 7, 9 & 10 are roots of *heaps;* nodes 3 & 5 are roots of *semiheaps*.
- *rebuildHeap* is invoked on the node in the array *preceding* node 9.

Heapify

```
Heapify(Node x)
      largest = max {left(x), right(x)}
      if ( largest > x )
            exchange (largest, x)
            heapify (x)
```



6 5 3 1 8 7 2 4

Vantagens:

- Tem uma boa eficiência dentro do grupo de Selection Sort
- Complexidade não é ruim
- Contido em si mesmo

Desvantagens:

- Um quicksort funcionaria mais rápido, geralmente

Fim :^)