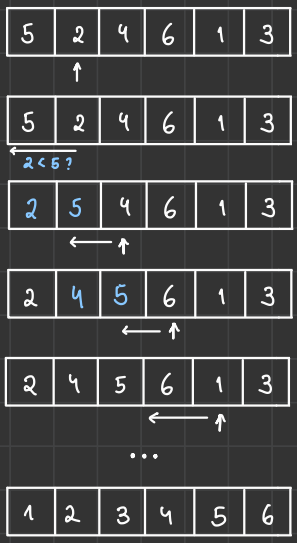
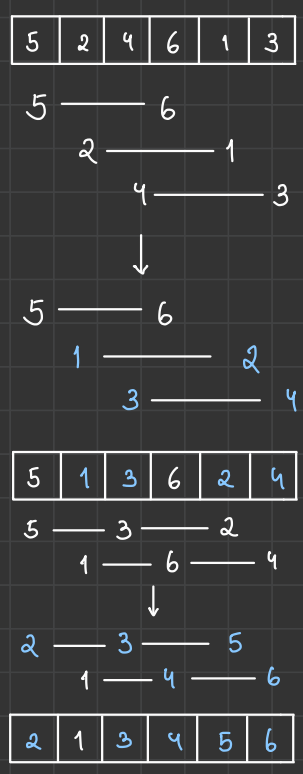
Insertion Sort



Complexidade Temporal:

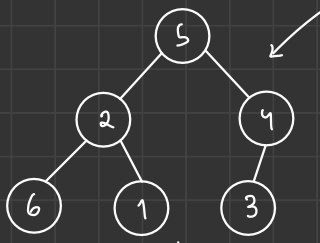
- melhor caso: $O(n)$ caso já esteja ordenado.
- caso médio e pior caso: $O(n^2)$

Shell Sort



Heap Sort

5	2	4	6	1	3



$O(n \log n)$

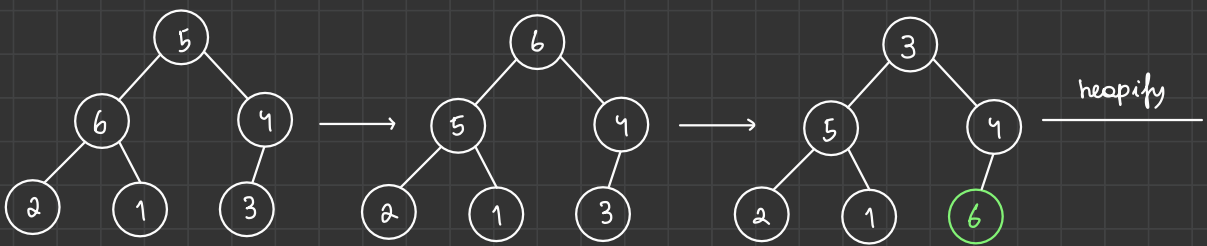
Complexidade temporal:

- Pior caso (construção da heap): $O(n.\log(n))$
- Primeira fase: $O(n)$
- Segunda fase: $O(n)$

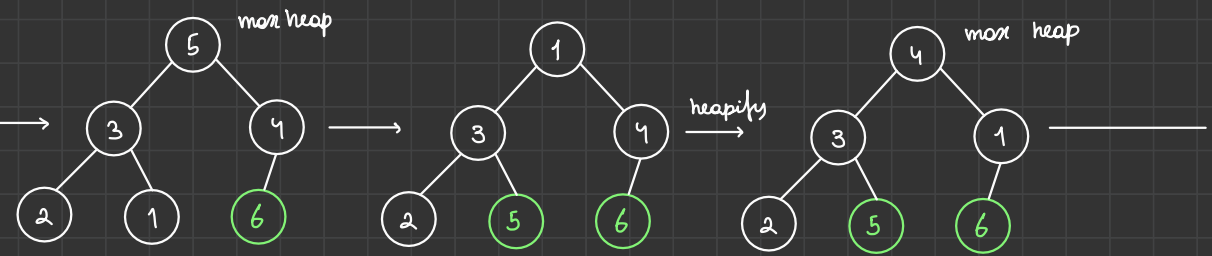
- o parent node é sempre maior ou igual aos filhos.
- trocar o último com o primeiro e remove-lo.
- Não é garantido ordenamento na horizontal

→ Ordenada de cima para os filhos.

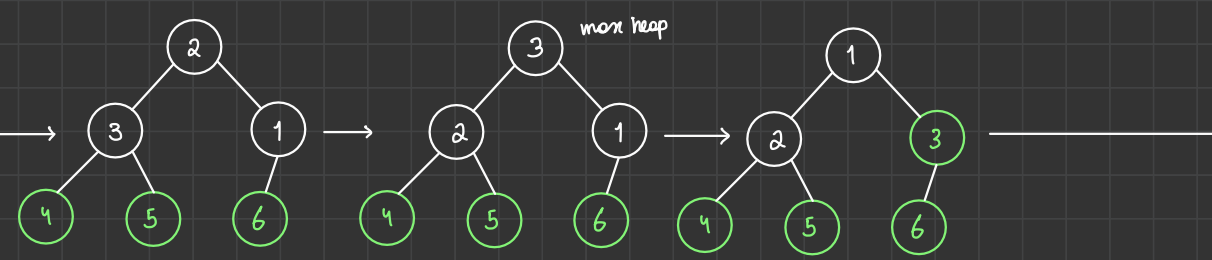
max heap



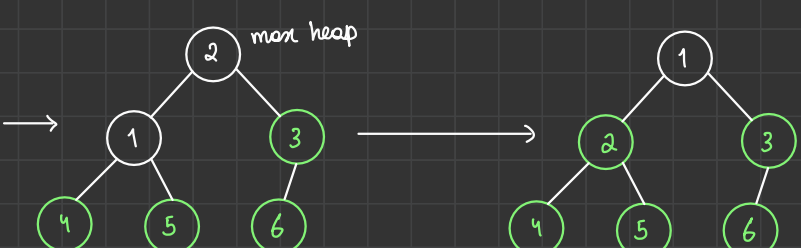
max heap



max heap



max heap



Radix Sort

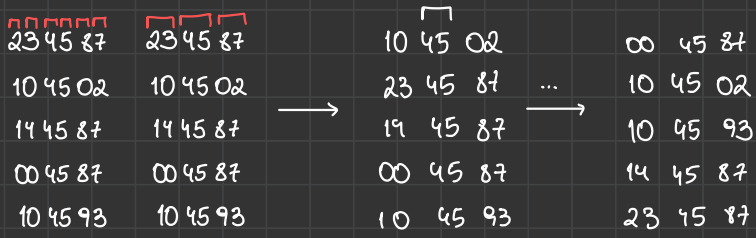
LSD (least significant digit) "começar pelo fim"

MSD (most significant digit) "começar pelo início"

Complexidade temporal:

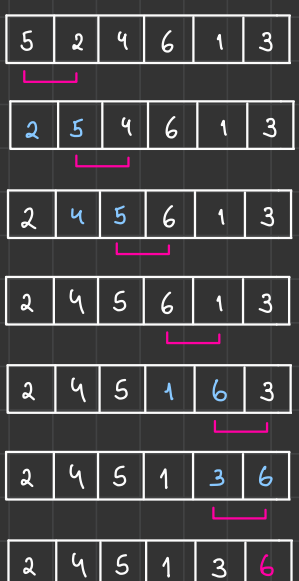
- base k
- $\rightarrow O(n . \log_k(n))$

base 10 | base 100

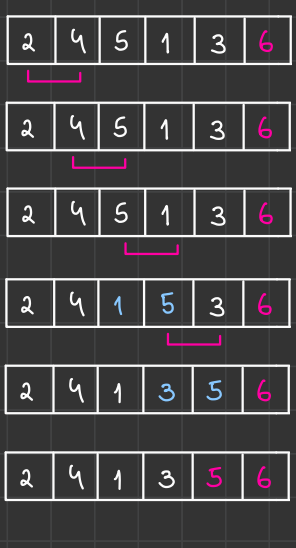


Bubble Sort

1ª iteração:



2ª iteração:



Complexidade:

- Pior caso : $O(n^2)$
- Melhor caso : $O(n)$

Selection Sort

↑ : current

↑ : current minimum

Complexidade:

- Pior caso : $O(n^2)$
- Melhor caso : $O(n)$

Quick Sort

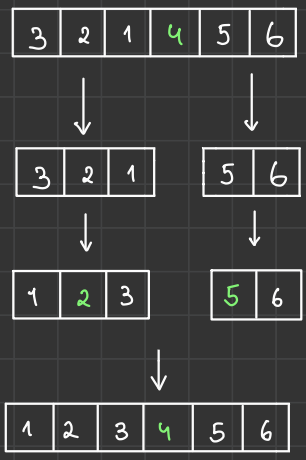
Escolha do Pivot:

- Selecionar o primeiro, o do meio e o final, ordena-los e escolhemos o do meio.

- procurar o 1º nº maior que o pivot à esquerda
- procurar o 1º nº menor que o pivot à direita

- Repetir até index do esquerdo for maior que o index do direito ⟶ trocar pivot com o maior à esquerda

Quick Sort

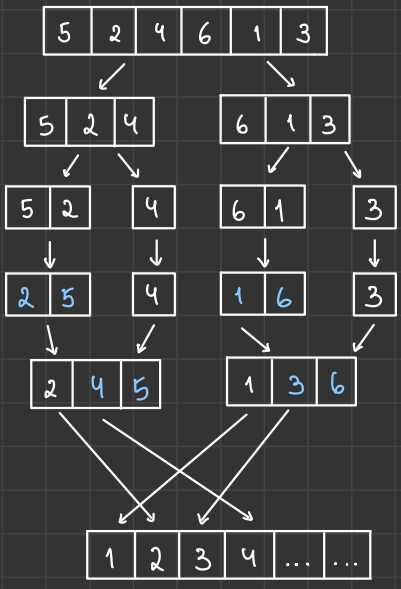


- repetir nas partições

Complexidade:

- Pior Caso: $O(n^2)$
- Caso Médio: $O(n \log n)$

Merge Sort



$6 // 2 = 3$

Complexidade:

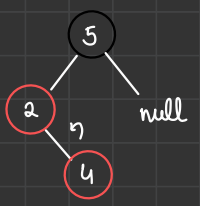
- Pior caso: $O(n \log(n))$
- $O(n \log(n))$

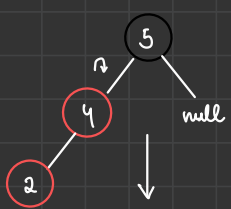
Árvore Vermelha Preta

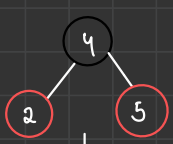
$O(\log n)$

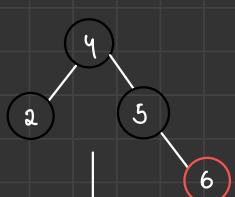
5	2	4	6	1	3

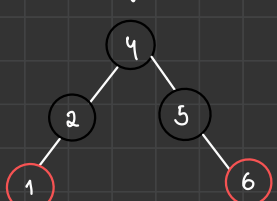
- Um node só pode ser preto ou vermelho
- A raiz e folhas Null são pretas
- Se um node é vermelho, os seus filhos são pretos





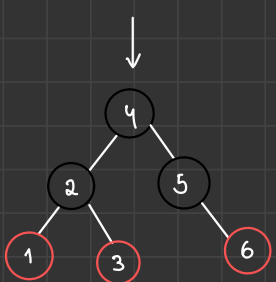






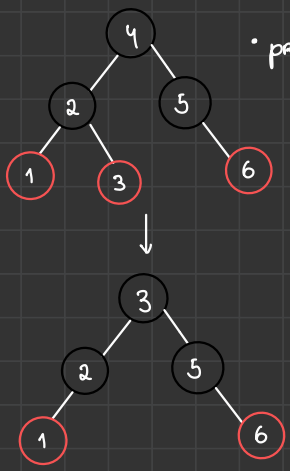
Inserção:

- O novo node é sempre vermelho
- Se o pai for preto, não alterar nada
- Se o pai for vermelho:
  - → se o irmão do pai for preto ou null: Rotações e Recolor
  - → se for vermelho, Recolor:
    o pai fica vermelho, se não for root, Recolor



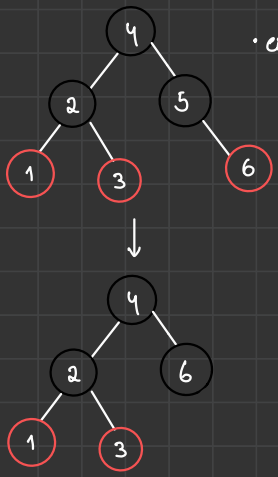
Árvore Vermelha Preta

Delete(4):



- procurar o maior à esquerda: 3

Delete(5):



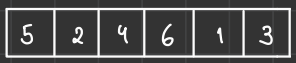
- como não tem filho à esquerda, ligar ao direito

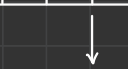
Delete fixer:

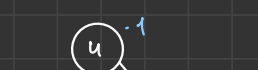
1. O nó substituído é vermelho ou a raiz é vermelha, pintar o nó de preto.
2. O nó substituído e o seu irmão são pretos, e os filhos do irmão são pretos, o irmão é pintado de vermelho
3. O nó substituído é preto, o irmão é preto, mas os filhos são coloridos: Rotação e ajuste às cores
4. O nó é preto, o irmão é preto, e pelo menos um filho do irmão é vermelho: Rotação e ajuste

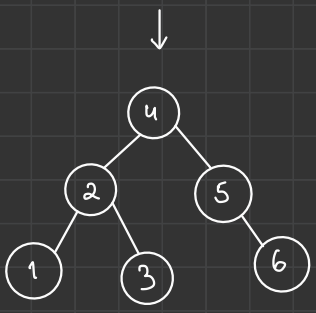
Arvore AVL

balance factor = tamanho da subtree esquerda - tamanho da subtree direita
Considera-se balanceada se o valor estiver entre {-1, 0, 1}







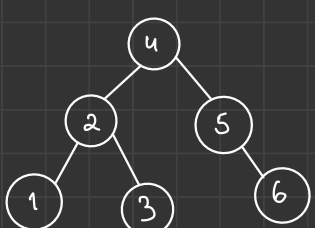


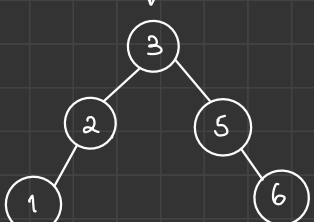
Complexidade:
- O(log n) em todos os casos

Nota:
- Funciona como uma binary search tree, novo elemento é adicionado à esquerda se for menor, caso contrário é adicionado à direita.

Delete:
- Ao remover a raiz, troca pelo maior elemento à esquerda

Delete (4):





SPLAY Tree