PO (Pesquisa e Ordenação)

- Java
- POO (programação orientado objeto)
- Pesquisa: Métodos de busca e árvores
  não binária (string e números)

Algoritmos: Busca Binária, Trie, N-ória, B+, B

- Ordenação: Bubble, Merge Sort, Radix

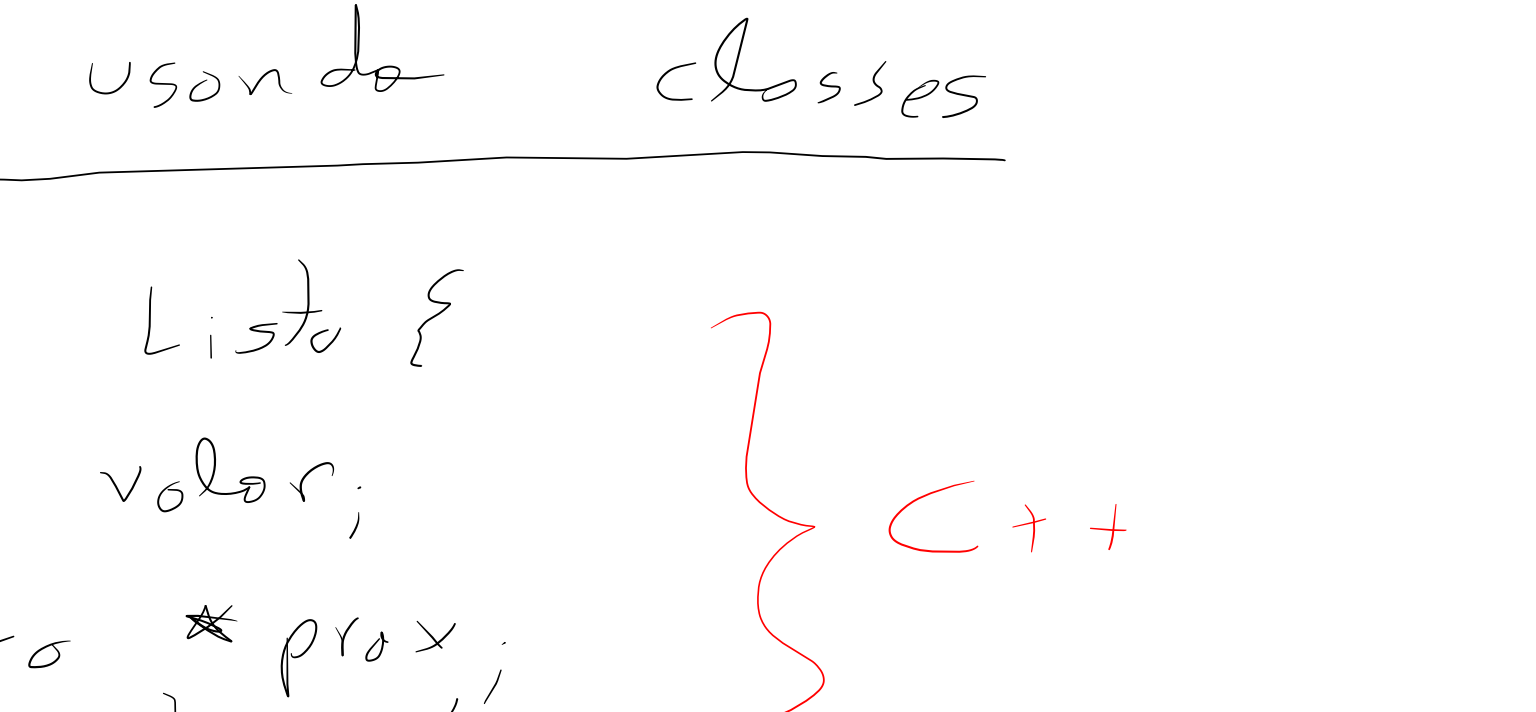
# Lista usando classes

```
struct Lista {
    int valor;
    Lista *prox;
};
```

C++

classe

Java:

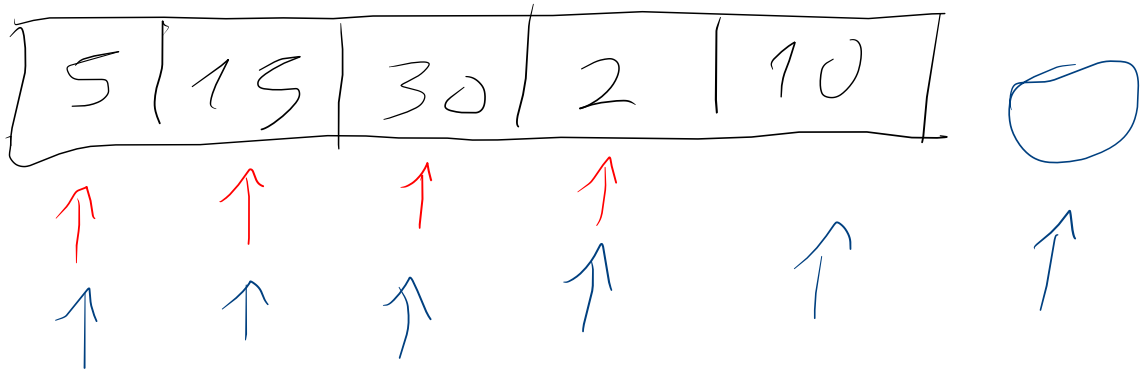


Tarefa:

- Lista ordenada
- Fila
- Pilha

# Buscas

## 1. Exaustivo / Linear




19 = 5
19 = 15
19 = 30
19 = 2
19 = 10

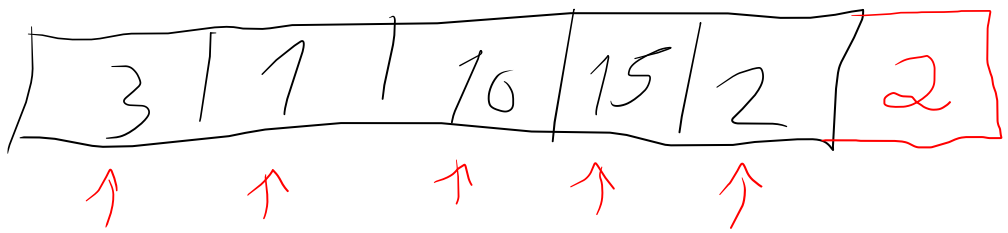
Não achou

2 = 5
2 = 15
2 = 30
2 = 2 ✓

Busca Linear:

- Passo por cada posição do array (comparando)
- Se chegou no final do array: Não achou
- Senão: Achou

2. Sentinela



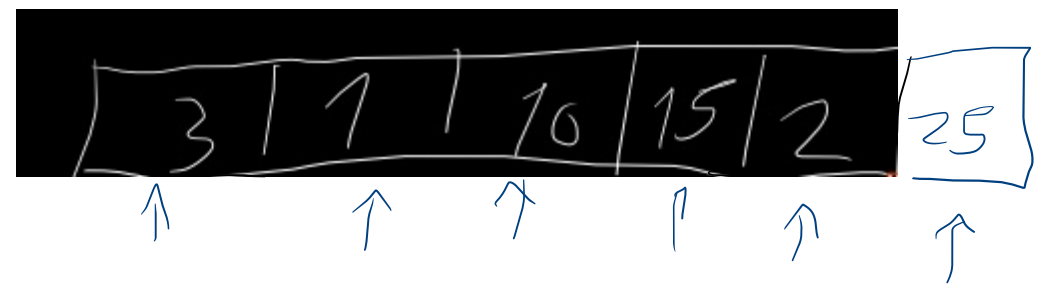

2 = 3

2 = 1

2 = 10

2 = 15

2 = 2 ✓




25 = 3

25 = 1

25 = 10

25 = 15

25 = 2

25 = 25 (Falso, achou sentinela)

Busca Sentinela:

- Adicionar o sentinela (valor no final)
- Passar pelas posições do array (comprio)
- Se achou o sentinela: Não existe
- Senão: Achou

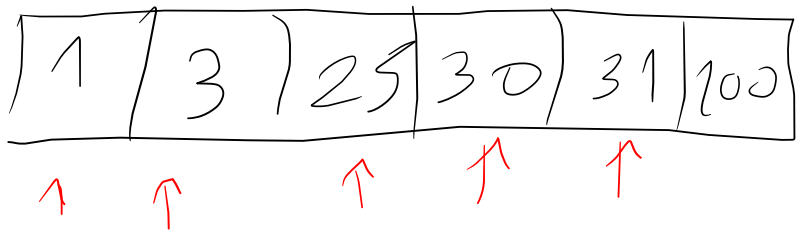
3. Indexada

Observação: Só vale para vetores ordenados

Vantagem: Se em algum momento o meu valor for menor do que a posição eu posso parar a busca.

$$arr[0] \leq arr[1] \leq arr[2] \dots \leq arr[n]$$

$$20 \leq arr[2]$$



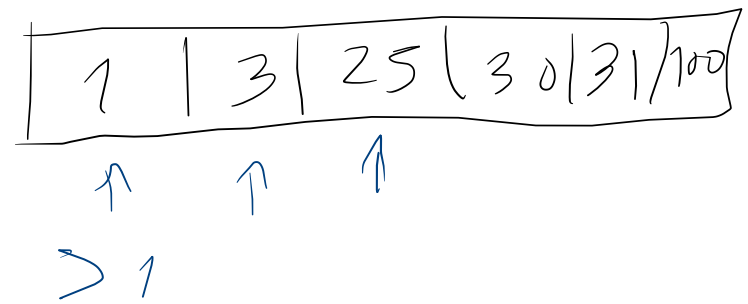

31 > 1

31 > 3

31 > 25

31 > 30

31 > 31 (Para) => 31 = 31 ✓




17 > 1

17 > 3

17 > 25 (Para) => 17 = 25 (Falso)

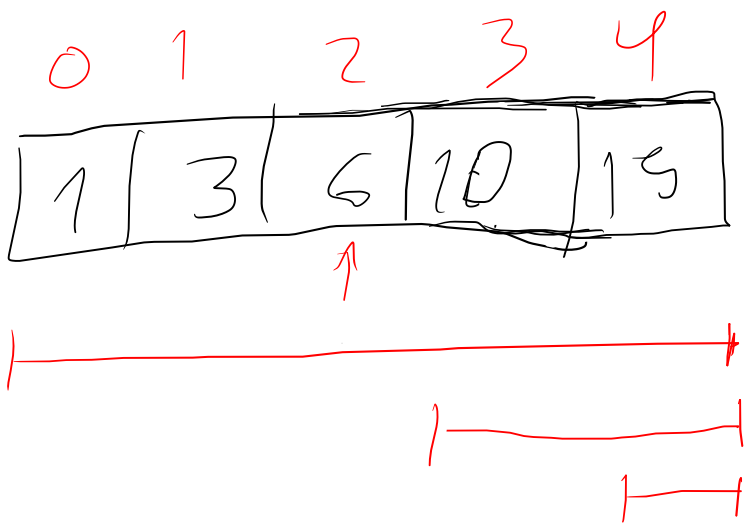
Busca Indexada

- Passo por todas as posições que arr[i] é menor que o buscado
- Se parar e for igual: achou
- Senão: não achou

# 4. Busca Binária

Observação: O array tem que está ordenado.

Vantagem: Busca mais rápido O(log n)




15 = 6

15 = 10

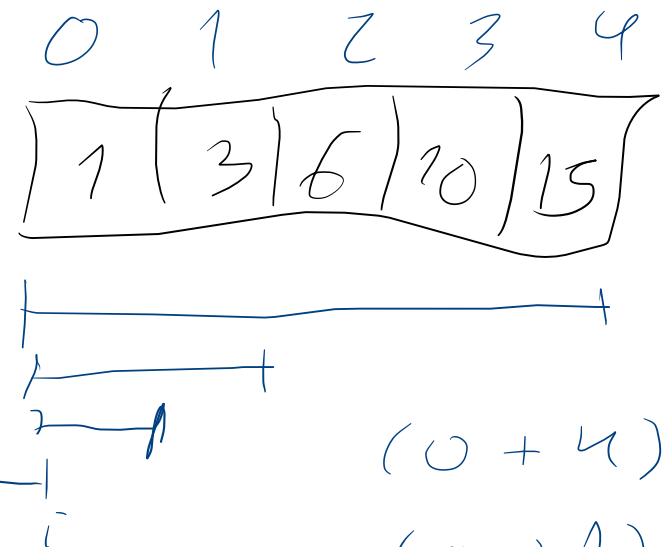
15 = 13 ✓

(0 + 4) / 2 = 2

(3 + 4) / 2 = 3

(4 + 4) / 2 = 4

ini ≤ fim




0 = 6

0 = 1

0 = 1

(0 + 4) / 2 = 2

(0 + 1) / 2 = 0

(0 + 0) / 2 = 0

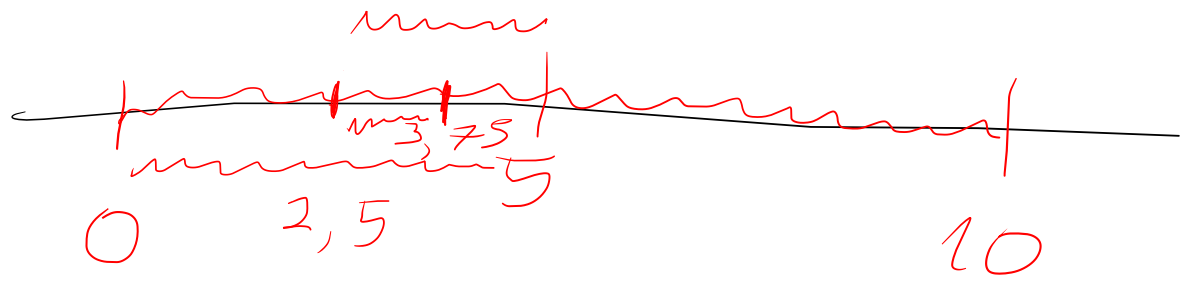
(0 + -1)

BuscBinaria:

- Pego inicio e fim
- Verifico o meio do intervalo:
  - Se for igual: achei
  - Se for maior: mudo inicio
  - Se for menor: mudo o meio
- Se inicio > fim, para: Não achei

Aplicação: Achar raiz quadrado.

$\sqrt{9} = ? \; y \Rightarrow y^2 = 9$




$5^2 > 9$ (V)

$2,5^2 > 9$ (F)

$3,75^2 > 9$ (V)

$3,125^2 > 9$ (V)

$2,81^2 > 9$ (F)

$x > y \Rightarrow \sqrt{x} > \sqrt{y}$

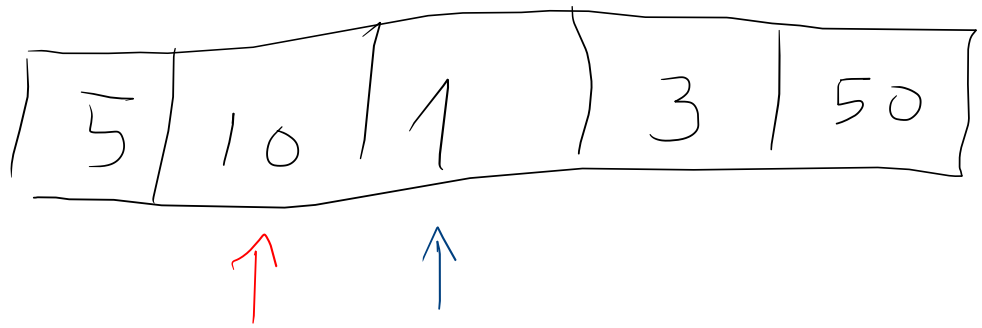
$100 > 25 \Rightarrow \sqrt{100} = 10 > \sqrt{25} = 5$

$[0,10], [0,5], [2.5,5], [2.5,3.75], [2.5,3.125], [2.81,3.125]$

Tarefa: Implementar os busca em lista.

# Ordenação

## 1. Bubble Sort




5 > 10 (F)

10 > 1 (V) → Troca

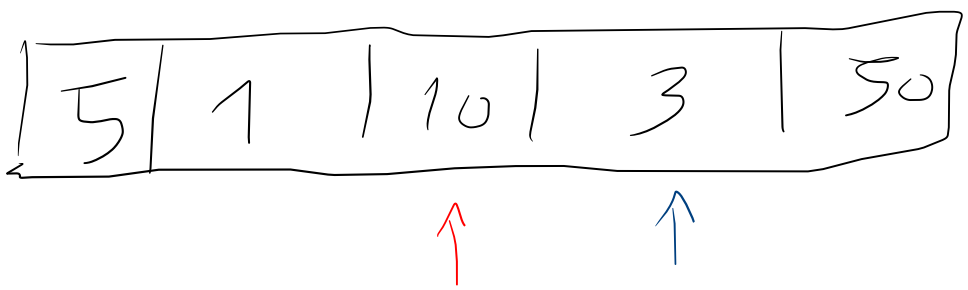
10 > 3 (V) → Troca

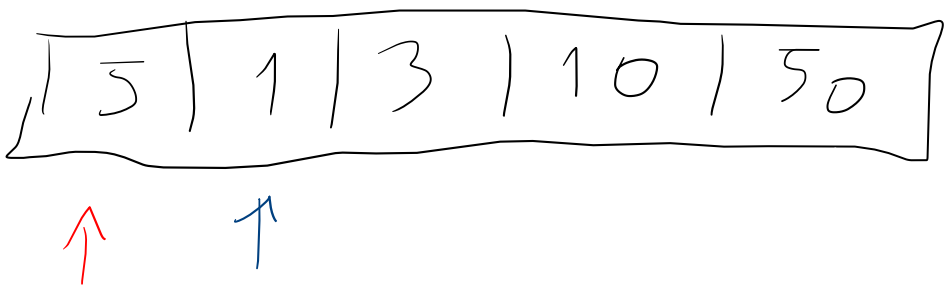
10 > 50 (F)

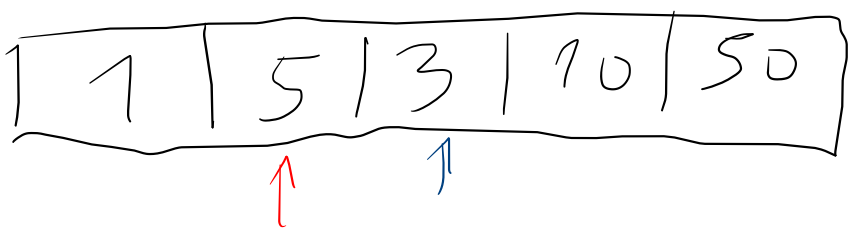
5 > 1 (V) → Troca

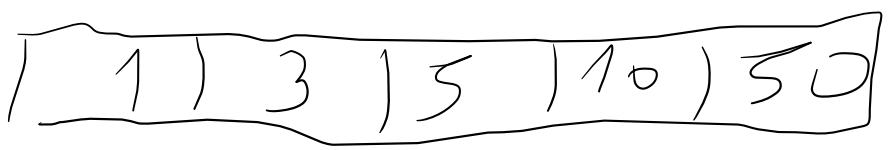
5 > 3 (V) → Troca

Ordenado!



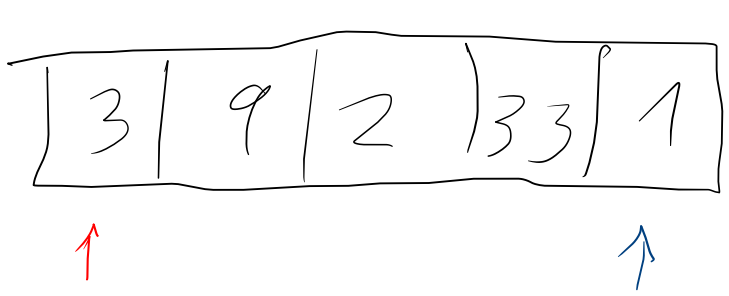



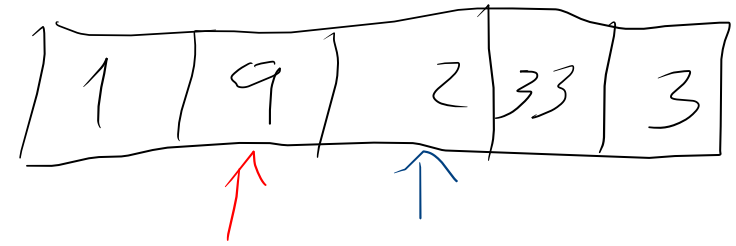



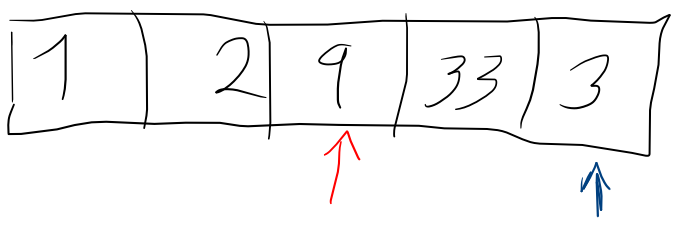



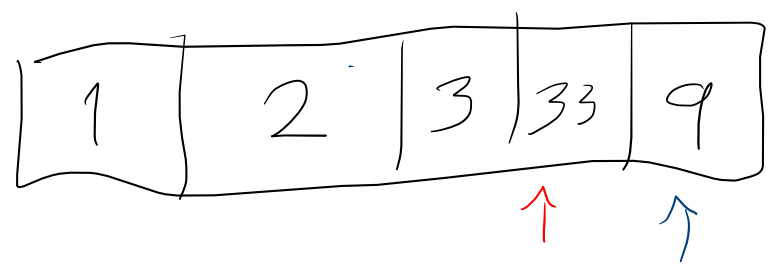
- Posso por todo array n vezes
- Em cada posição compara o atual com o próximo
- Se for maior troca

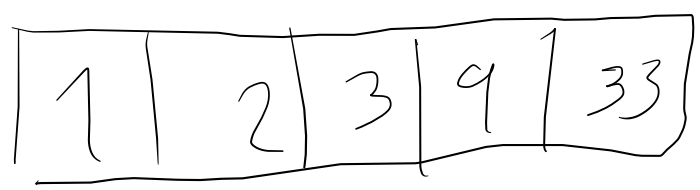
2 - Selection Sort



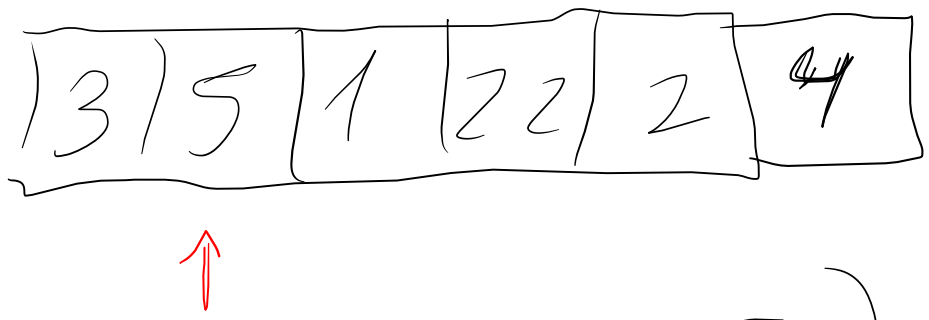



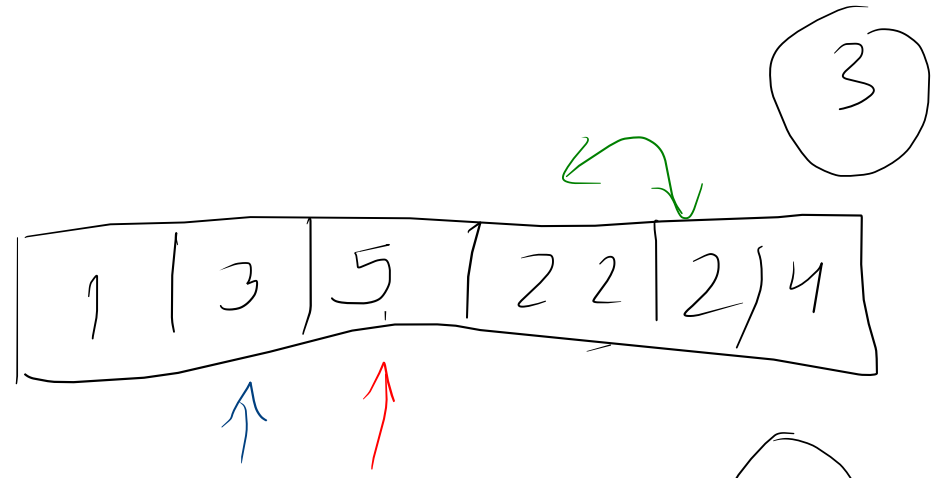



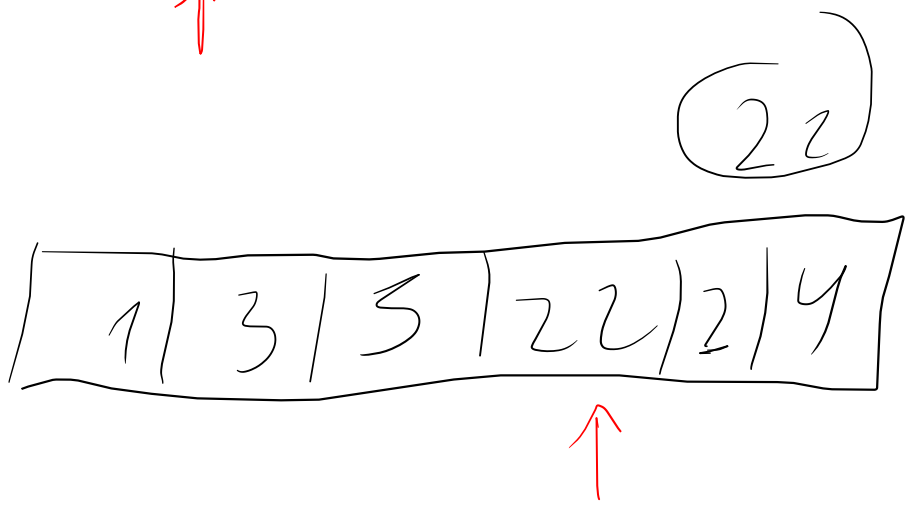



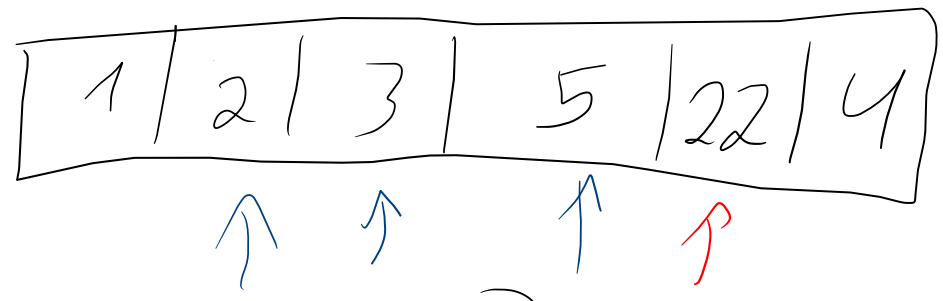
- Passa em cada posição do array
- Busco o menor cara depois dela e troco os dois (se tiver menor)

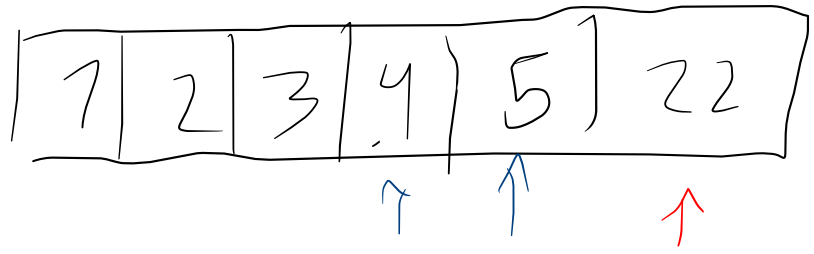
3. Insertion Sort



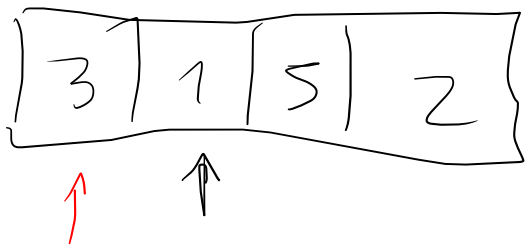



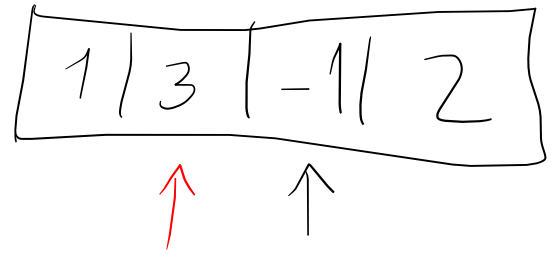
- Passo em cada posição do array
- Para cada posição acho a posição que ele deveria estar se o vetor acabasse nela.

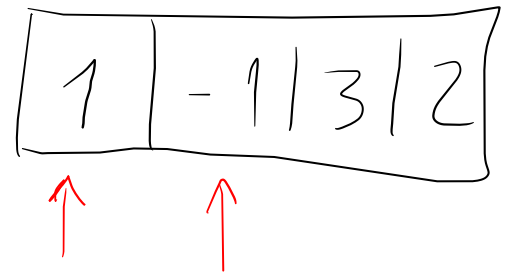

$a_1 \leq a_2 \leq \dots \leq a_n$

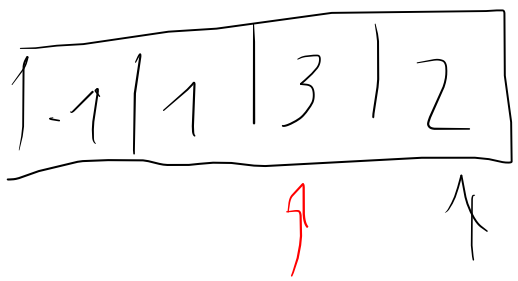
- Coloco ele lá e trago para frente todos entre o atual e a posição nova.

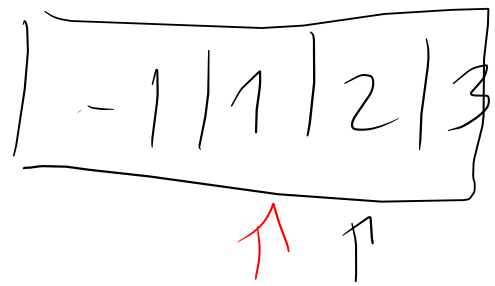
Versão 2



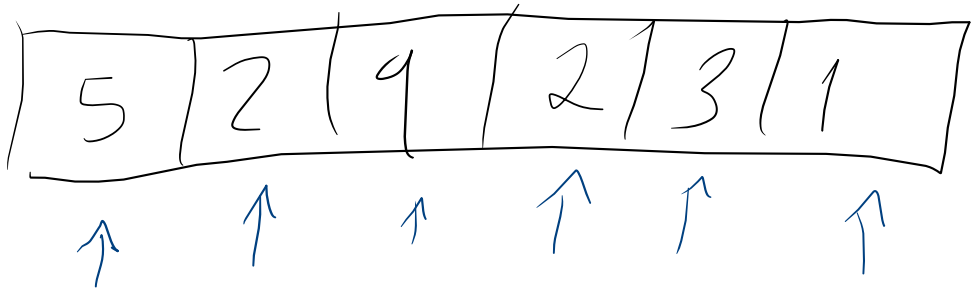



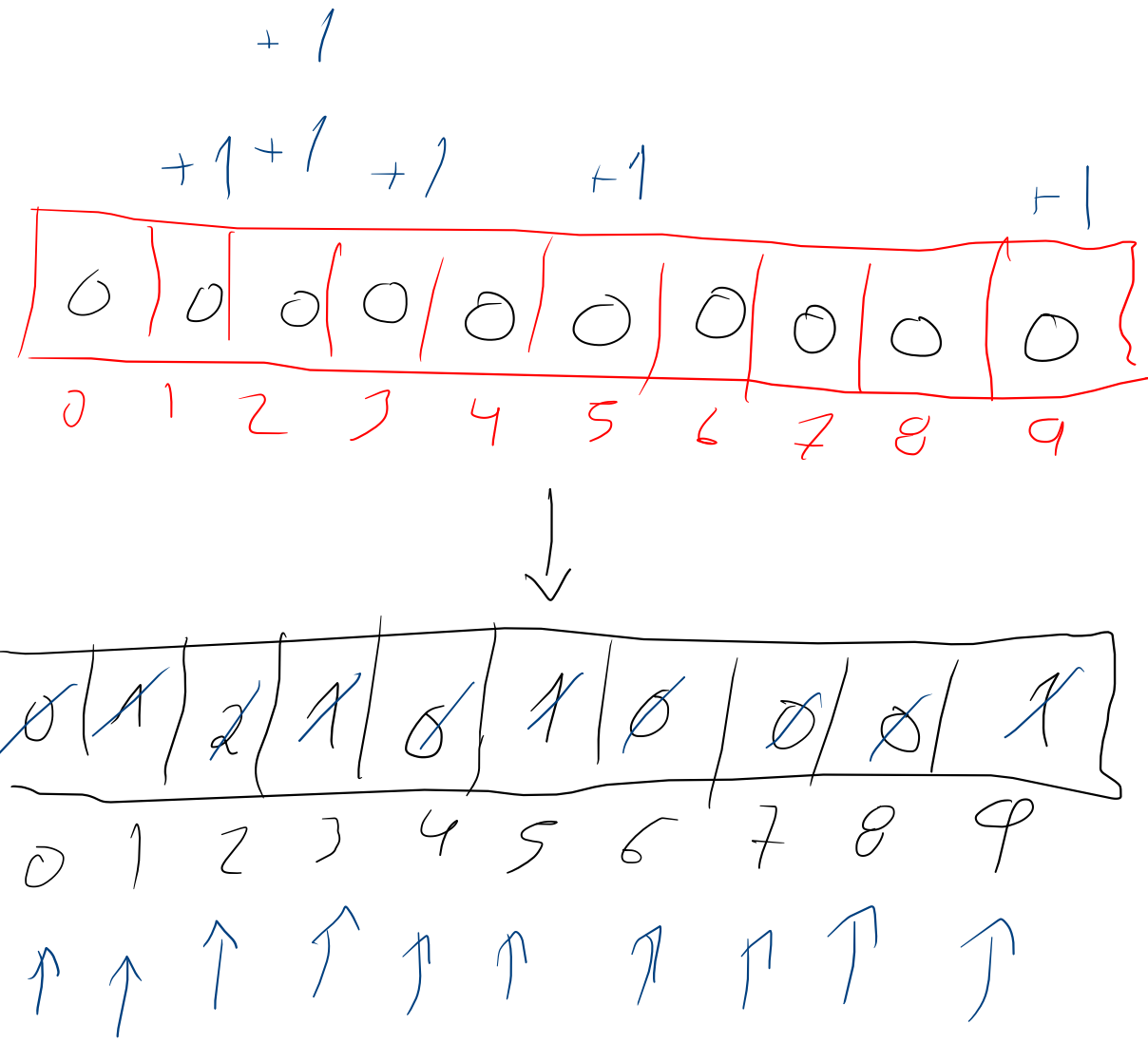

4. Counting Sort

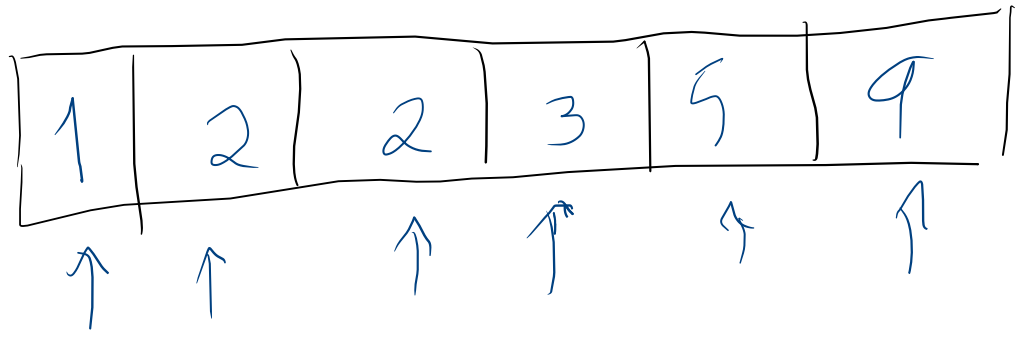
↳ Eficiente para inteiros pequenos

↳ Só funciona para inteiros e double em alguns casos.


5 2 9 2 3 1

max = 9


+1
+1 +1 +1 +1 +1
0 0 0 0 0 0 0 0 0 0
0 1 2 3 4 5 6 7 8 9
0 1 2 1 0 1 0 0 0 1
0 1 2 3 4 5 6 7 8 9


1 2 2 3 5 9

- Acho o máximo do vetor
- Crio um array de tamanho max+1
- Passo em cada elemento e incrementa na posição do número no array auxiliar
- Depois passo no array auxiliar e descarrego no array original.

Fluxo Trello

Trovou

A Fazer

Em andamento

Errado

Voltou

Pronto

Certo

Revisão

Concluído

Versão

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 1 | 3 | 1 | 2 |

in

max = 3

+1 +1 +1 +1 +1 +1 +1

| 0 | 0 | 0 | 0 |
|---|---|---|---|
| 0 | 1 | 2 | 3 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 3 | 2 | 1 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 4 | 2 | 1 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 4 | 6 | 1 |

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 4 | 6 | 7 |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 2 | 1 | 3 | 1 | 2 |

out

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 1 | 2 | 2 | 3 |

$S_3 = q_3 + q_2 + q_1 + q_0$

$S_n = q_n + q_{n-1} + \cdots + q_1 + q_0$

Para double: Tenho que saber o número de casas pós virgula.

Ex: Altura

$1.62 \xrightarrow{\times 100} 162$

$1.57 \rightarrow 157$

$1.86 \rightarrow 186$