

PO (Pesquisa e Ordenação)

↳ Java

↳ POO (programação orientada objeto)

↳ Pesquisa: Métodos de busca e círculos

não binária (string e números)

Algoritmos: Busca Binária, Trie, N-óres,

B+, B

↳ Ordenação: Bubble, Merge Sort, Radix

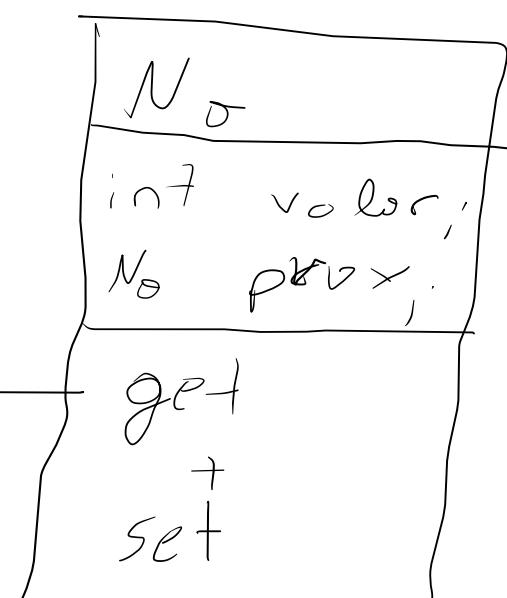
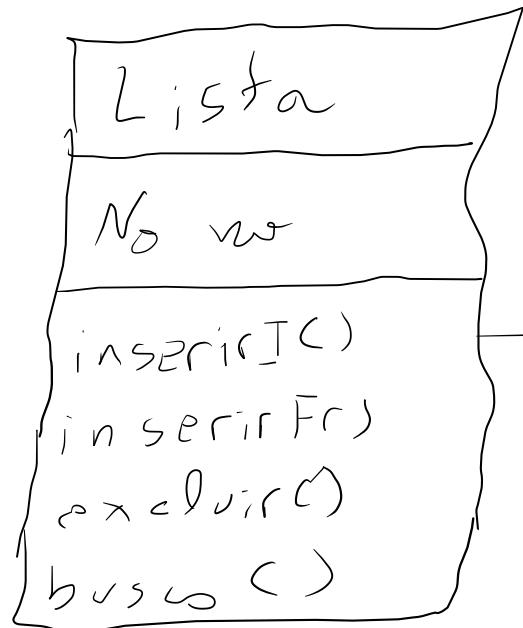
# Lista usando classes

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

Classe

++

Jovô:



Torefö:

- Listad ordensds
- Fils
- Pilks

# Buscos

## 1. Exaustivo / Linear

5	15	30	2	10	
↑	↑	↑	↑		
↑	↑	↑	↑	↑	↑

$$2 = 5$$

$$2 = 15$$

$$2 = 30$$

$$2 = 2 \quad \checkmark$$

$$19 = 5$$

$$19 = 15$$

$$19 = 30$$

$$19 = 2$$

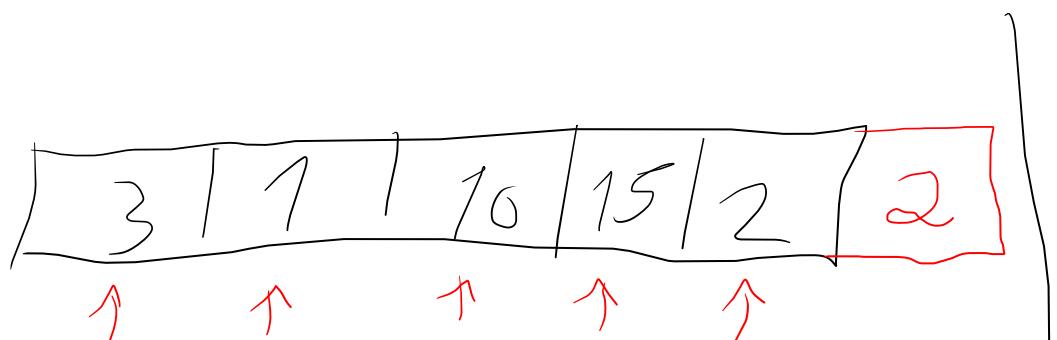
$$19 = 10$$

No o chou

## Busca Linear:

- Posso per correr pelas posições do array (comparando)
- Se chegou no final do array: Não achou
- Senso: Achou

## 2. Sentinels



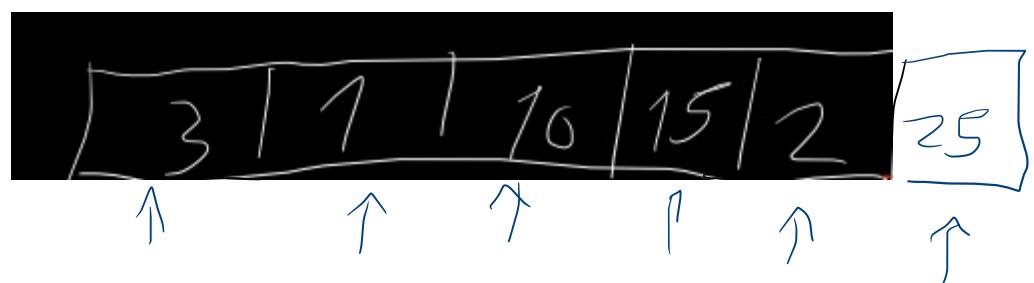
$$2 \neq 3$$

$$2 = 1$$

$$2 = 10$$

$$2 = 15$$

$$2 = 2 \checkmark$$



$$25 = 3$$

$$25 = 1$$

$$25 = 10$$

$$25 = 15$$

$$25 = 2$$

$$25 = 25 \text{ (False, não sentinela)}$$

## Busca Sentinelas

- Adicionar o sentinelas (valor no final)
- Possuir pelos posições do array (comprido)
- = Se não houver sentinelas; Não existe
- Senão, Achou

## 3. Indexada

Observações: Só vale para vetores ordenados

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

$$\text{arr}[0] \leq \text{arr}[1] \leq \text{arr}[2] \dots \leq \text{arr}[n]$$

$$20 \leq \text{arr}[2]$$

1	3	125	30	31	100
↑	↑	↑	↑	↑	

$$31 > 1$$

$$31 > 3$$

$$31 > 25$$

$$31 > 30$$

$$31 > 31 \text{ (Poro)} \Rightarrow 31 = 31 \quad \checkmark$$

1	3	125	30	31	100
↑	↑	↑	↑	↑	

$$17 > 1$$

$$17 > 3$$

$$17 > 25 \text{ (Poro)} \Rightarrow 17 = 25 \text{ (False)}$$

↑  
101

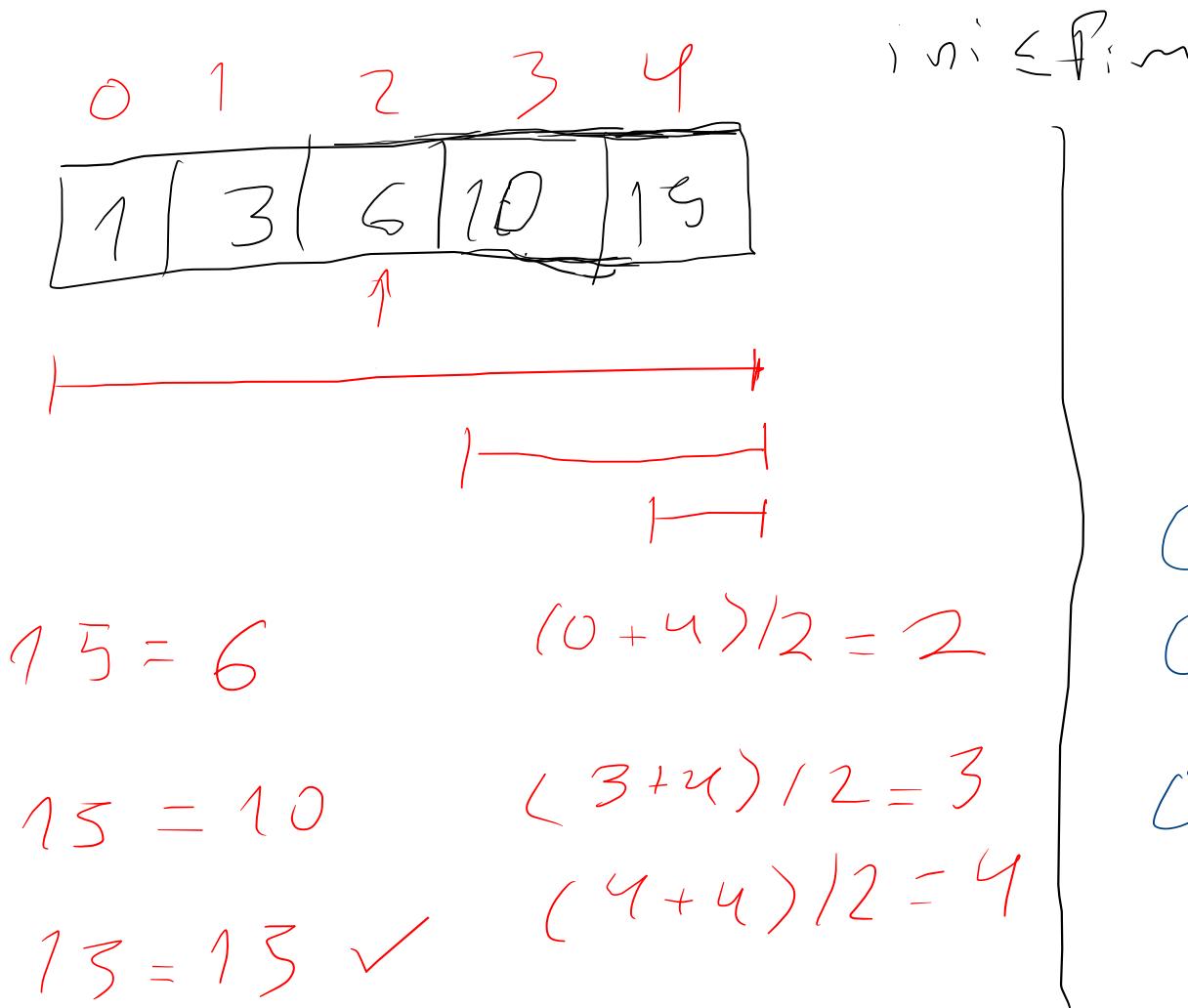
## Busca Indexada

- Posso per todos os posições quer ser [i] é menor que o busco
- Se por ou e for igual achar
- Senso: não achar

# 4. Buscando Binário

Observação: O array tem que estar ordenado.

Vantagem: Busca mais rápido  $O(\log n)$



$$\left. \begin{array}{l} 0 = 6 \\ 0 = 1 \\ 0 = 1 \end{array} \right\} \quad \begin{array}{l} (0+4)/2 = 2 \\ (0+1)/2 = 0 \\ (0+0)/2 = 0 \\ (0+ -1) \end{array}$$

0	1	2	3	4
1	3	6	10	15

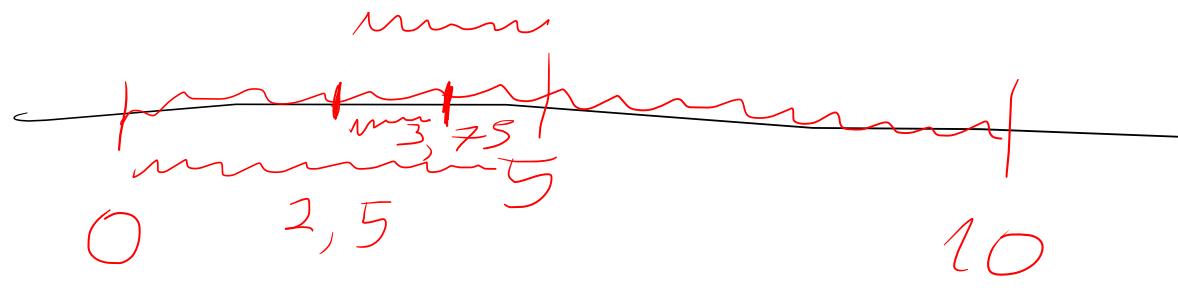
Busca Binaria:

- Pedro inicio e fim
- Verifica se inicio do intervalo:
  - Se for igual: ok
  - Se for maior: mudar inicio
  - Se menor: mudar o meio
- = Se inicio > fim, para: No : ok

Aplicaçõõ: Anchor  $r_{G,2}$  que é droado.

$$\sqrt{q} = ? \quad y \Rightarrow y^2 = q$$

$$5^2 > q \quad (\vee)$$



$$2,5^2 > q \quad (\text{F})$$

$$3,75^2 > q \quad (\vee)$$

$$3,125^2 > q \quad (\vee)$$

$$2,81^2 > q \quad (\text{F})$$

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

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

$$[2.81, 3.125]$$

Tarefa: Implementar os busses em lista.

# Ordensgröße

1. Bubble Sort +

5	10	1	3	50

↑      ↑

$$5 > 10 \quad (\text{F})$$

$$10 > 1 \quad (\text{V}) \rightarrow \text{True}$$

$$10 > 3 \quad (\text{V}) \rightarrow \text{True}$$

$$10 > 50 \quad (\text{F})$$

$$5 > 1 \quad (\text{V}) \rightarrow \text{True}$$

$$5 > 3 \quad (\text{V}) \rightarrow \text{True}$$

Ordenssch.

5	1	3	10	50

↑      ↑

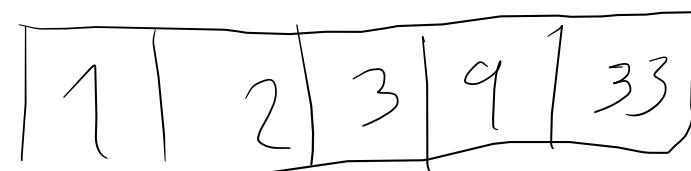
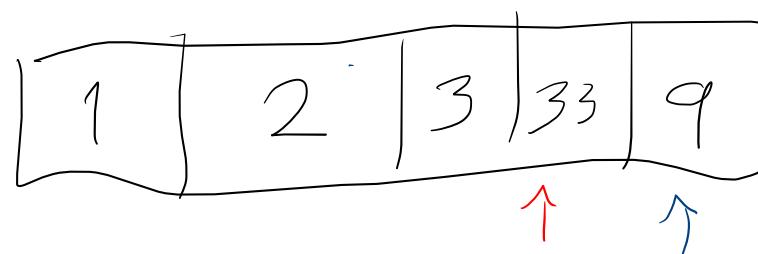
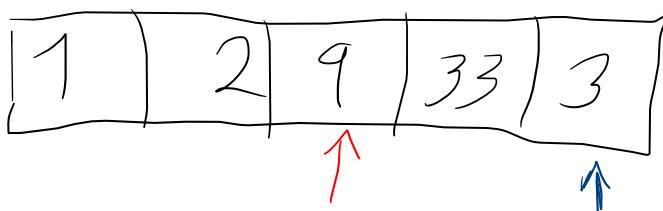
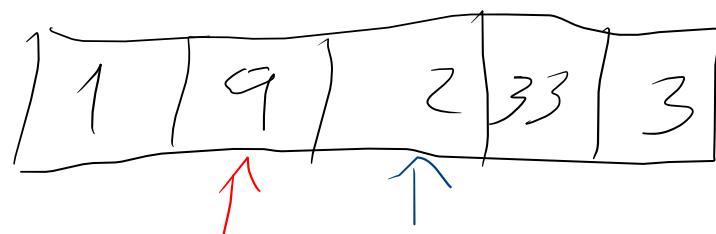
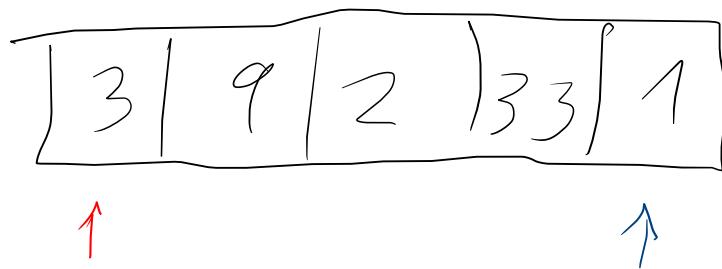
1	5	3	10	50

↑      ↑

1	3	5	10	50

- Posso per todo o array n vezes
- Em cada posição comparo o atual com o próximo
- Se for maior troco.

## 2 - Selection Sort

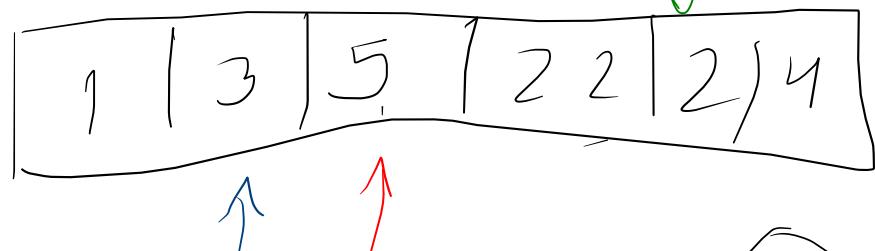


- Possa em cada posição do array
- Busco o menor valor depois dela e troço os dois (se tiver menor)

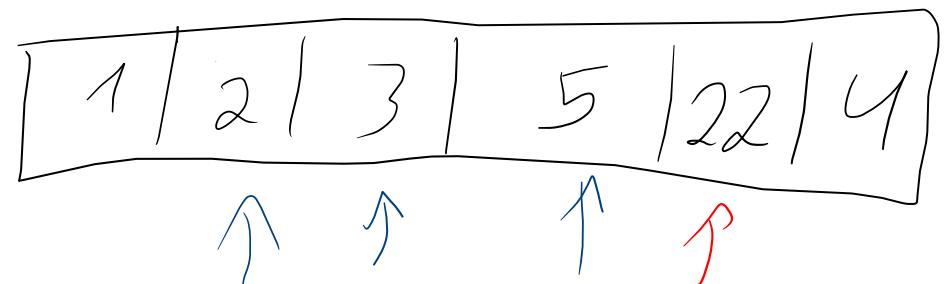
### 3. Insertion Sort



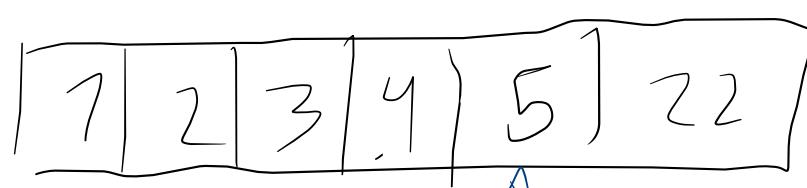
(2)



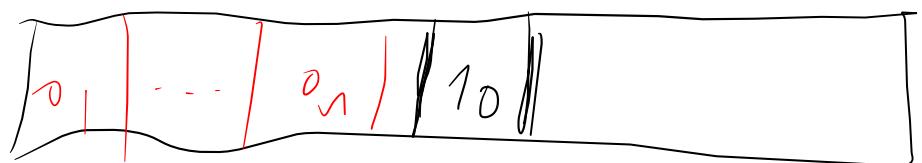
(2)



(4)



- Posso em cada posição do array
- Porém cada posição o chega a posição que ela deveria estar se o vetor fosse sóbresscrito.



$$o_1 \leq o_2 \leq \dots \leq o_n$$



- Vou chegar a troço para frente todos entre o final e a posição  $m$ .

## Versão 2

3	1	5	2
↑	↑		

1	3	-1	2
↑		↑	

1	-1	3	2
↑		↑	

-1	1	3	2
↑		↑	

-1	1	2	3
↑		↑	

## 4. Counting Sort

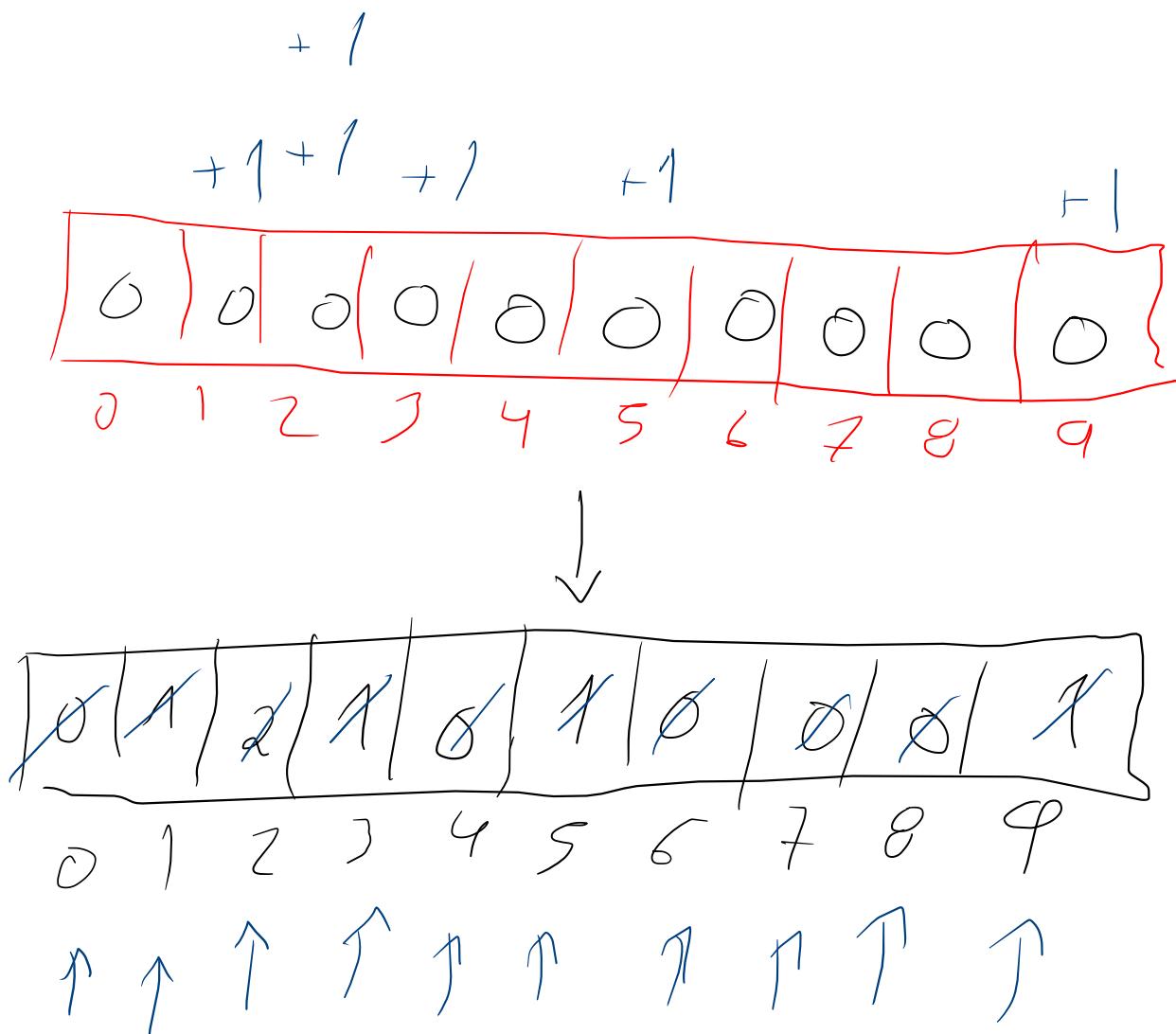
↳ Eficiente para inteiros pequenos

↳ Só funções para inteiros e double em alguns casos.

5	2	9	2	3	1
↑	↑	↑	↑	↑	↑

$$m \circ x = 9$$

1	2	2	3	5	9	
↑	↑	↑	↑	↑	↑	



- Acho o máximo do vetor
- Crio um orroy de tamanho  $m \times 1$
- Posso em cada elemento e incremento na posição de número no orroy auxiliar
- Depois posso no orroy auxiliar e descorregar no orroy original.



Versão

0	1	2	3	4	5	6
1	0	1	2	1	3	1
1	1	1	1	1	1	1

$$max = 3$$

1	1	3	1	2	1	1
0	1	2	3			

1	1	3	1	2	1	1
0	1	2	3			

0	1	1	1	2	2	3
0	1	2	3	4	5	6

+1	+1	+1	+1
0	1	2	3

+1	+1	+1	+1
0	1	2	3

1	0	2	1	1	3	1
0	1	2	3	4	5	6

$$S_3 = \varphi_3 + \varphi_2 + \varphi_1 + \varphi_0$$
$$S_n = \varphi_n + \varphi_{n-1} + \dots + \varphi_1 + \varphi_0$$

Para doble: Túlo que saber o  
número de cosos pós vírgula.

Ex: Altura

$$1.62 \xrightarrow{\times 100} 162 \leftarrow$$

$$1.57 \rightarrow 157$$

$$1.86 \rightarrow 186$$

$$\left. \begin{array}{r} 157 \\ 162 \\ 186 \end{array} \right\}$$

↓

$$\overline{1.57/1.62/1.86}$$

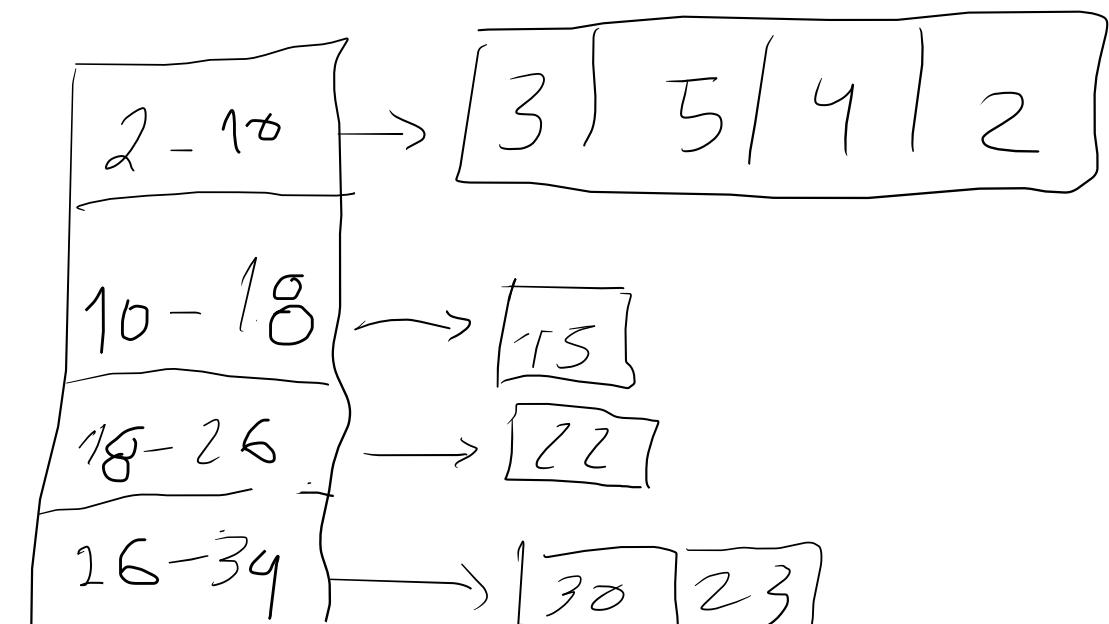
## 5. Bucket Sort

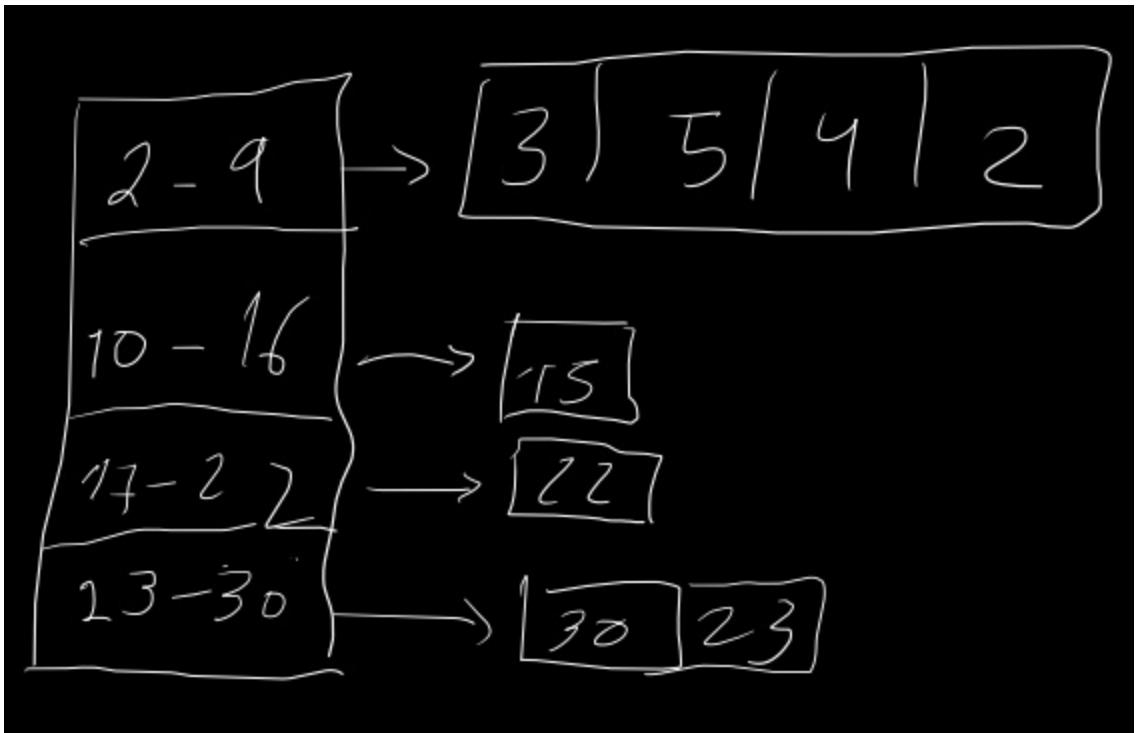
- ↳ Nós necessitamos de ordens
- ↳ Cria bodes e chama outros algoritmos para ordenar.

$$\text{Bolde} = 4$$

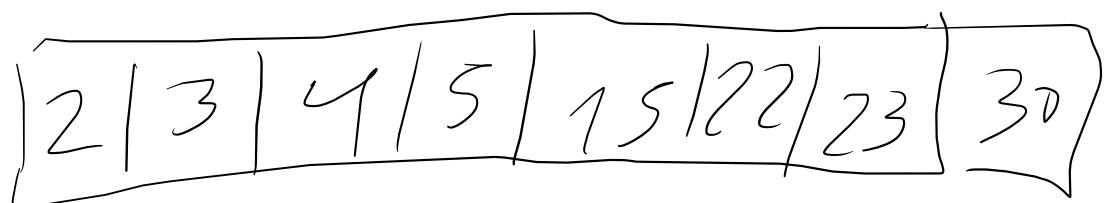
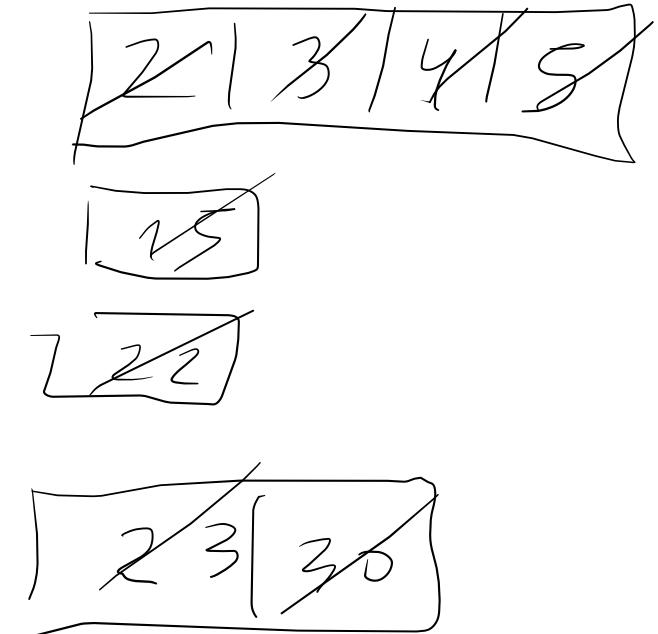


$$\min = 2 \quad \left\{ \begin{array}{l} 3_0 - 2 = \frac{28}{4} = 7 \\ \max = 30 \end{array} \right.$$





$\Rightarrow$



- Acha o min e max, calcula a diferença e divide pelo número de bloques
- Para cada intervalo  $[i, i+d]$ ,  $[i+d + 1, i+2d]$ ,

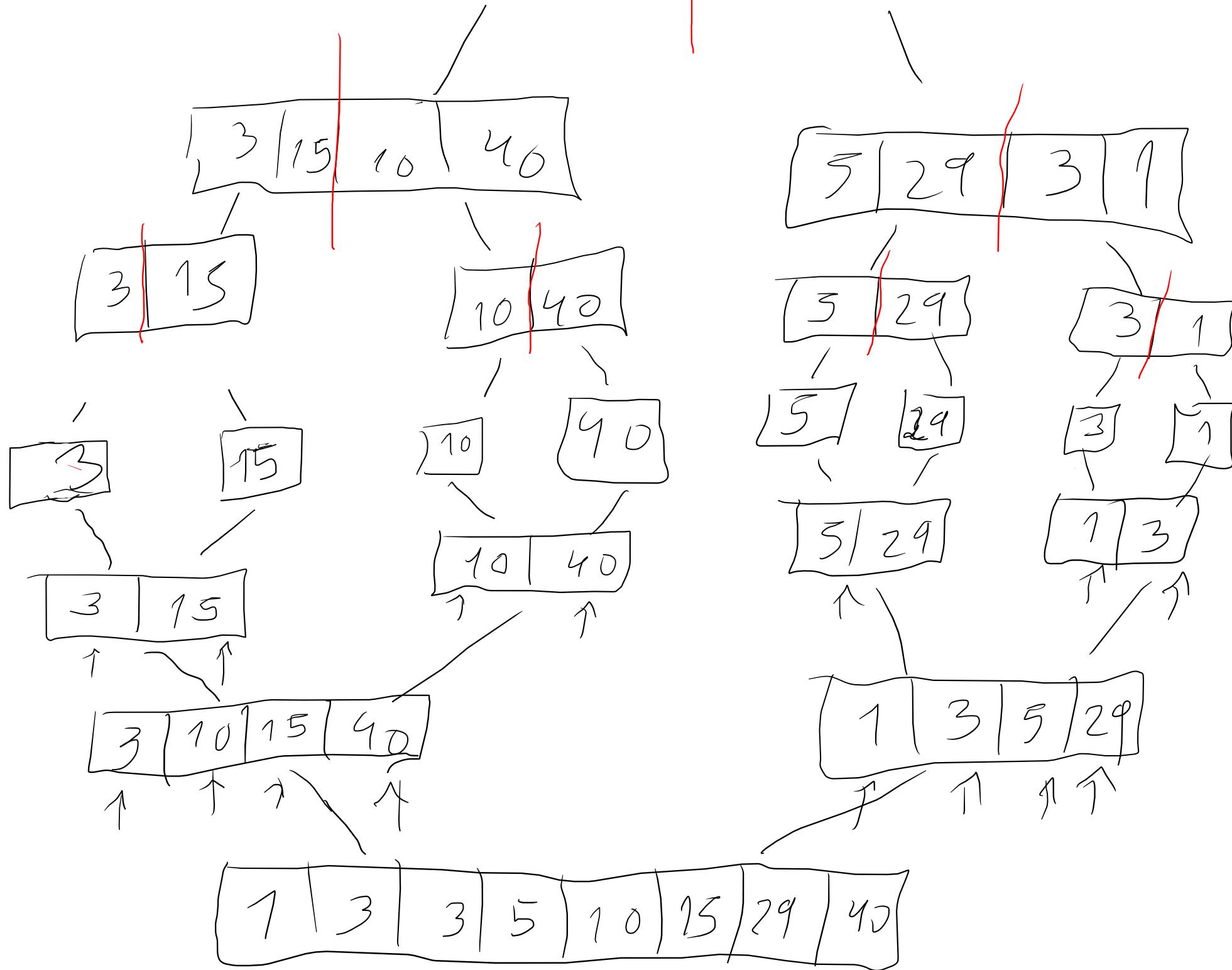
adições no respetivo bólde e ordena.  
• Orais posso em cada bólde em  
ordem e distribuir.

## 6. Merge Sort

Um dos algoritmos mais rápido,  
só que duplo memória.

Divisão e conquista

3	15	10	40	5	29	3	1
---	----	----	----	---	----	---	---



- Divide e array no meio e chama o algoritmo para os dois metodos, isto que o array temos tamanho 1.
- Para os dois arrays fuso o fusão dos dois metodos.

Fuso:

- Posso array do mesmo tamanho perguntando qual é menor
- Adiciono o menor e compro o proxímo
- Depois se sobrar em algum vetor, descobreço