Métodos de classificação

Jacqueline Midlej do Espírito Santo

Roteiro

Introdução

Métodos iterativos
- Ordenação por trocas
- Método das bolhas
- Ordenação por inserção

Métodos recursivos - Divisão e Conquista
- Quick Sort
- Merge Sort

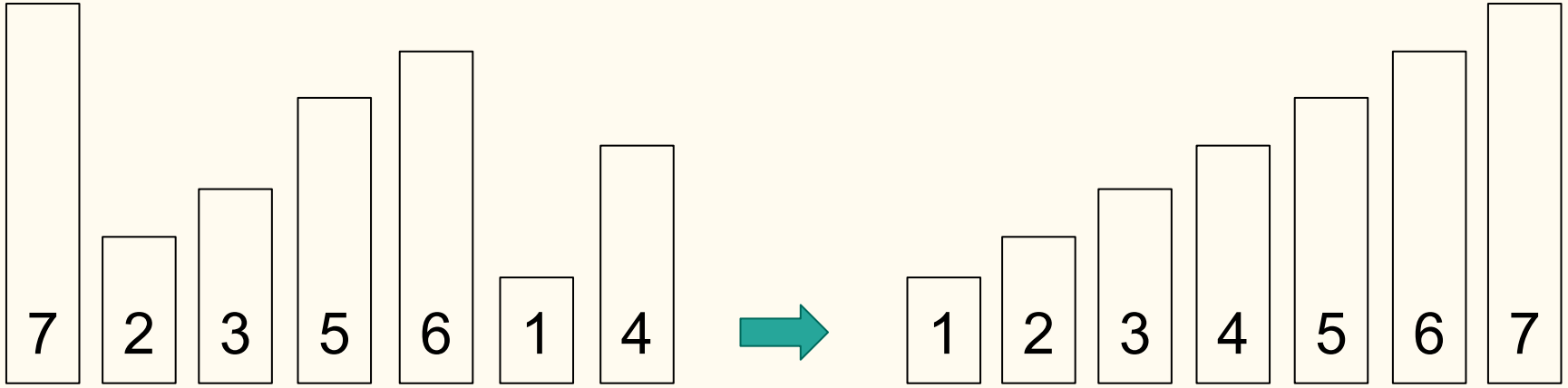
Complexidade algorítmica dos algoritmos de classificação

Pré-requisito

- Vetores e ponteiros
- Recursão
- Noção de notação assintótica de crescimento

Introdução

- **Problema:** Ordenar uma lista de número inteiros



Introdução

- **Problema mais amplo:** ordenar uma lista de objetos (dados estruturados) de acordo com um critério de ordenação pré estabelecido
  - Pessoas. Critério: por nome, ou cpf, ou data de nascimento)
  - Documentos. Critérios: por data de criação, de atualização, nome...
  - ...

Roteiro

Introdução

Métodos iterativos

- **Ordenação por trocas**
- Método das bolhas
- Ordenação por inserção

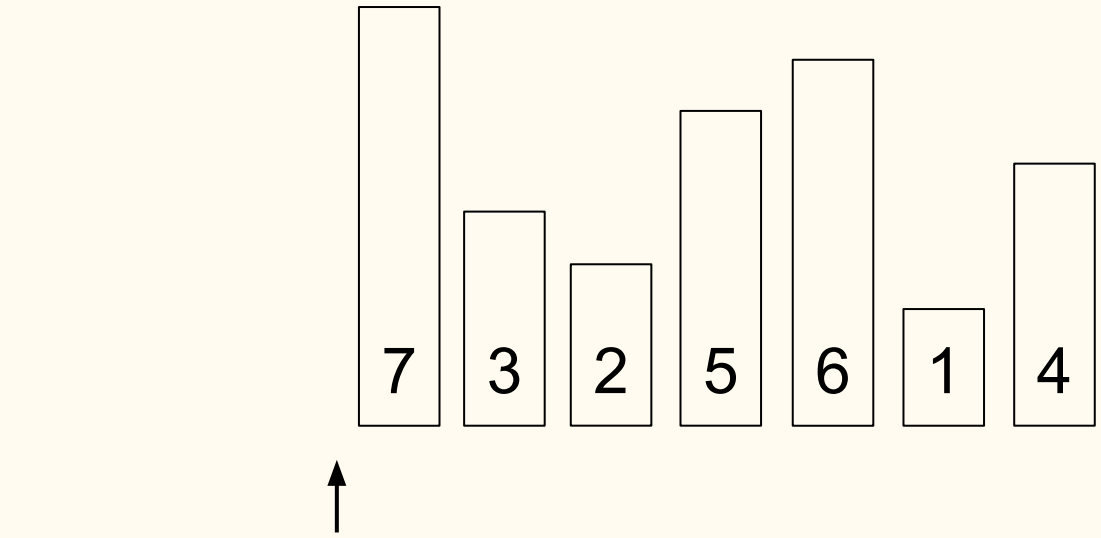
Métodos recursivos - Divisão e Conquista

- Merge Sort
- Quick Sort

Complexidade algorítmica dos algoritmos de classificação

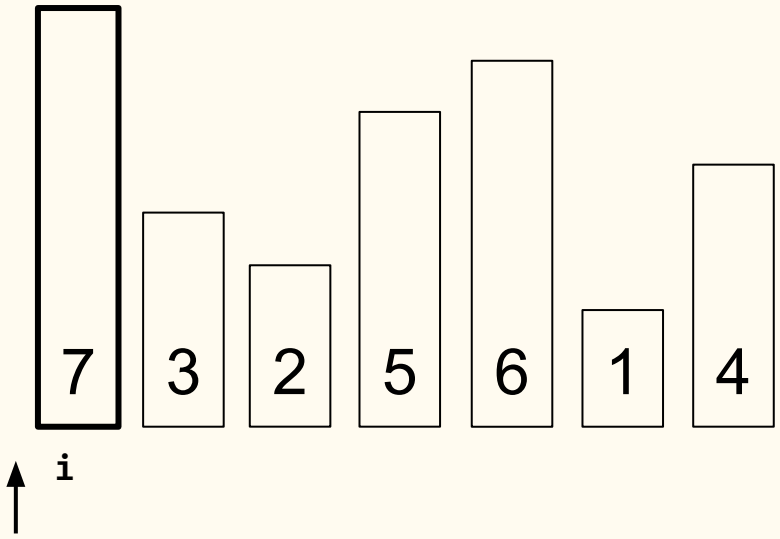
Ordenação por trocas

- Compara o elemento com todos os demais ainda não ordenados
- Posiciona o menor elemento corretamente, posteriormente o segundo menor...



Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

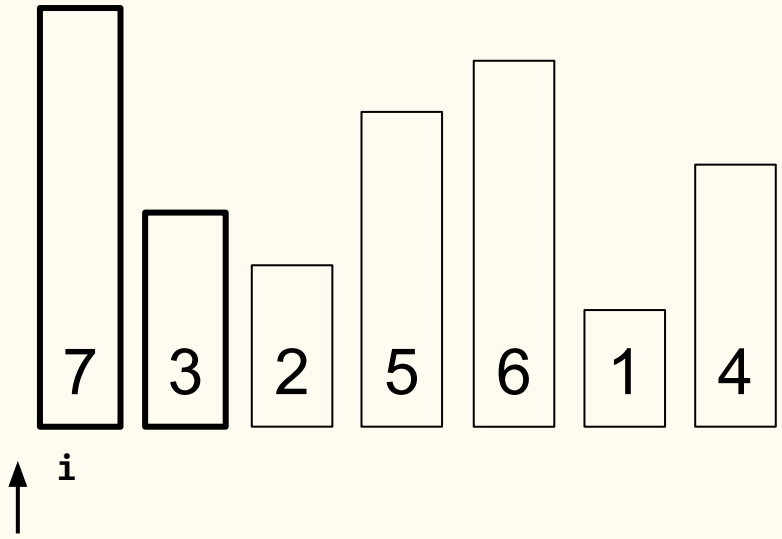


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

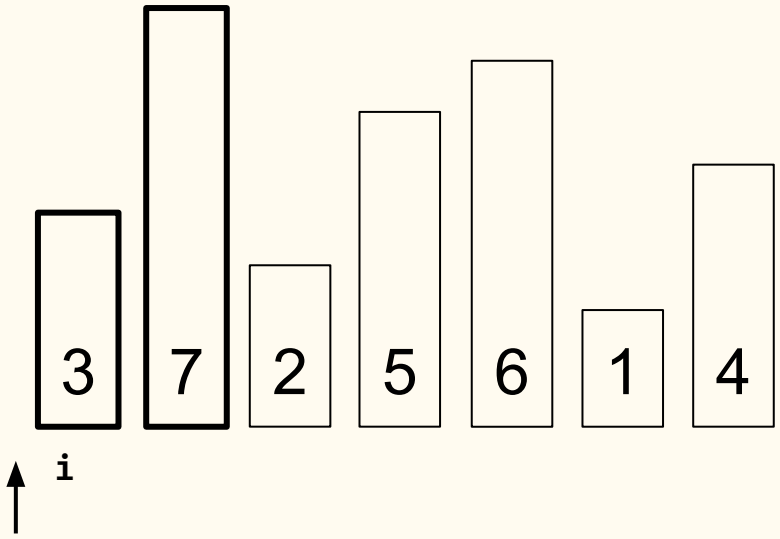


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

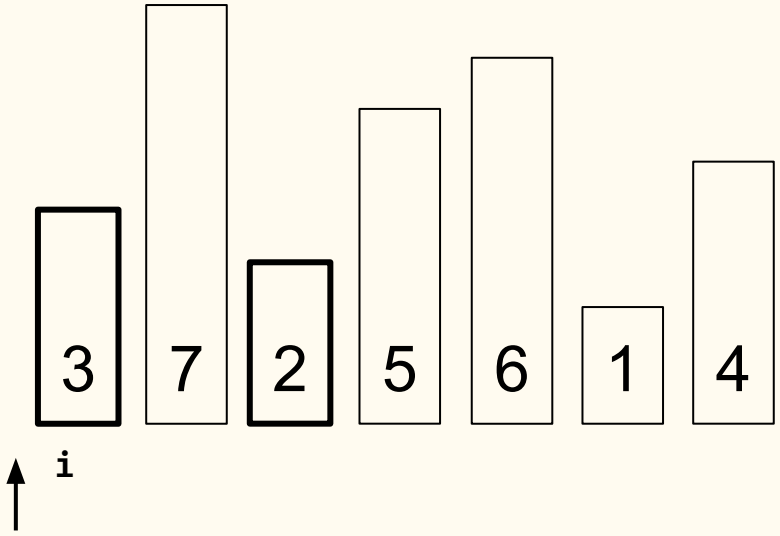


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

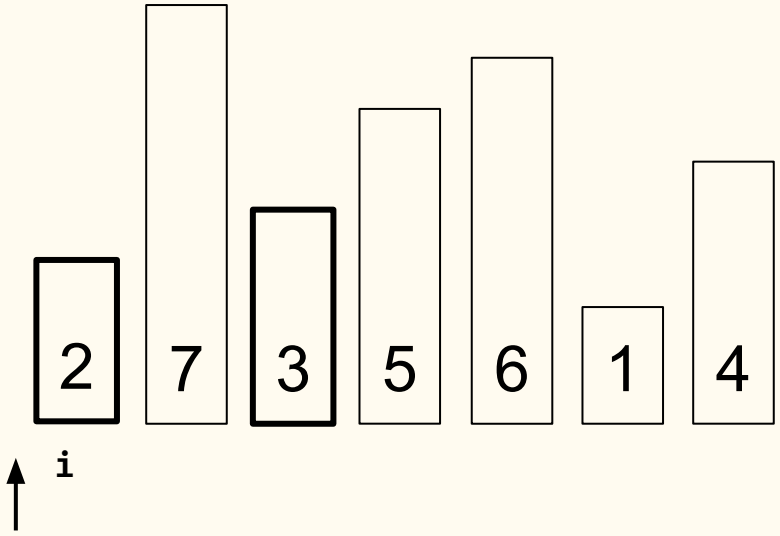


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

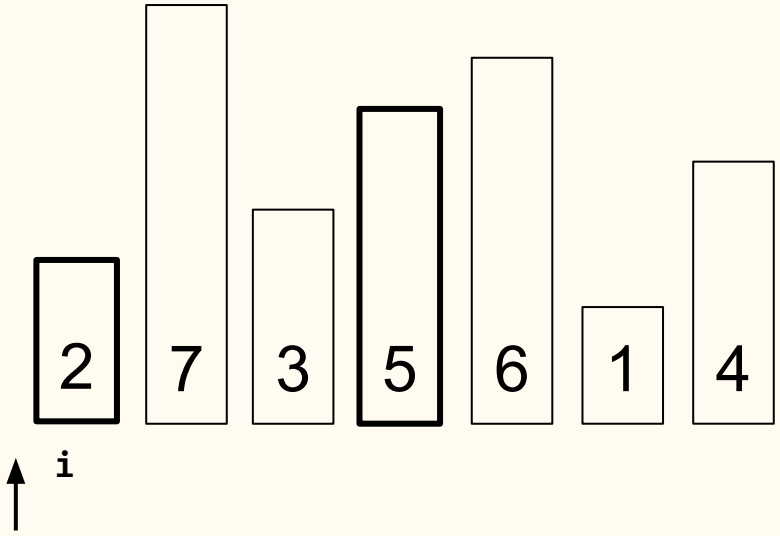


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

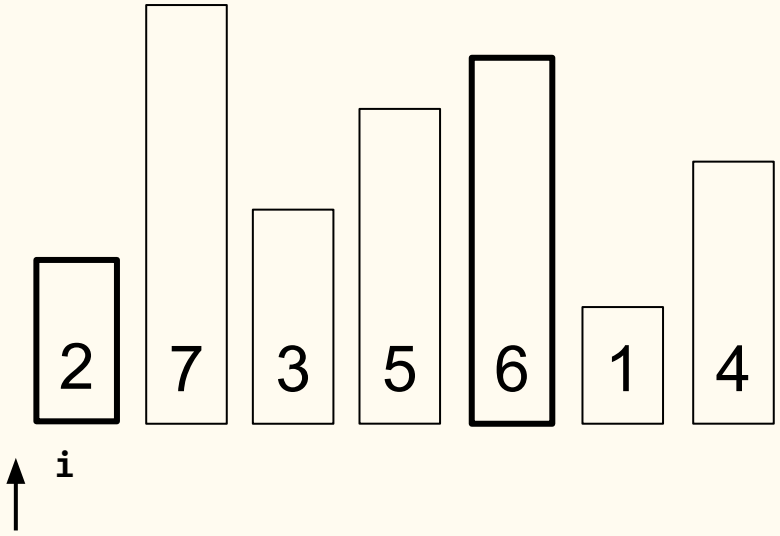


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

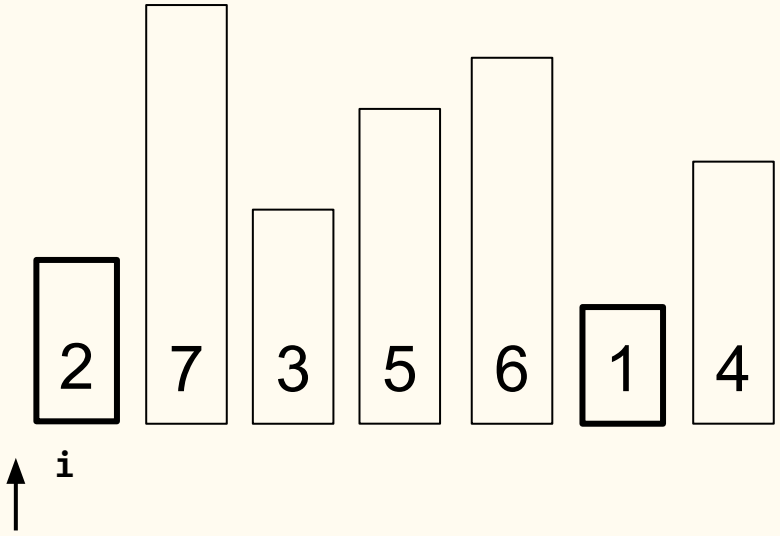


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

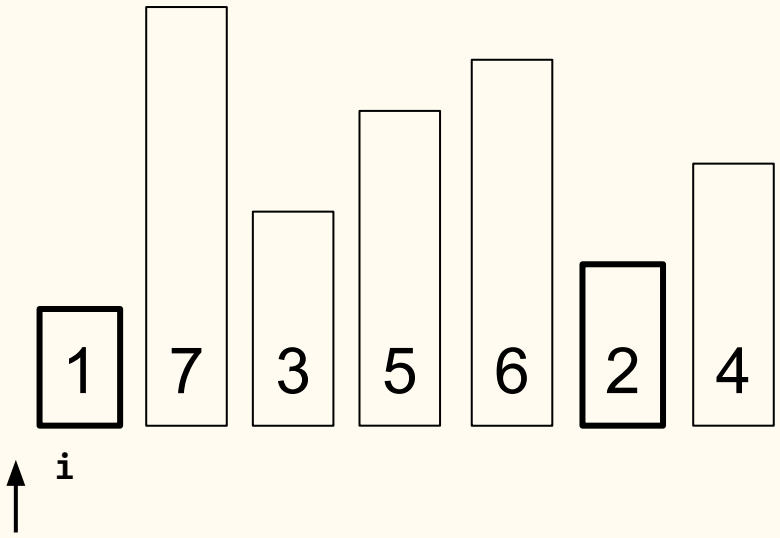


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

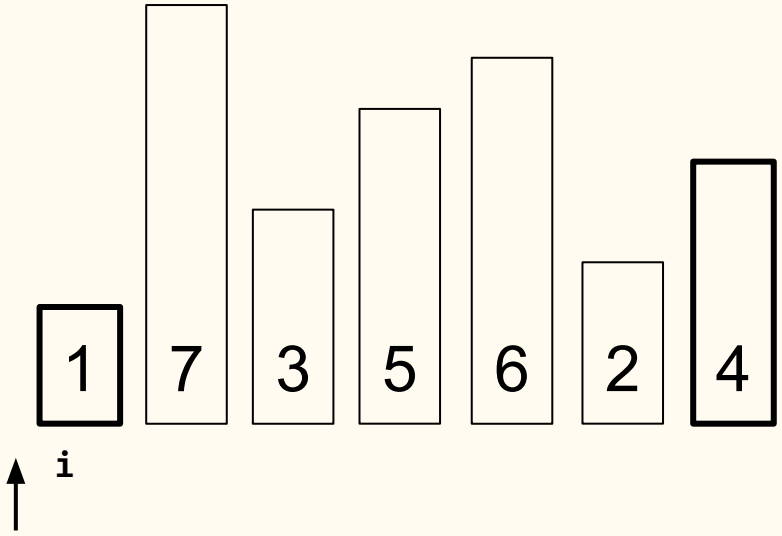


Parte ordenada

Parte desordenada

Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

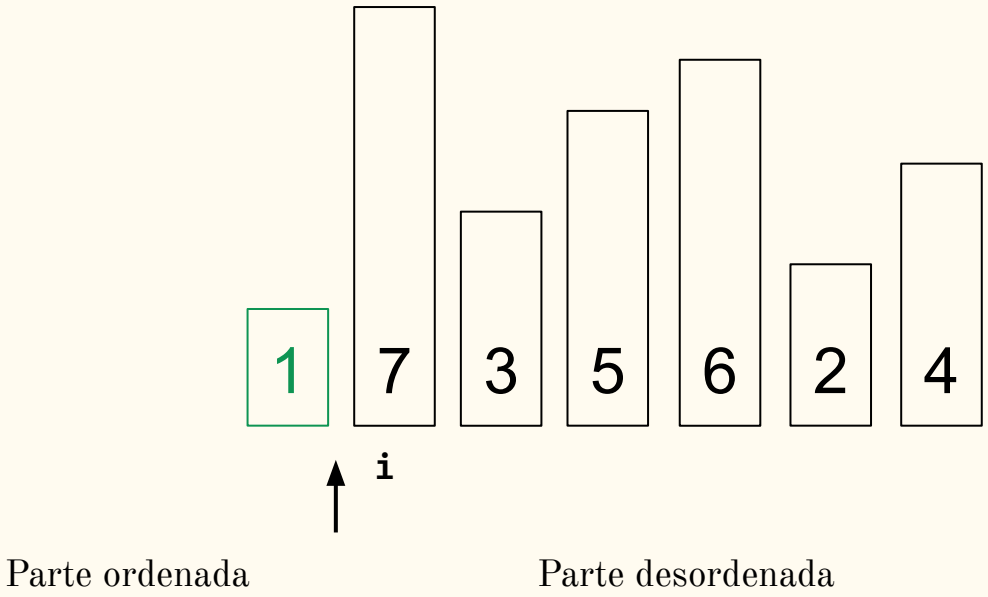


Parte ordenada

Parte desordenada

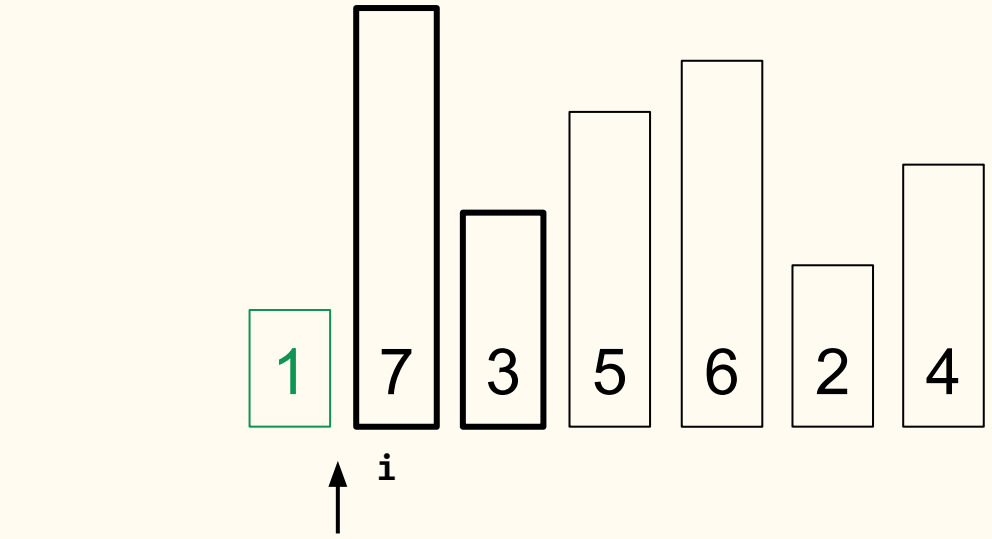
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



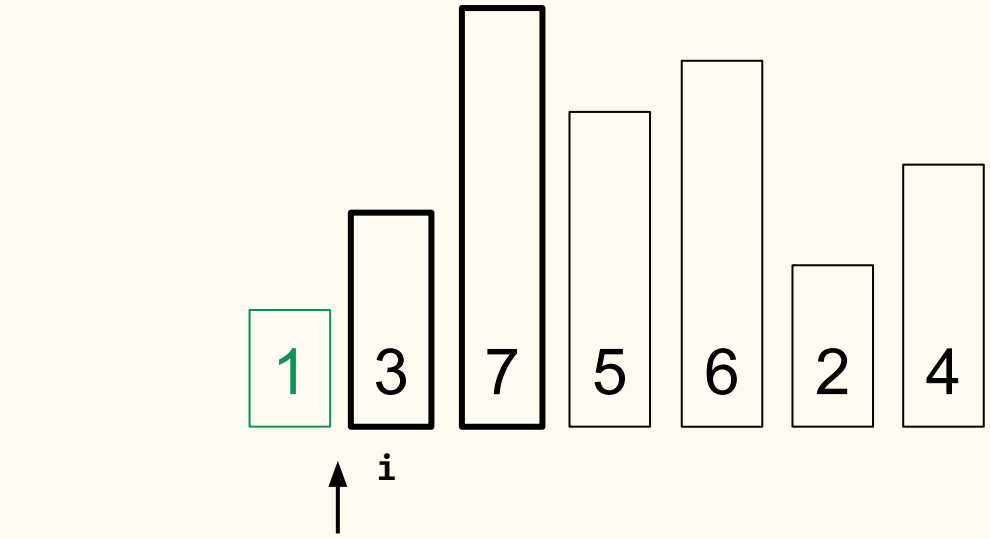
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



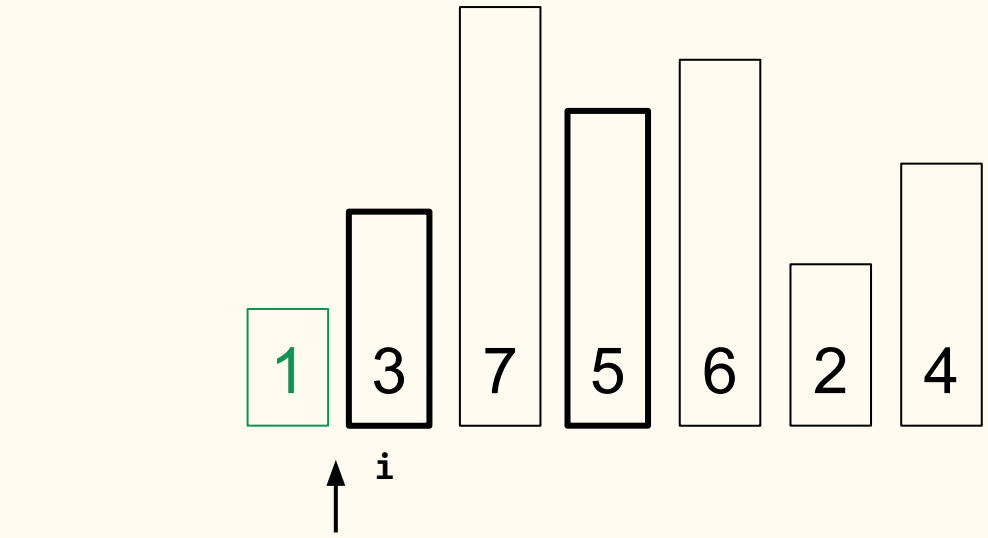
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



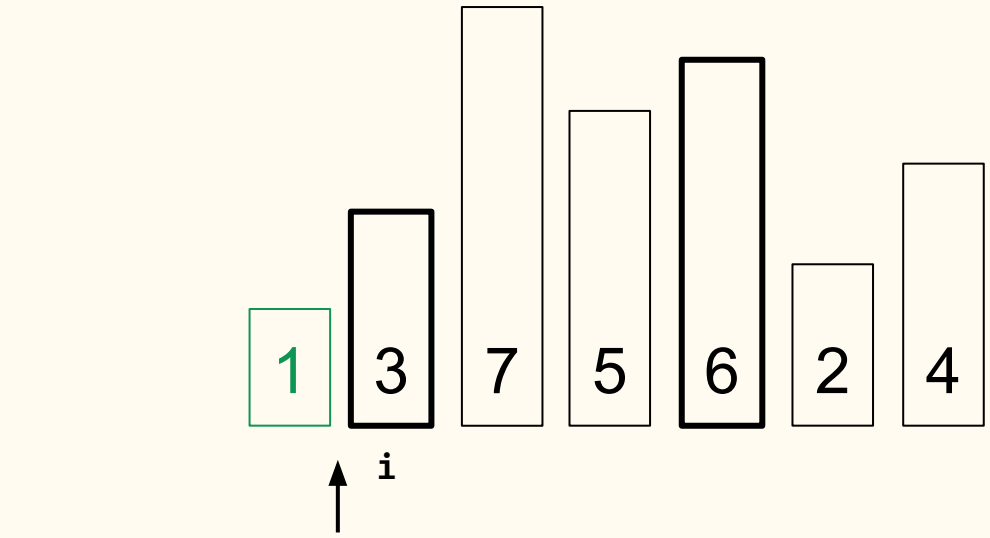
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



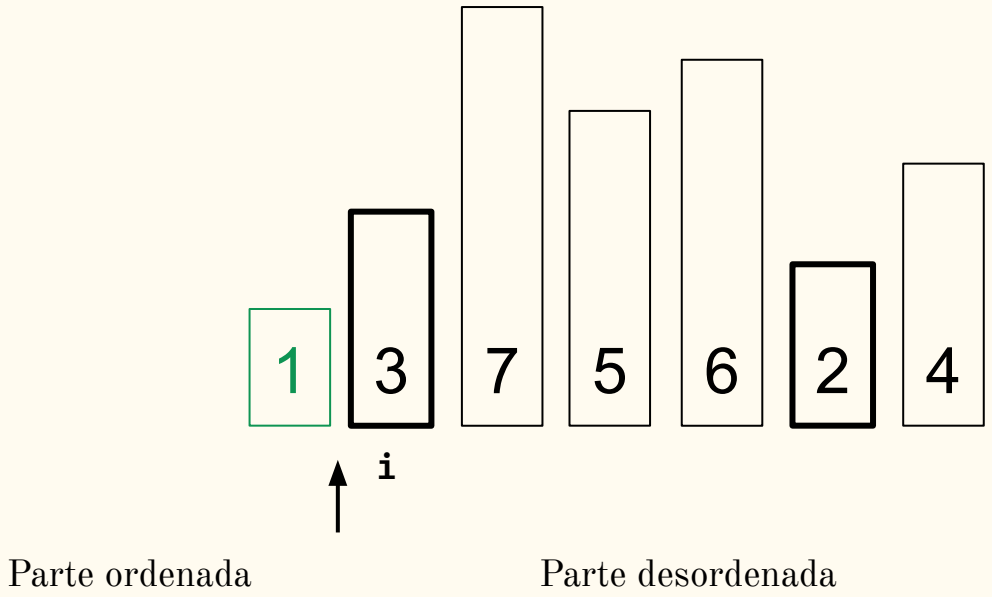
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



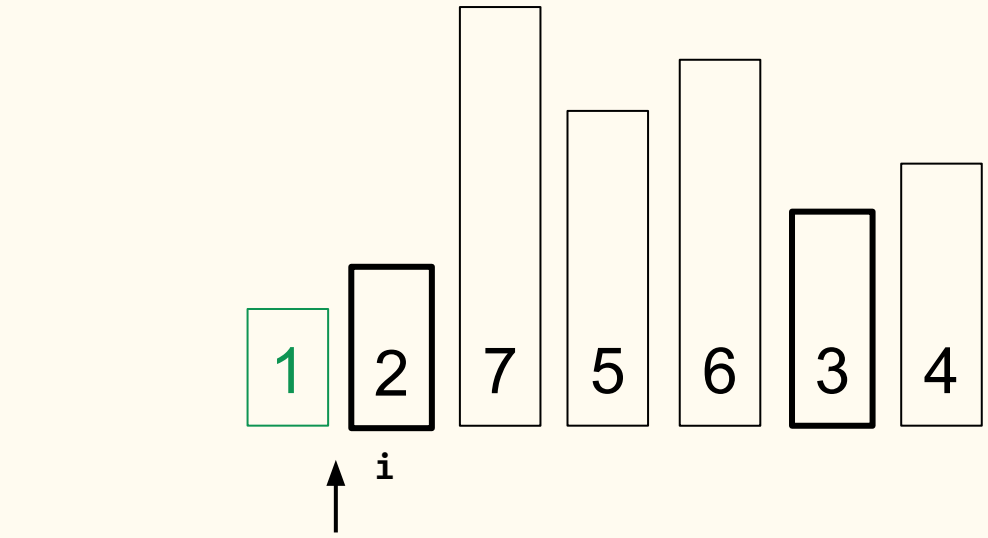
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



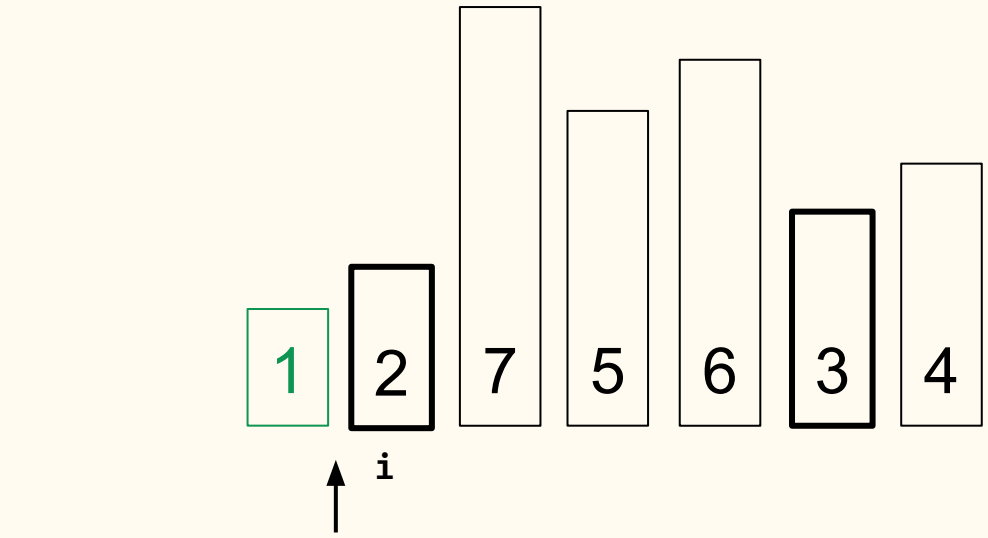
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



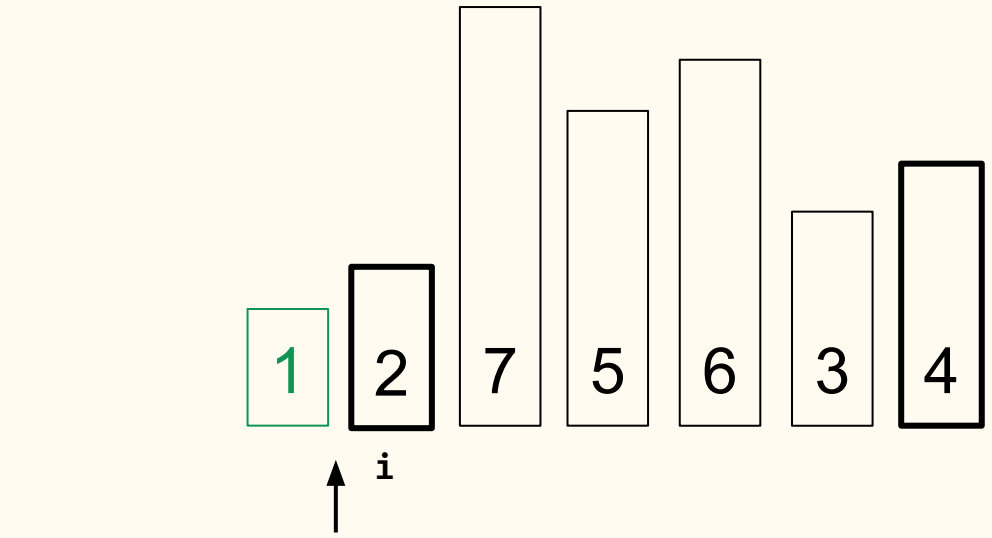
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



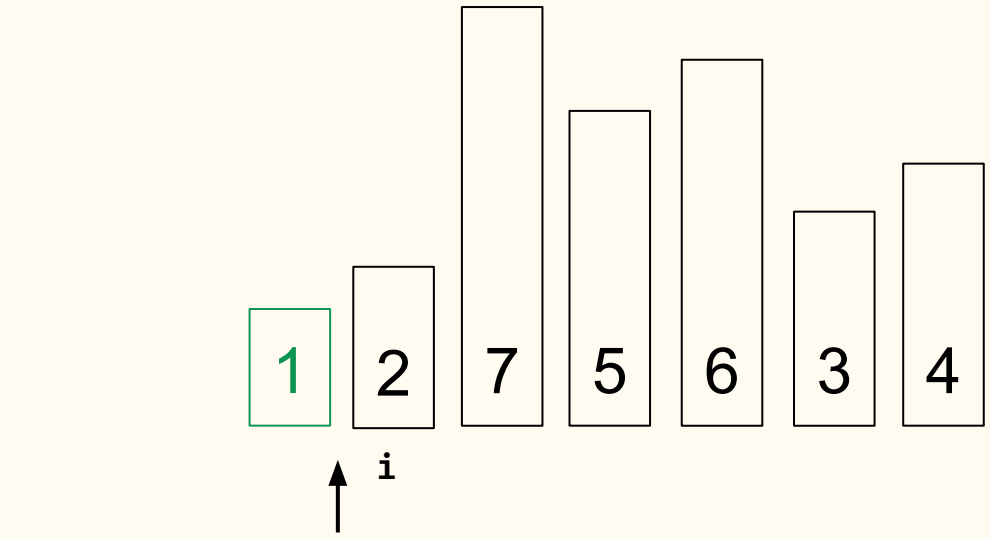
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



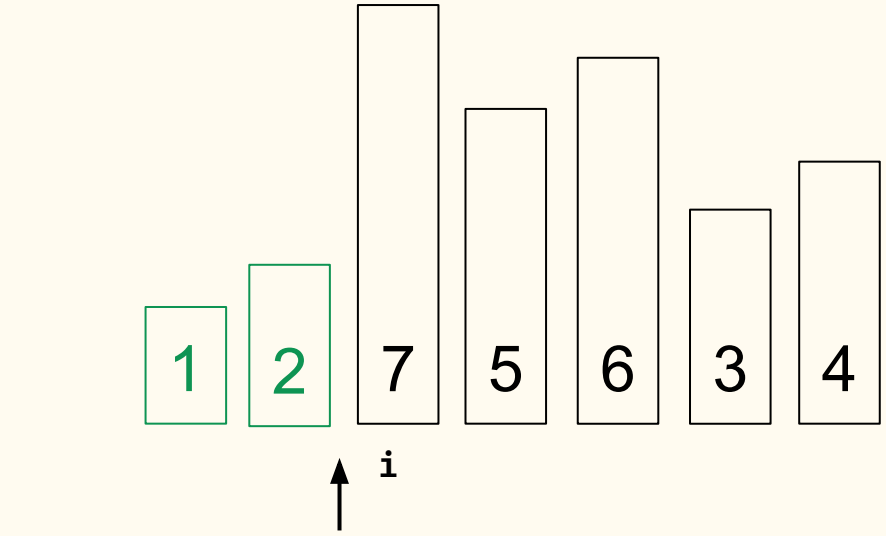
Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i



Ordenação por trocas - passo a passo

- A cada iteração, única regra: passa pelos elementos da lista desordenada, troca se o elemento é menor que i

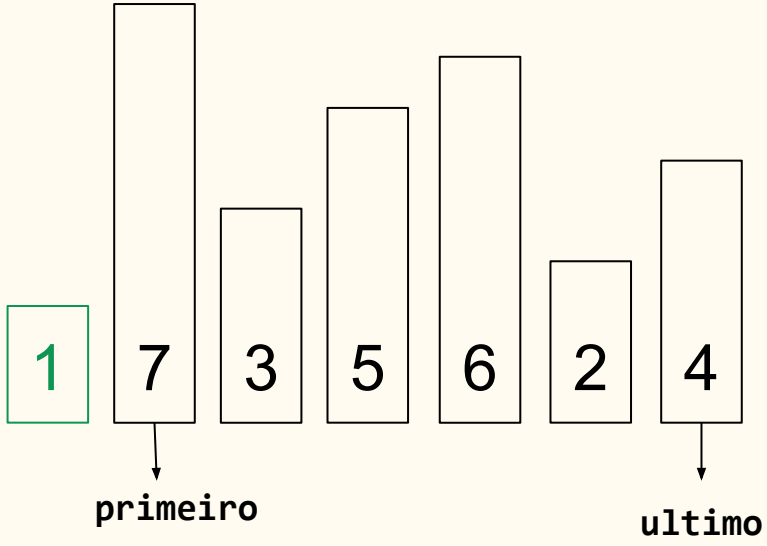


Ordenação por trocas - Codificando...

```
Void troca(int *a, int *b) {
    int aux = *a;
    *a = *b;
    *b = aux;
}
```

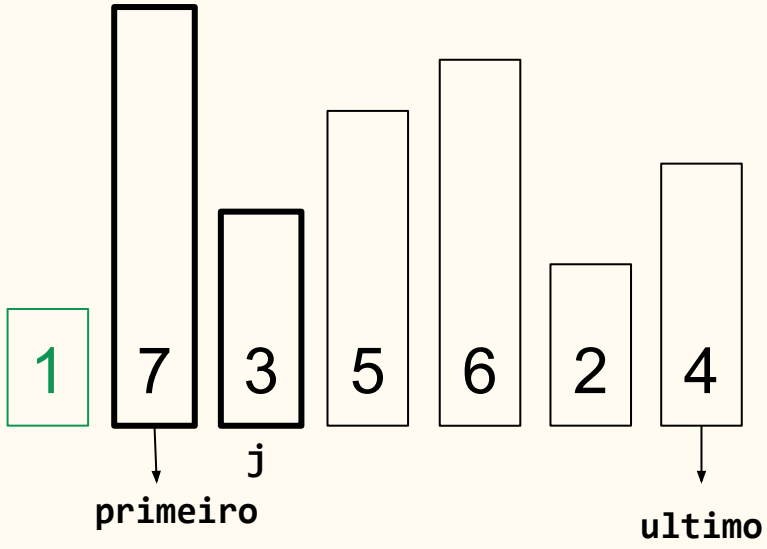
Ordenação por trocas - Codificando...

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(&vetor[j], &vetor[primeiro]);
}
```



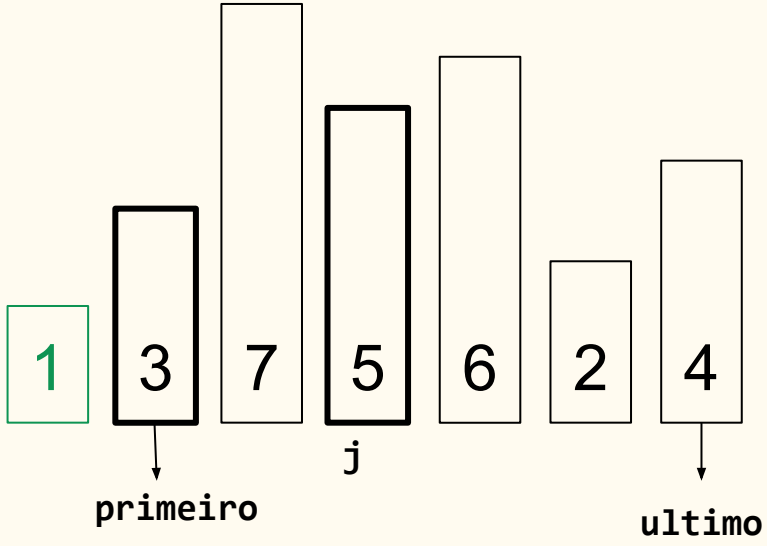
Ordenação por trocas - Codificando...

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(&vetor[j], &vetor[primeiro]);
}
```



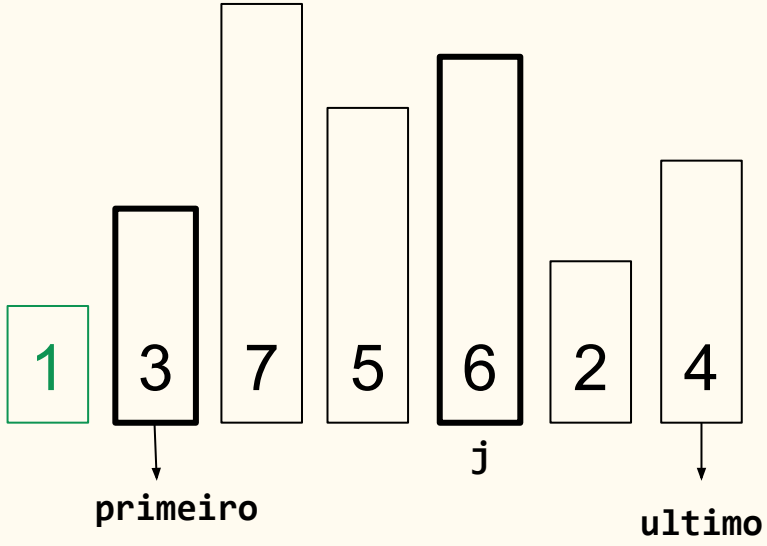
Ordenação por trocas - Codificando...

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(&vetor[j], &vetor[primeiro]);
}
```



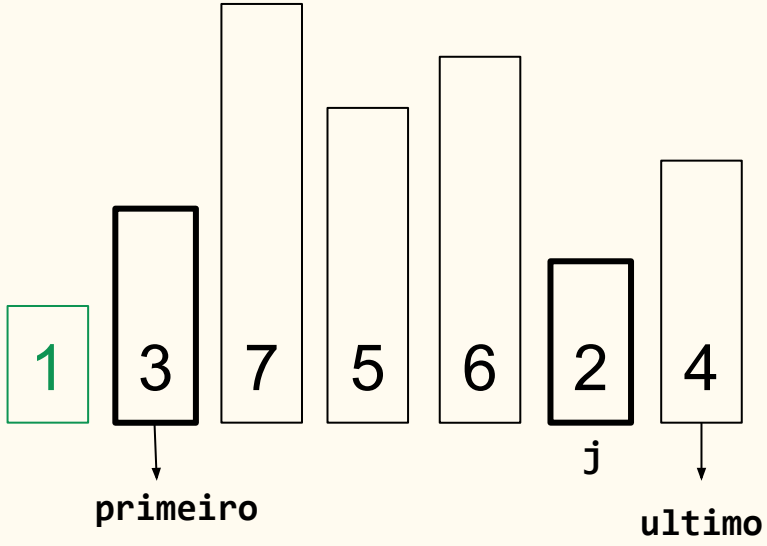
Ordenação por trocas - Codificando...

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(&vetor[j], &vetor[primeiro]);
}
```



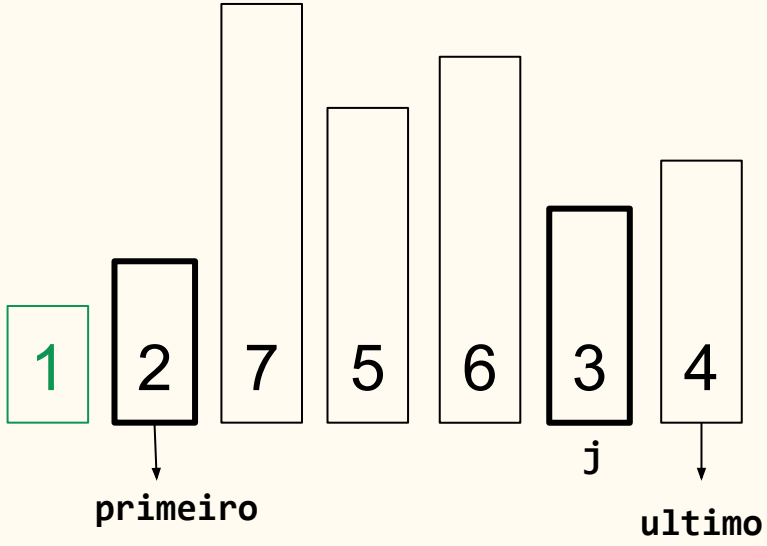
Ordenação por trocas - Codificando...

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(&vetor[j], &vetor[primeiro]);
}
```



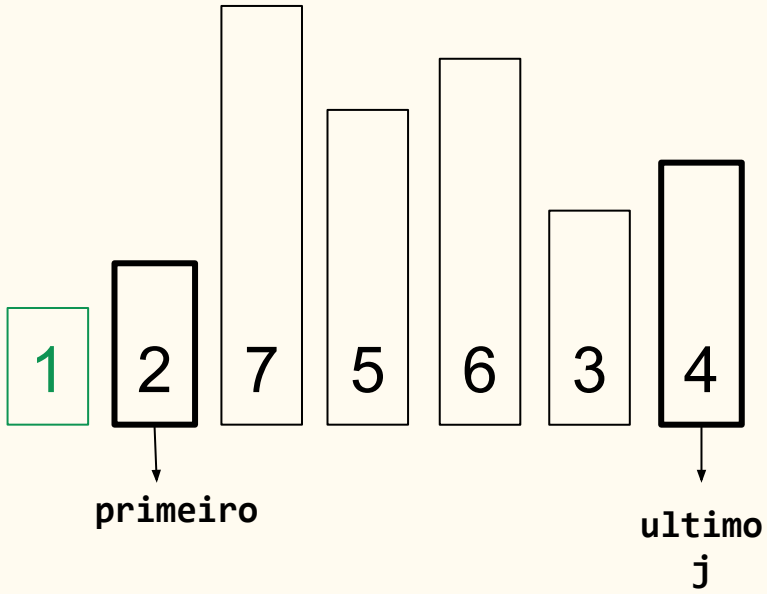
Ordenação por trocas - Codificando...

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(&vetor[j], &vetor[primeiro]);
}
```



Ordenação por trocas - Codificando...

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(&vetor[j], &vetor[primeiro]);
}
```



Ordenação por trocas - Codificando...

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(vetor[j], vetor[primeiro]);
}

void Ordenacao_troca (int vetor[], int n) {
     int i ;
     for (i = 0 ; i < n-1 ; i++)
          posiciona_menor(vetor, i, n);
}
```

Ordenação por trocas - Complexidade

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(vetor[j], vetor[primeiro]);
}
```

nº comparações:
nº de trocas:

Ordenação por trocas - Complexidade

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(vetor[j], vetor[primeiro]);
}
```

ultimo-primeiro-1

1

nº comparações: ultimo-primeiro-1
nº de trocas:

Ordenação por trocas - Complexidade

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(vetor[j], vetor[primeiro]);
}

void ordenacao_troca (int vetor[], int n) {
     int i ;
     for (i = 0 ; i < n-1 ; i++)
          posiciona_menor(vetor, i, n);
}
```

nº comparações: ultimo-primeiro-1
nº de trocas: depende do vetor
máximo: ultimo-primeiro-1
mínimo: 0

Ordenação por trocas - Complexidade

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(vetor[j], vetor[primeiro]);
}
```

nº comparações: ultimo-primeiro-1
nº de trocas: depende do vetor
máximo: ultimo-primeiro-1
mínimo: 0

```
void ordenacao_troca (int vetor[], int n) {
     int i ;
     for (i = 0 ; i < n-1 ; i++)
          posiciona_menor(vetor, i, n);
}
```

i=0:
posiciona_menor(vetor, 0,n) = n-0-1 = n-1
i=1:
posiciona_menor(vetor, 1, n)=n-1-1=n-2
i=2:
posiciona_menor(vetor, 2, n)=n-2-1=n-3
....
i=n-2 (última)
posiciona_menor(vetor,n-2,n)=n-(n-2)-1=1

Ordenação por trocas - Complexidade

```
void posiciona_menor (int v[], int primeiro, int ultimo){
     int j;
     for (j=primeiro+1; j<ultimo; j++)
          if (vetor[j]<vetor[primeiro])
               troca(vetor[j], vetor[primeiro]);
}

void ordenacao_troca (int vetor[], int n) {
     int i ;
     for (i = 0 ; i < n-1 ; i++)
          posiciona_menor(vetor, i, n);
}
```

nº comparações: ultimo-primeiro-1
nº de trocas: depende do vetor
máximo: ultimo-primeiro-1
mínimo: 0

i=0:
posiciona_menor(vetor, 0,n) = n-0-1 = n-1
i=1:
posiciona_menor(vetor, 1, n)=n-1-1=n-2
i=2:
posiciona_menor(vetor, 2, n)=n-2-1=n-3
....
i=n-2 (última)
posiciona_menor(vetor,n-2,n)=n-(n-2)-1=1

$$\sum_{i=0}^{n-1} = n(n-1)/2 = O(n^2)$$

Ordenação por trocas - Complexidade

Comparações:

$O(n^2)$

Trocas:

Melhor caso: $O(1)$

Pior caso: $O(n^2)$

Ordenação por trocas - Complexidade

Desafio: Como reduzir número de trocas mantendo a invariante do método?

invariante: após cada iteração i, o i-ésimo elemento estará ordenado corretamente

“Posiciona o menor elemento corretamente, posteriormente o segundo menor...”

Trocas: O(n)

Dica: basta alterar o posiciona menor

Roteiro

Introdução

Métodos iterativos

- Ordenação por trocas
- **Método das bolhas**
- Ordenação por inserção

Métodos recursivos - Divisão e Conquista

- Quick Sort
- Merge Sort

Complexidade algorítmica dos algoritmos de classificação

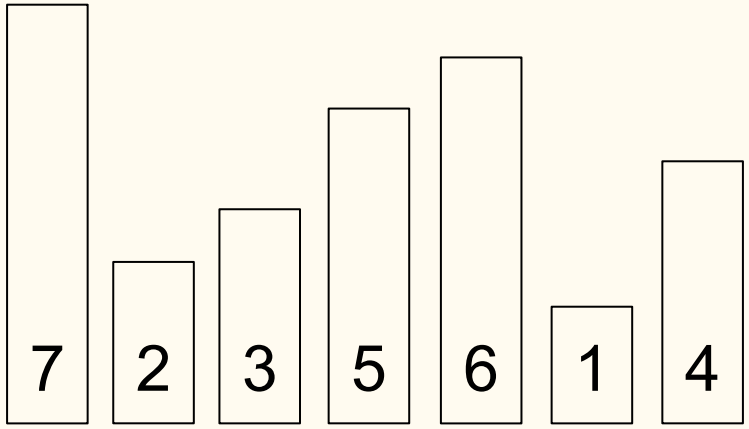
Método das bolhas - Ideia

- O nome do método é uma analogia a bolhas de ar que sobem gradualmente para a superfície
- Método é baseado em sucessivas trocas de valores, até que toda a lista esteja ordenada

- As trocas acontecem com a posição adjacentes
- Ao final de cada iteração, o maior valor estará ordenado

Método das bolhas - Passo a passo

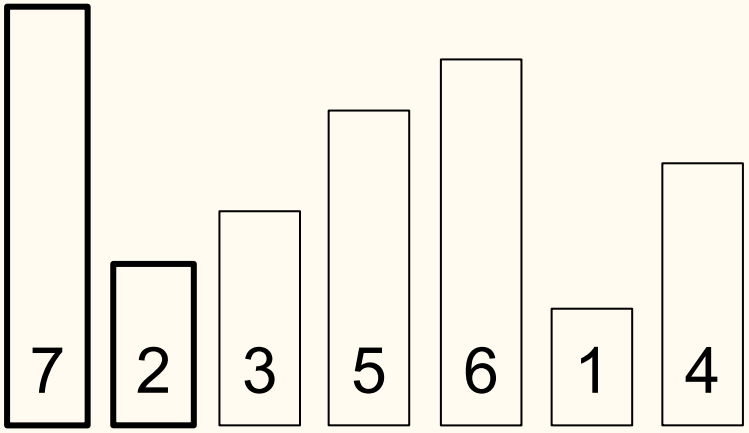
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

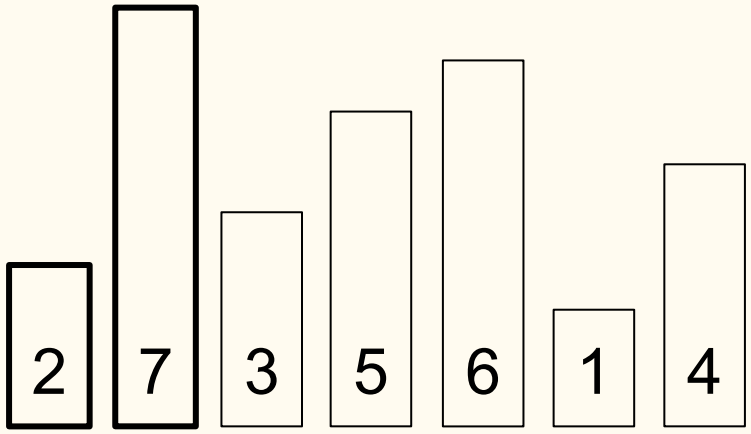
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

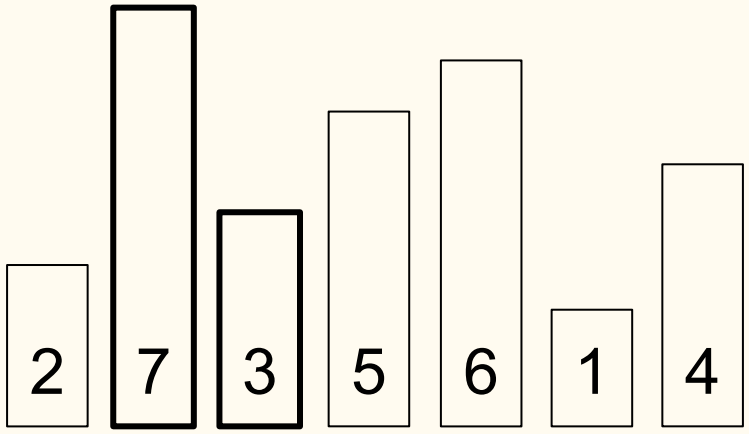
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

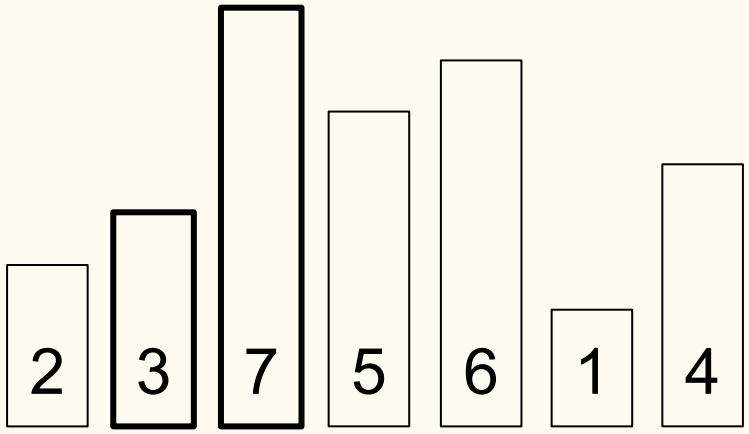
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

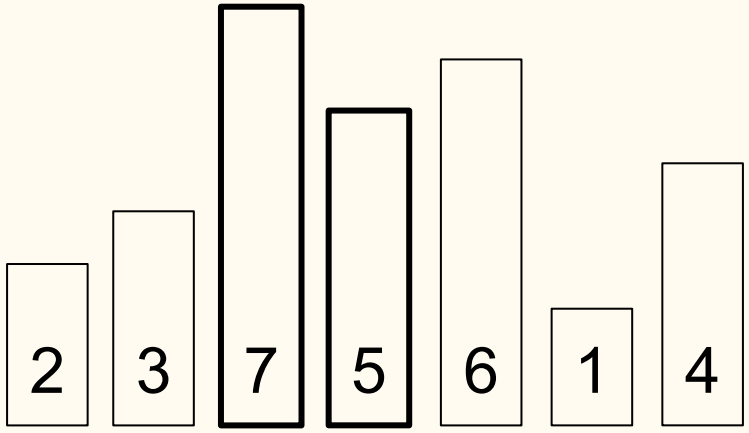
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

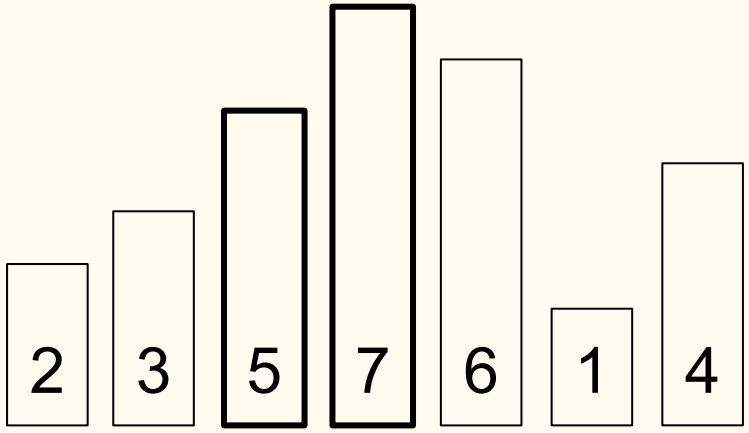
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

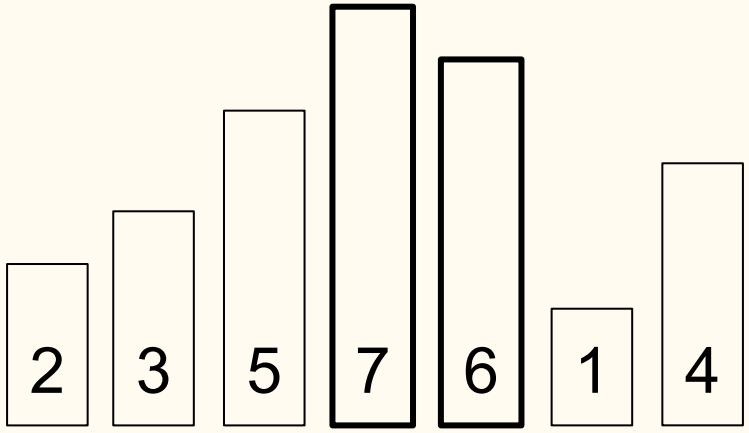
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

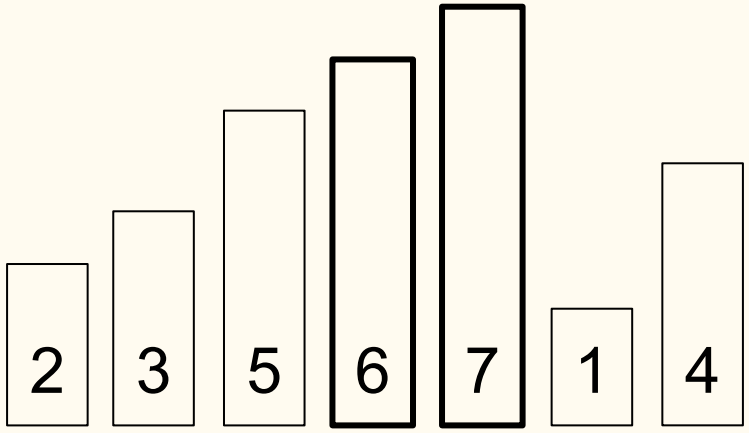
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

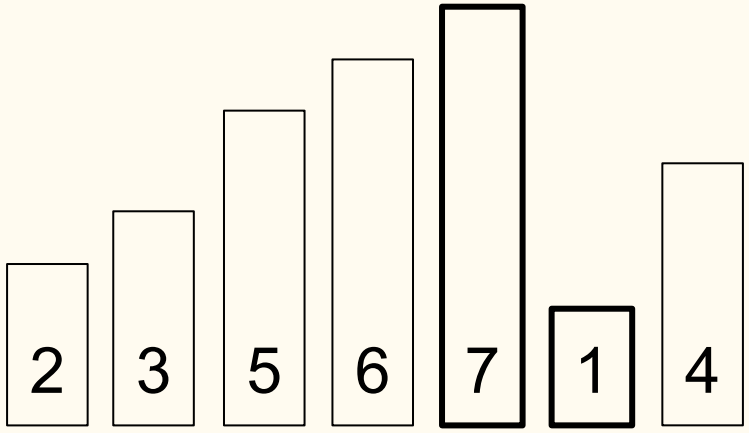
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

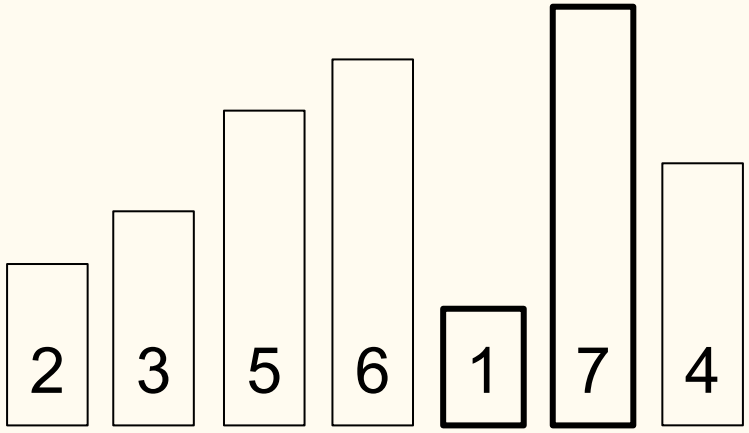
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

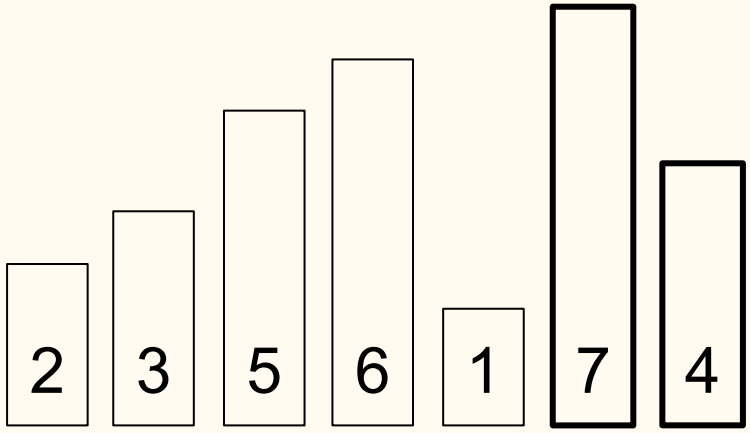
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

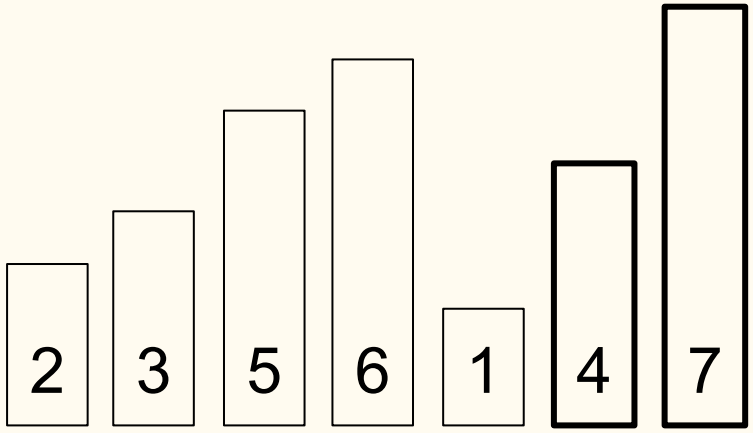
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

Método das bolhas - Passo a passo

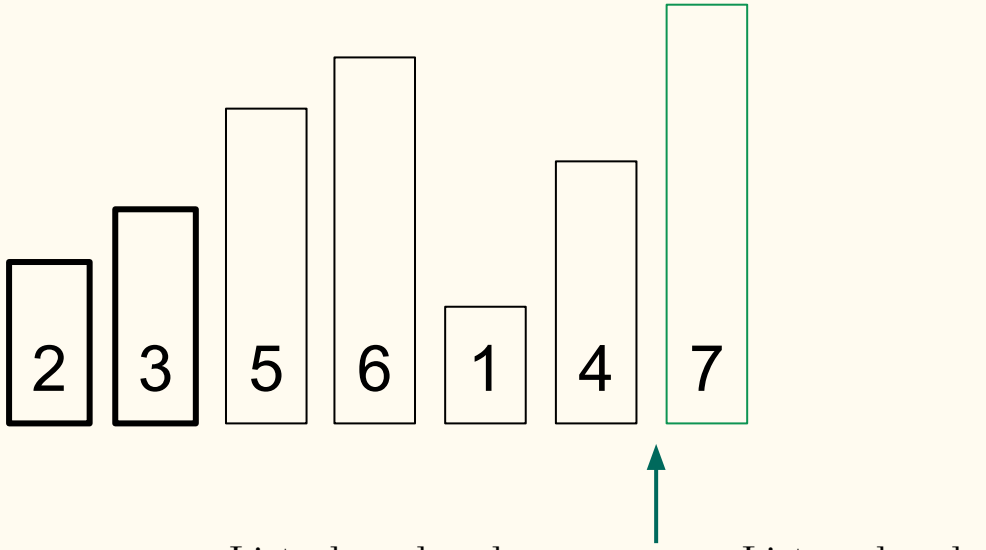
- Percorrer toda a lista e comparar 2 a 2
- Troca se valor da esquerda é menor que direita
- 1ª iteração



Lista desordenada

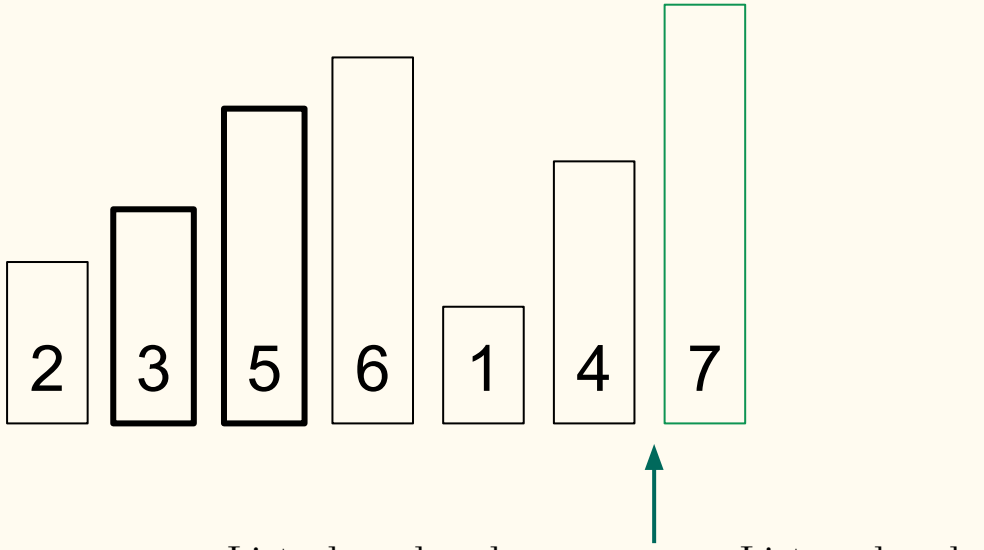
Método das bolhas - Passo a passo

- Próxima iteração não precisa ir até o último elemento,
- Percorrer apenas a lista desordenada
- Iteração 2



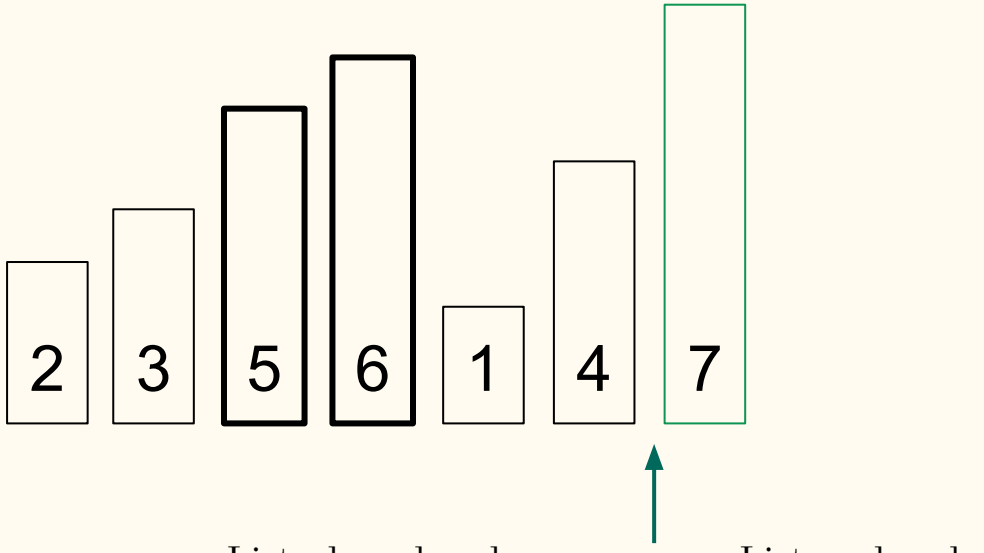
Método das bolhas - Passo a passo

- Próxima iteração não precisa ir até o último elemento,
- Percorrer apenas a lista desordenada
- Iteração 2



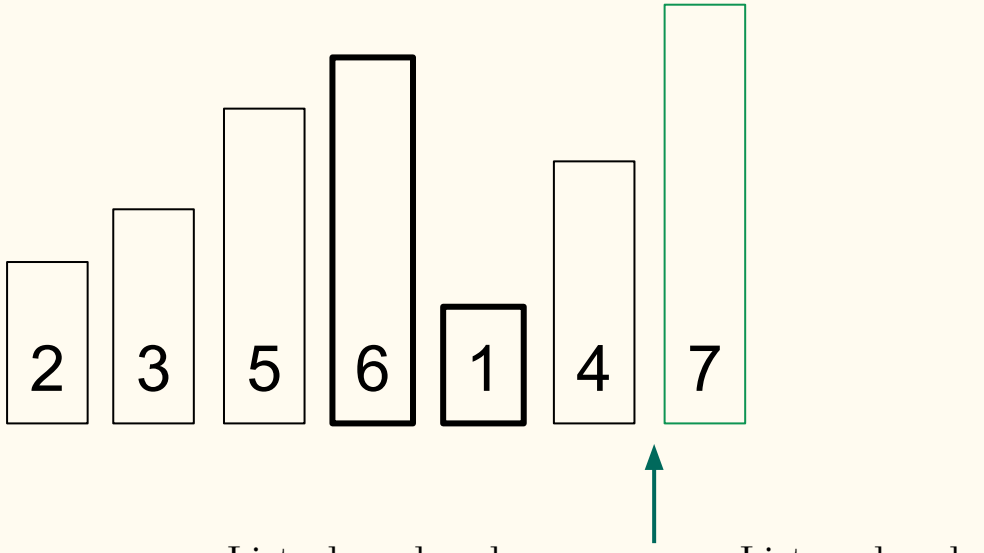
Método das bolhas - Passo a passo

- Próxima iteração não precisa ir até o último elemento,
- Percorrer apenas a lista desordenada
- Iteração 2



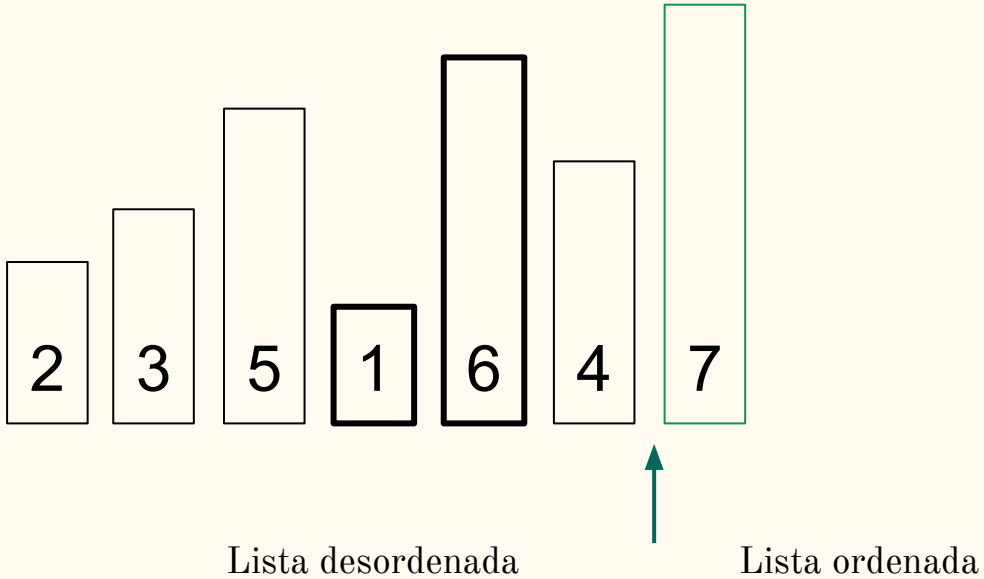
Método das bolhas - Passo a passo

- Próxima iteração não precisa ir até o último elemento,
- Percorrer apenas a lista desordenada
- Iteração 2



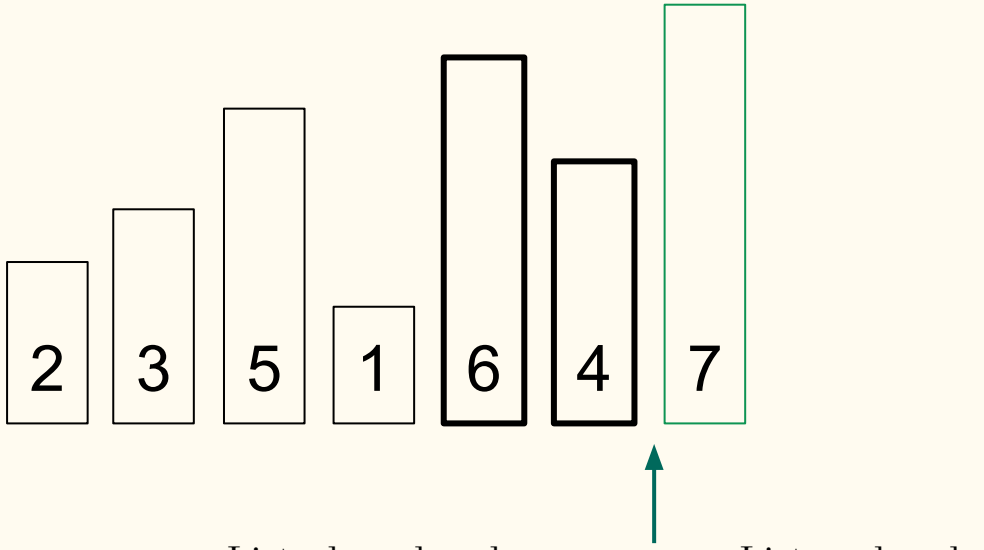
Método das bolhas - Passo a passo

- Próxima iteração não precisa ir até o último elemento,
- Percorrer apenas a lista desordenada
- Iteração 2



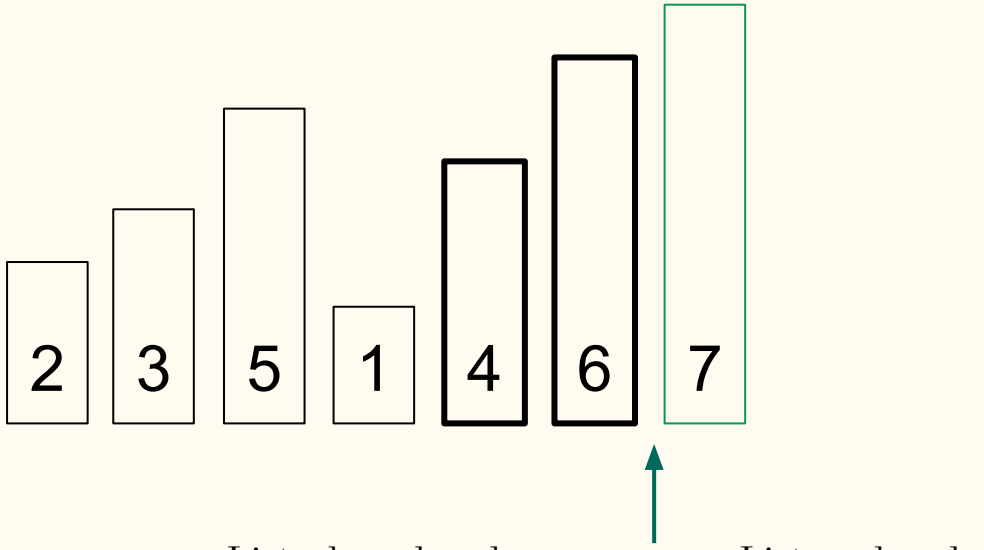
Método das bolhas - Passo a passo

- Próxima iteração não precisa ir até o último elemento,
- Percorrer apenas a lista desordenada
- Iteração 2



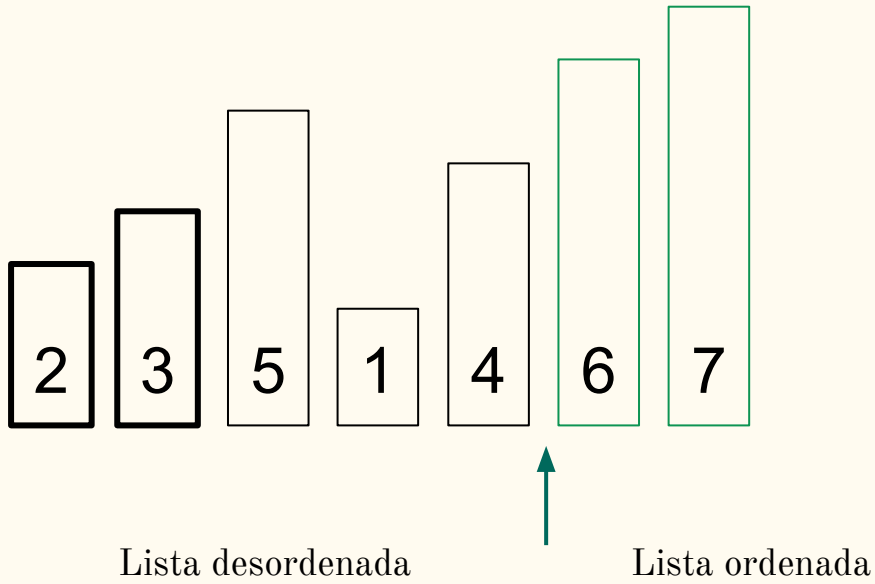
Método das bolhas - Passo a passo

- Próxima iteração não precisa ir até o último elemento,
- Percorrer apenas a lista desordenada
- Iteração 2



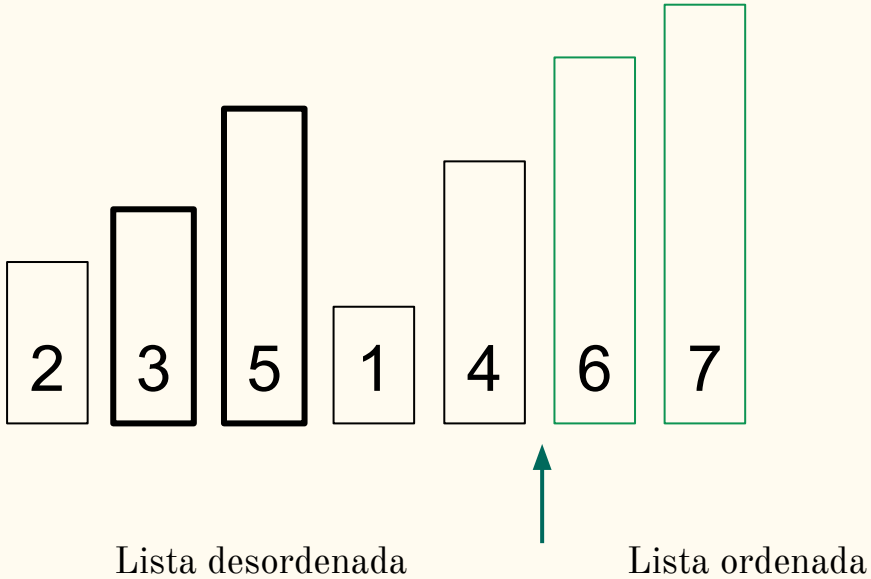
Método das bolhas - Passo a passo

- Iteração 3



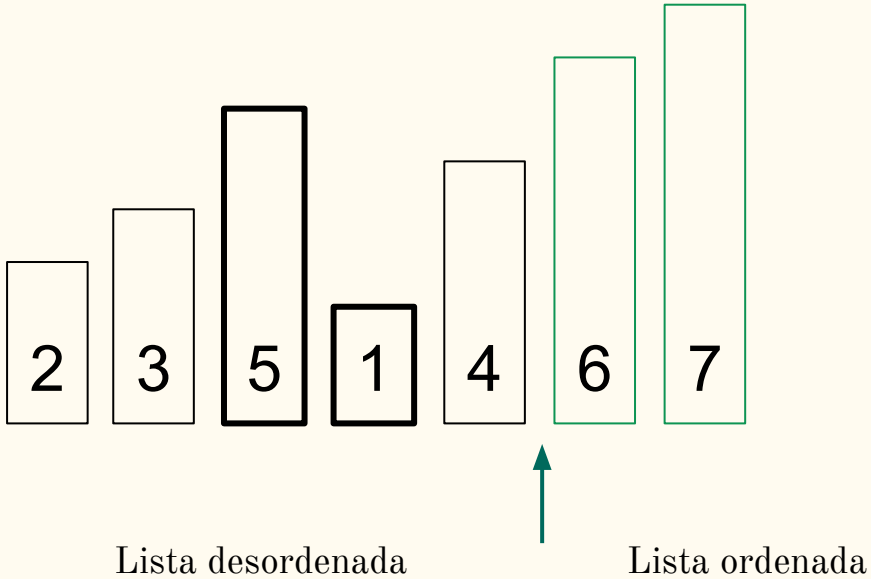
Método das bolhas - Passo a passo

- Iteração 3



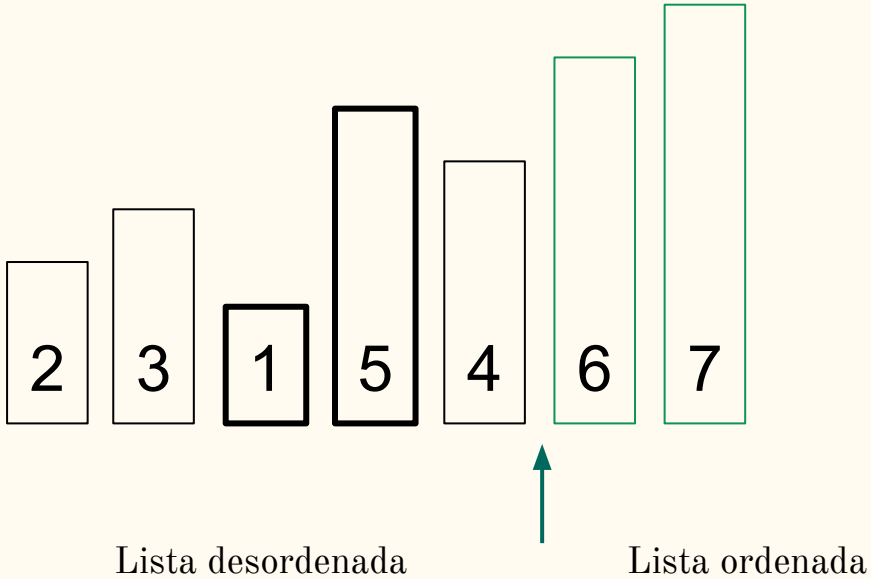
Método das bolhas - Passo a passo

- Iteração 3



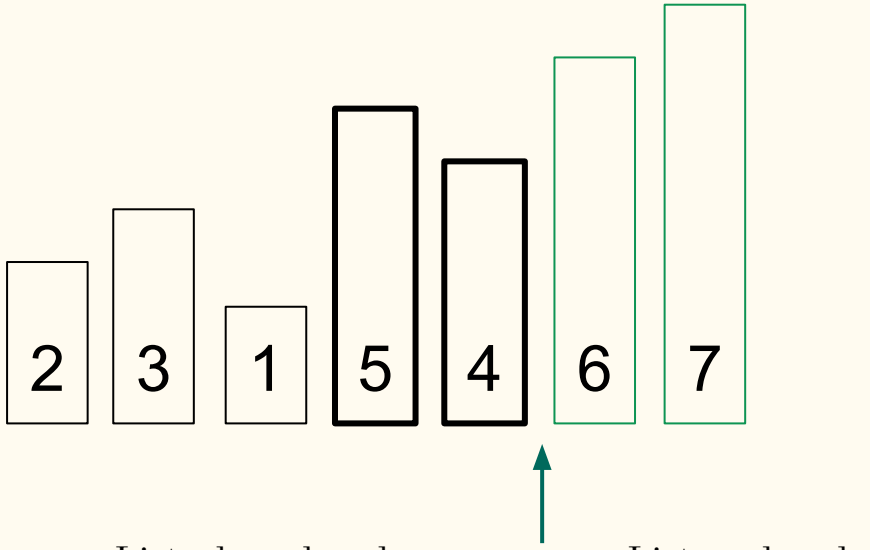
Método das bolhas - Passo a passo

- Iteração 3



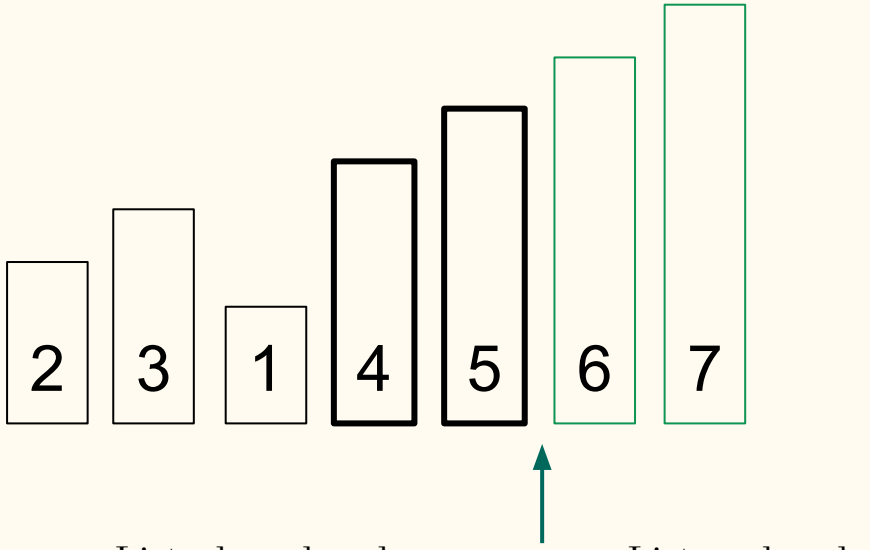
Método das bolhas - Passo a passo

- Iteração 3



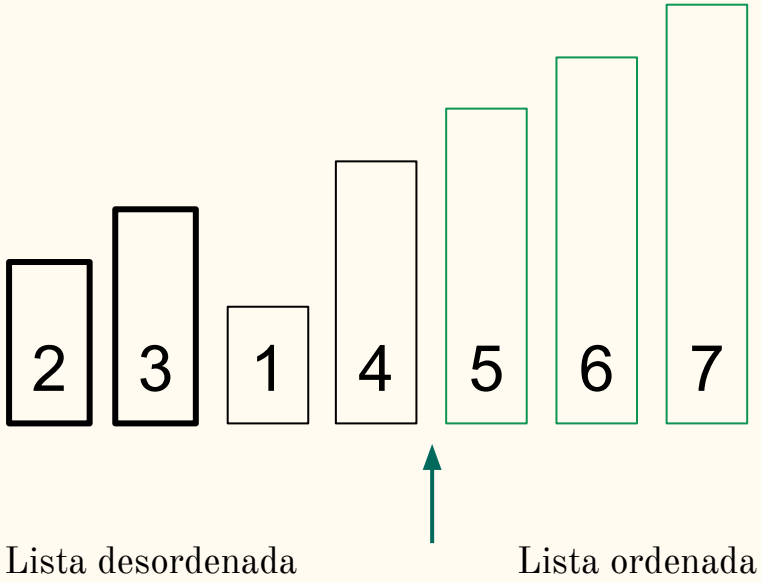
Método das bolhas - Passo a passo

- Iteração 3



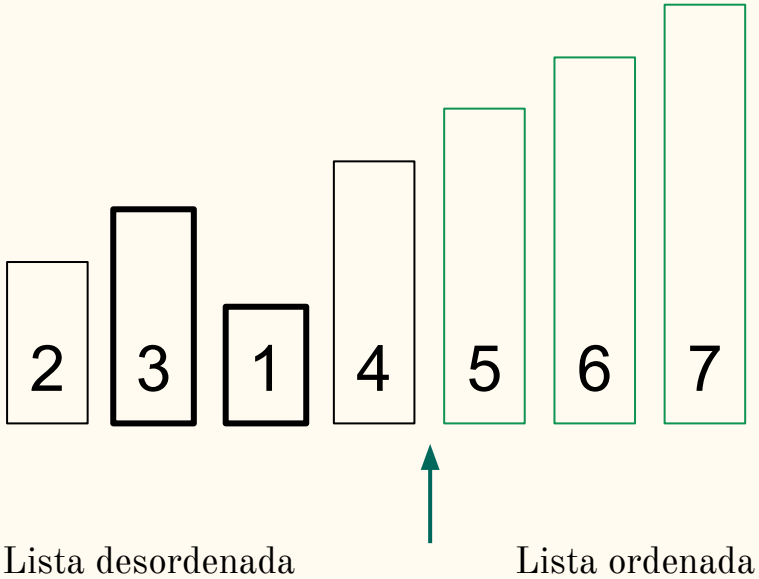
Método das bolhas - Passo a passo

- Iteração 4



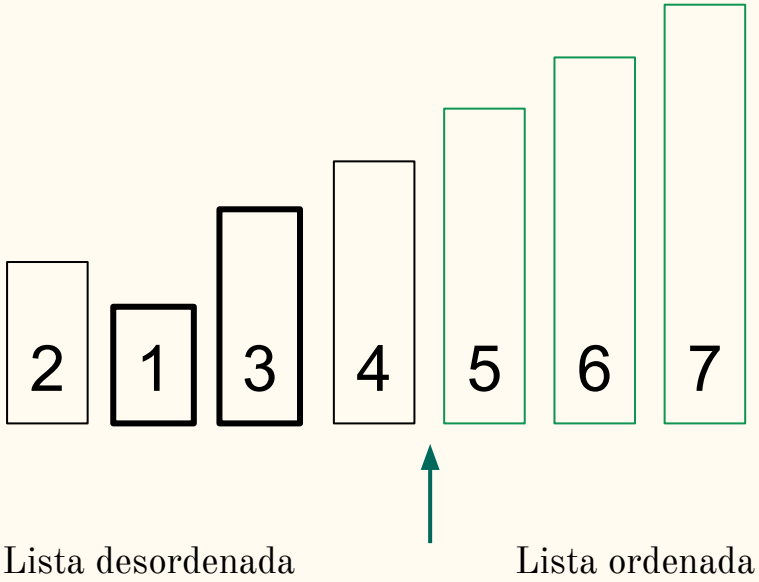
Método das bolhas - Passo a passo

- Iteração 4



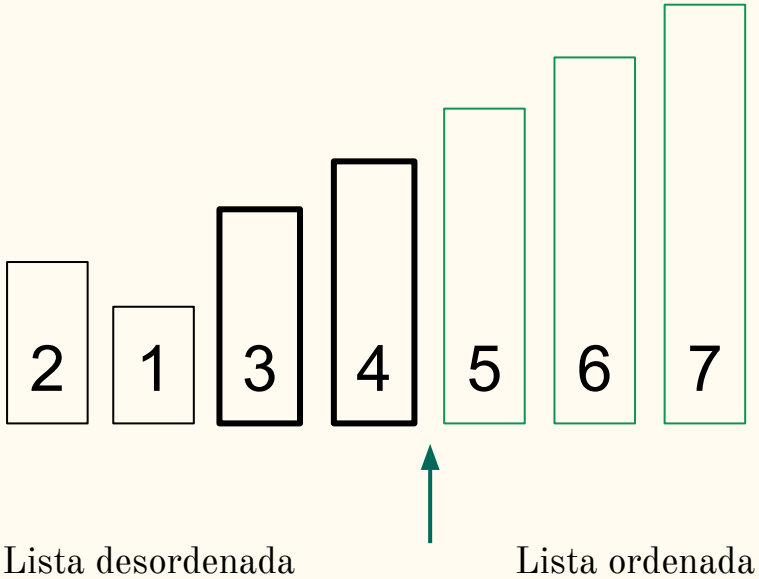
Método das bolhas - Passo a passo

- Iteração 4



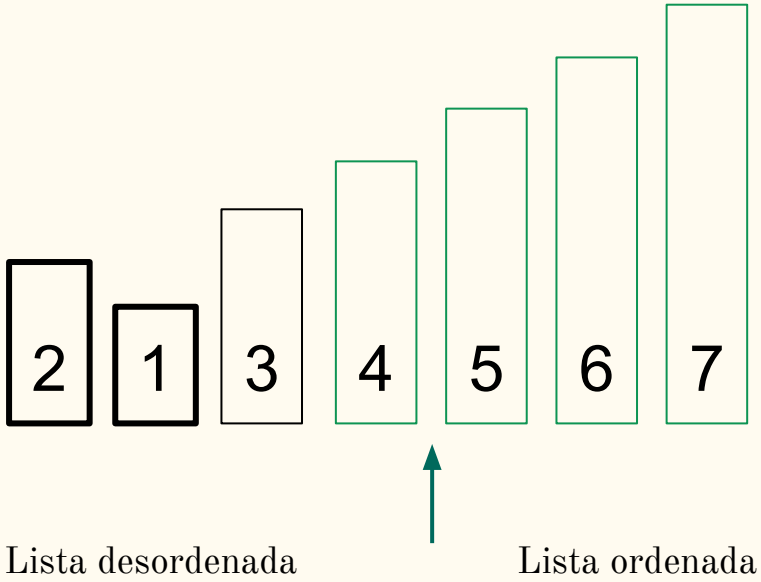
Método das bolhas - Passo a passo

- Iteração 4



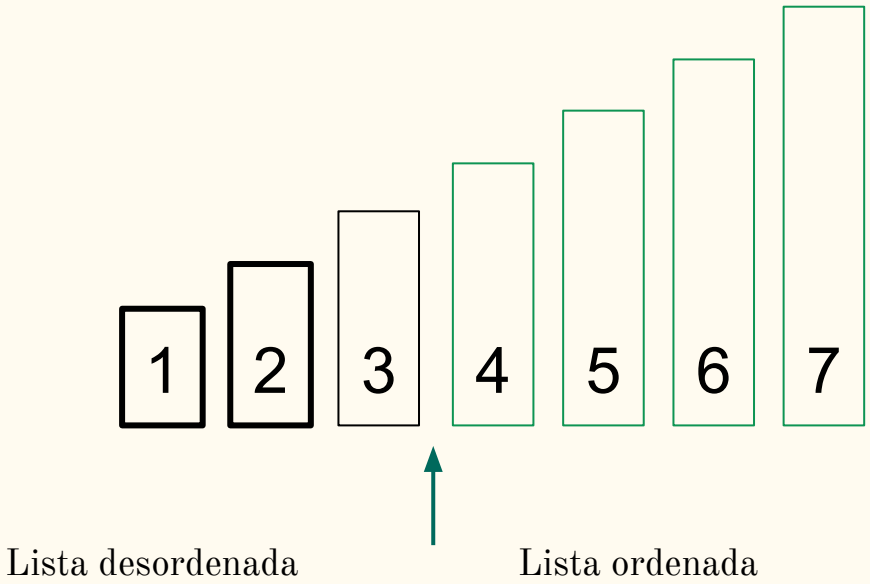
Método das bolhas - Passo a passo

- Iteração 4



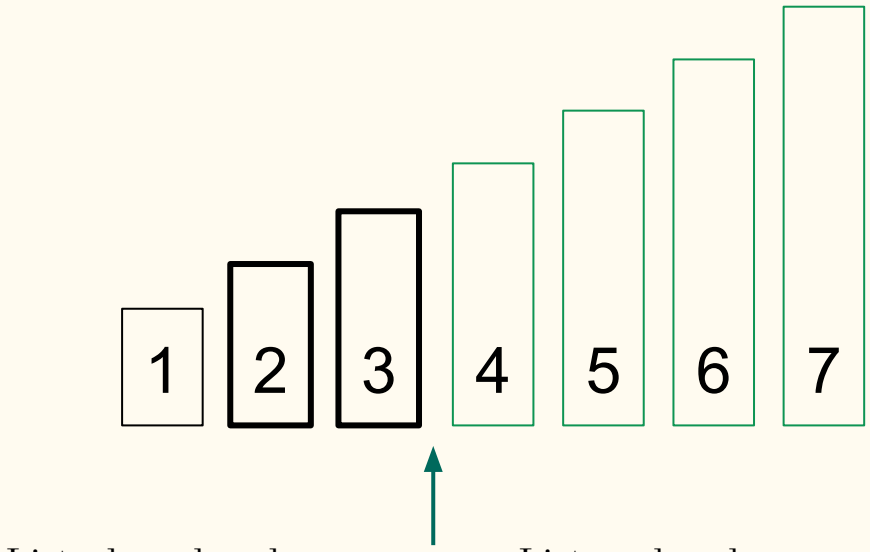
Método das bolhas - Passo a passo

- Iteração 5



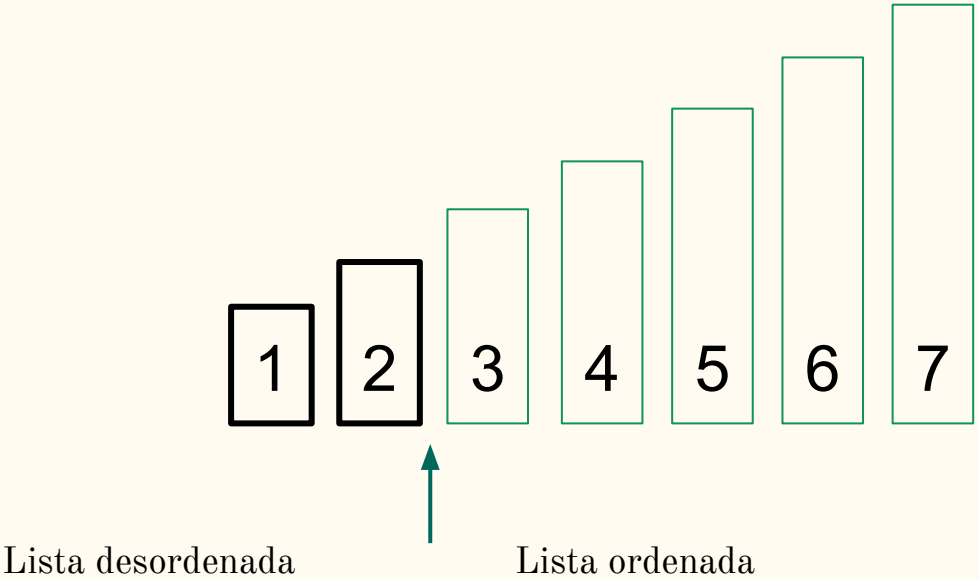
Método das bolhas - Passo a passo

- Iteração 5

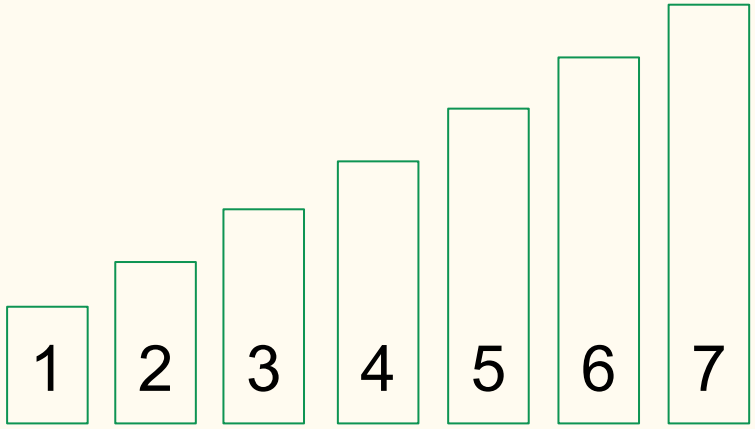


Método das bolhas - Passo a passo

- Fim da 5ª iteração
- Importante destacar que o algoritmo vai fazer todas as iterações mesmo que a lista já esteja ordenada.



Método das bolhas - Passo a passo



Lista ordenada

Método das bolhas - Codificando...

```
void bubble_sort (int vetor[], int n) {
    int j, i;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-1-i; j++)
            if (vetor[j] > vetor[j+1])
                troca (&vetor[j], &vetor[j+1]);
}
```

Método das bolhas - Complexidade

```
void bubble_sort (int vetor[], int n) {
    int j, i;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-1-i; j++)
            if (vetor[j] > vetor[j+1])
                troca (&vetor[j], &vetor[j+1]);
}
```

Método das bolhas - Complexidade

```
void bubble_sort (int vetor[], int n) {
    int j, i;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-1-i; j++)
            if (vetor[j] > vetor[j+1])  1
                troca (&vetor[j], &vetor[j+1]);
}
```

Comparações:
i=0:
n-1
i=1
n-2
i=2
n-3
...
i=n-2
n-1-(n-2)=1

Método das bolhas - Complexidade

```
void bubble_sort (int vetor[], int n) {
    int j, i;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-1-i; j++)
            if (vetor[j] > vetor[j+1])  1
                troca (&vetor[j], &vetor[j+1]);
}
```

Comparações:
i=0:
n-1
i=1
n-2
i=2
n-3
...
i=n-2
n-1-(n-2)=1

$$\sum_{i=0}^{n-1} = n(n-1)/2 = O(n^2)$$

Método das bolhas - Complexidade

Comparações: $O(n^2)$

Trocas:

Melhor caso: $O(1)$

Pior caso: $O(n^2)$

```
void bubble_sort (int vetor[], int n) {
     int j, i;
     for (i = 0; i < n-1; i++)
          for (j = 0; j < n-1-i; j++)
               if (vetor[j] > vetor[j+1])
                    troca (&vetor[j], &vetor[j+1]);
}
```

Método das bolhas - Variações

- Verificação de vetor ordenado:
  - Se não houve troca em 1 iteração, já está ordenado, pode parar

- ordenação oscilante:
  - Valores grandes chegam rápido na posição correta
  - Valores pequenos demoram várias iterações
  - Ordenação oscilante inverte a direção entre cada iteração, reduzindo número de trocas.

- Variações ainda tem complexidade $O(n^2)$, melhorando complexidade no melhor caso, reduz coeficiente em caso médio, melhora quando está quase ordenado

Roteiro

Introdução

Métodos iterativos
- Método das bolhas
- Ordenação por trocas
- **Ordenação por inserção**

Métodos recursivos - Divisão e Conquista
- Merge Sort
- Quick Sort

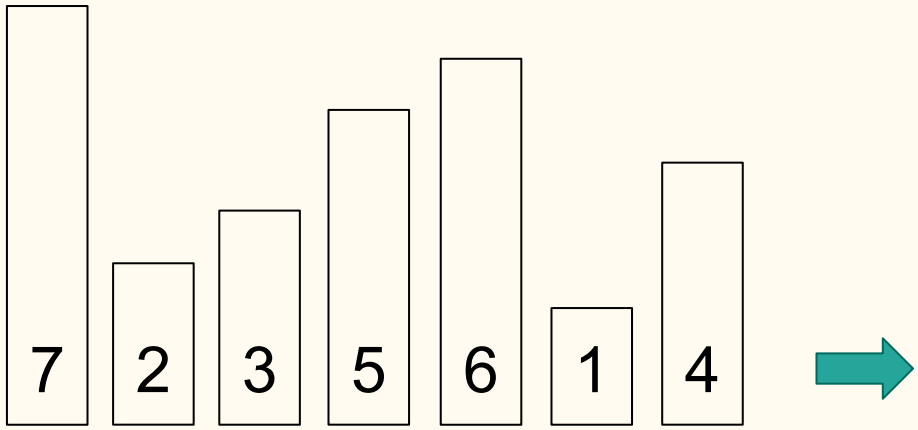
Complexidade algorítmica dos algoritmos de classificação

Ordenação por inserção - Ideia

- A ideia do método é baseada na ordenação de cartas de baralho em uma mão: a partir de uma mão ordenada, encontrar o local na nova carta

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

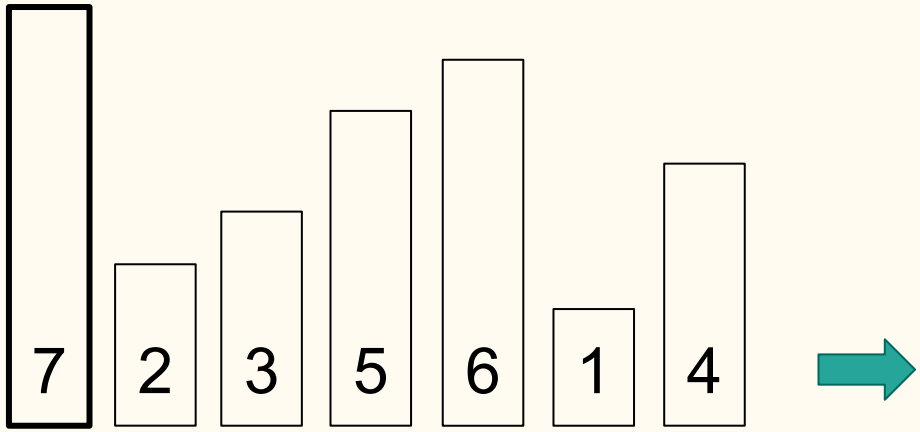


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

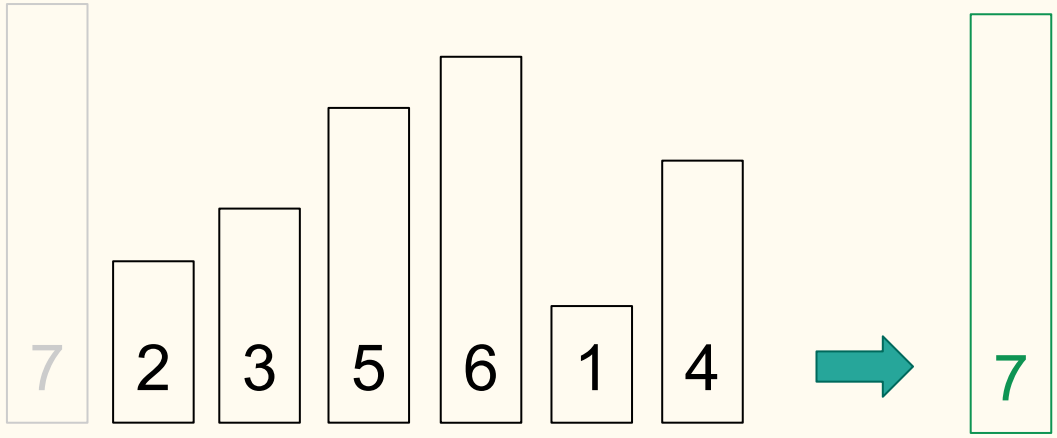


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

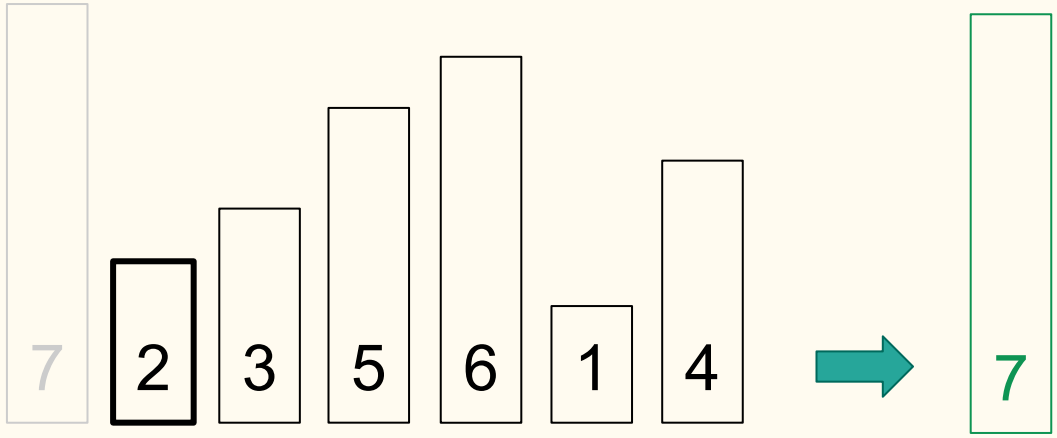


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

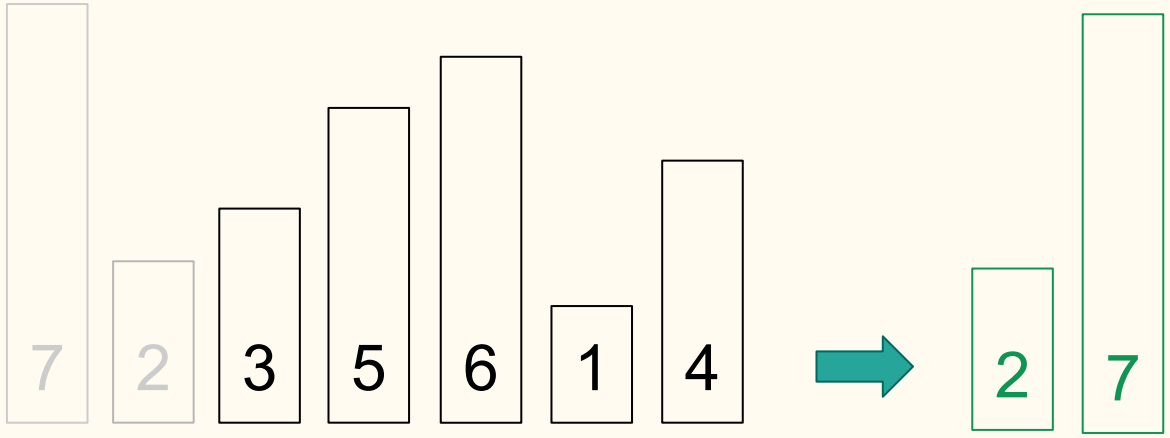


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

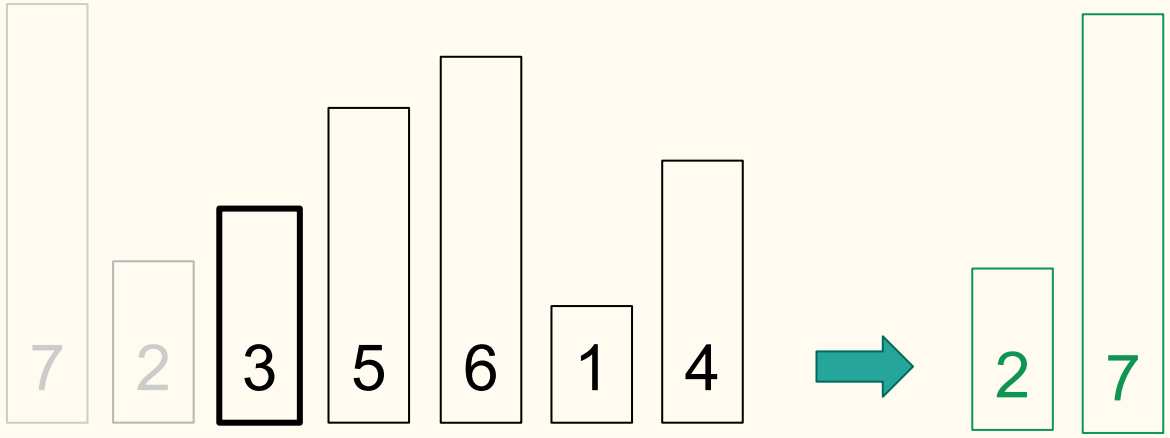


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

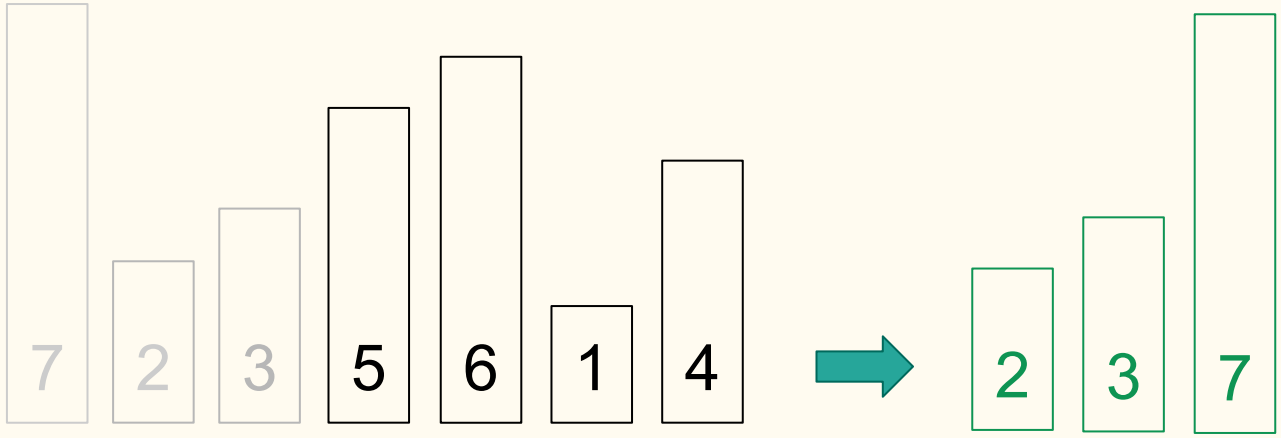


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

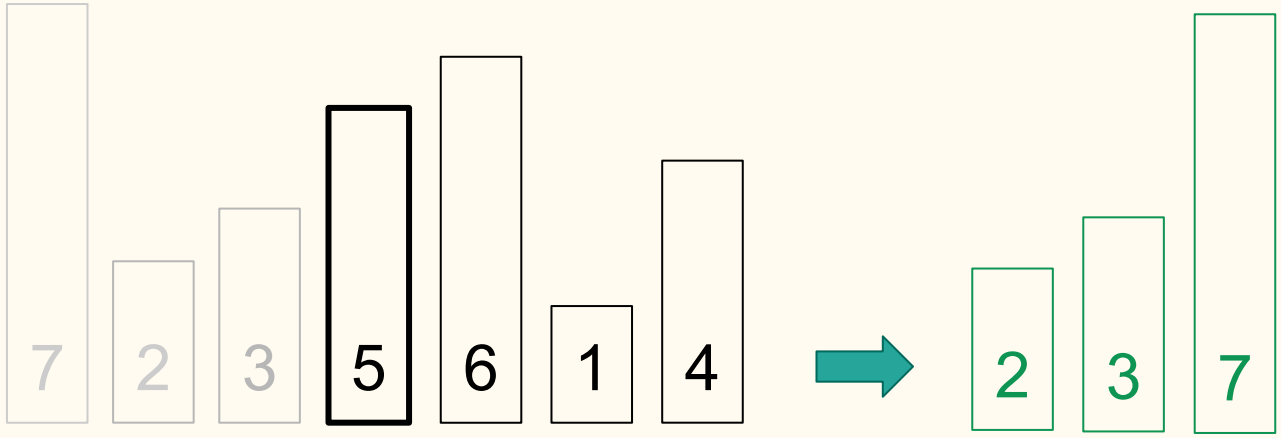


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

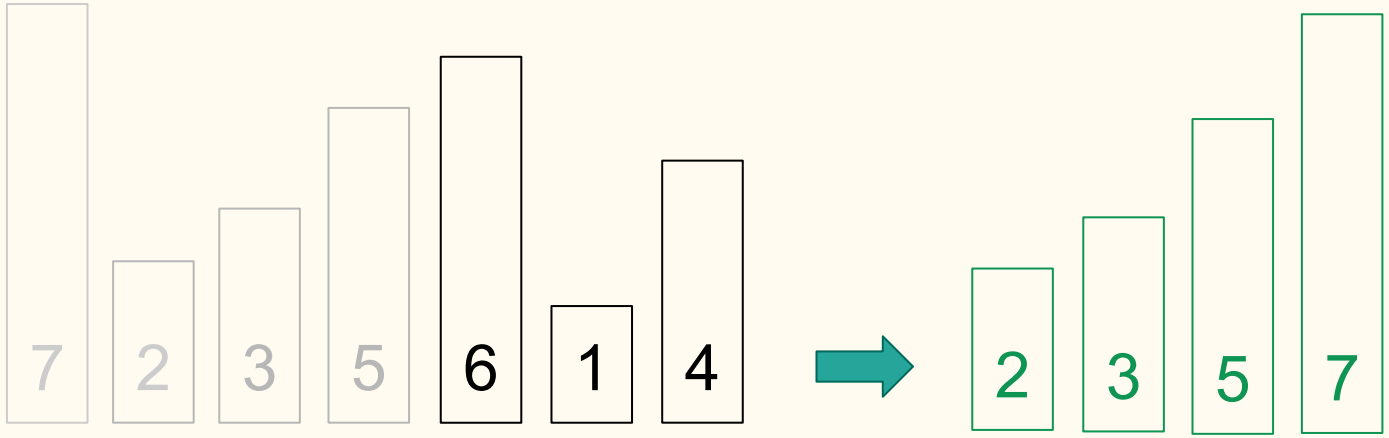


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

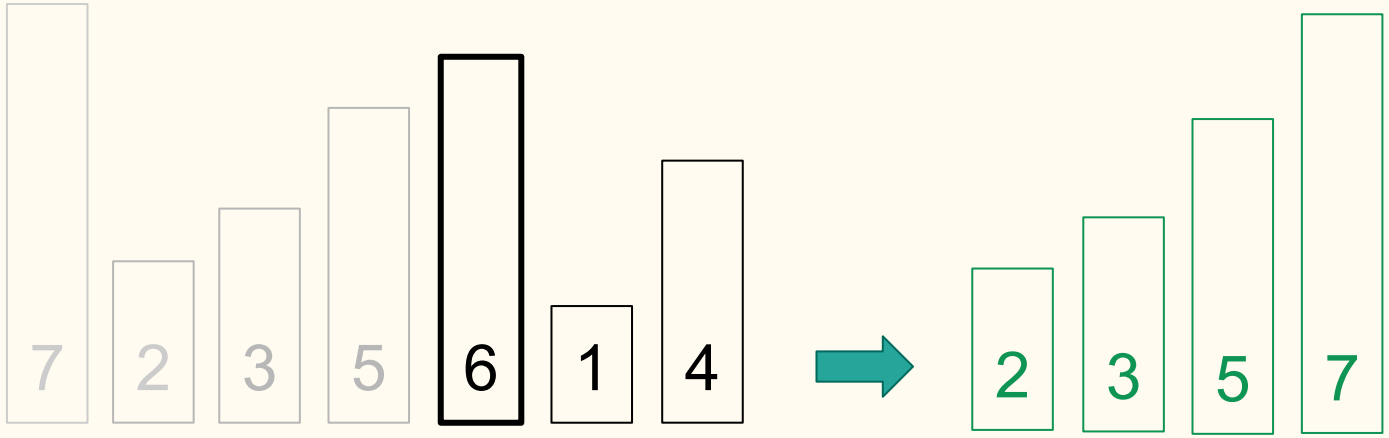


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

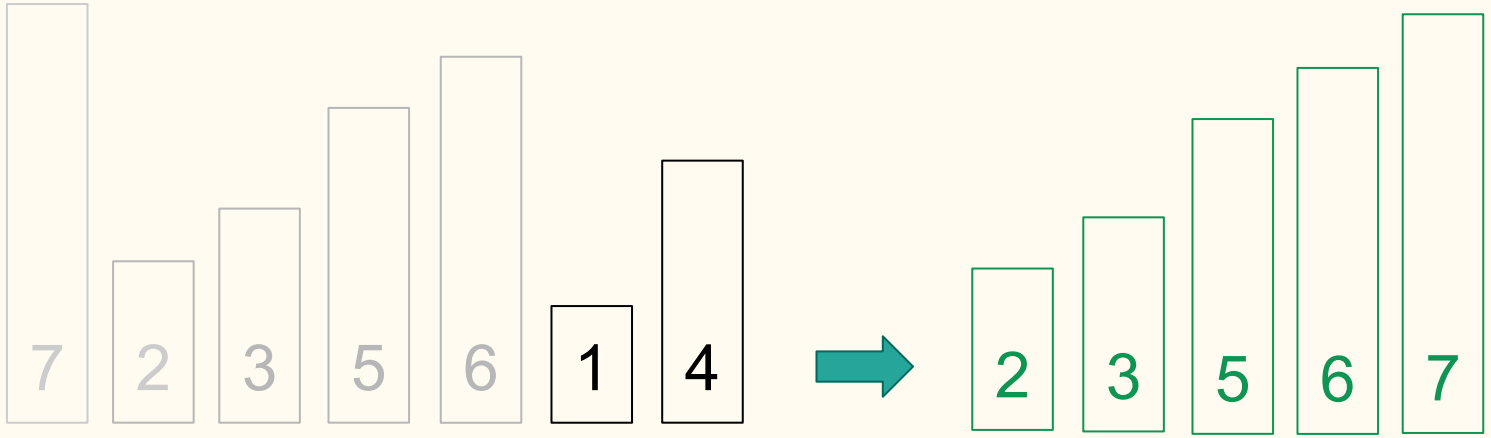


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

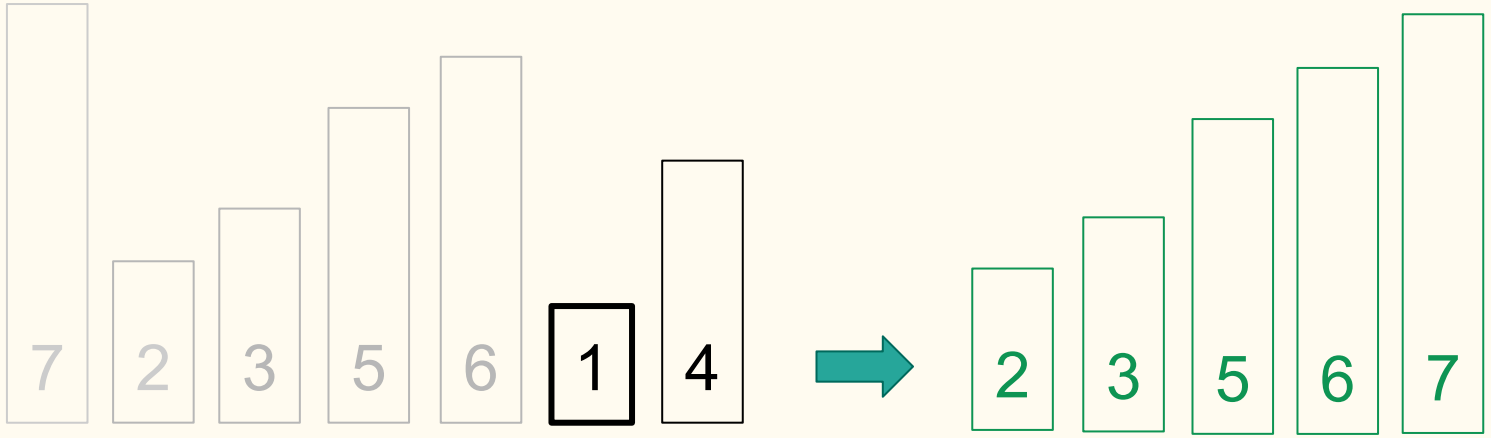


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

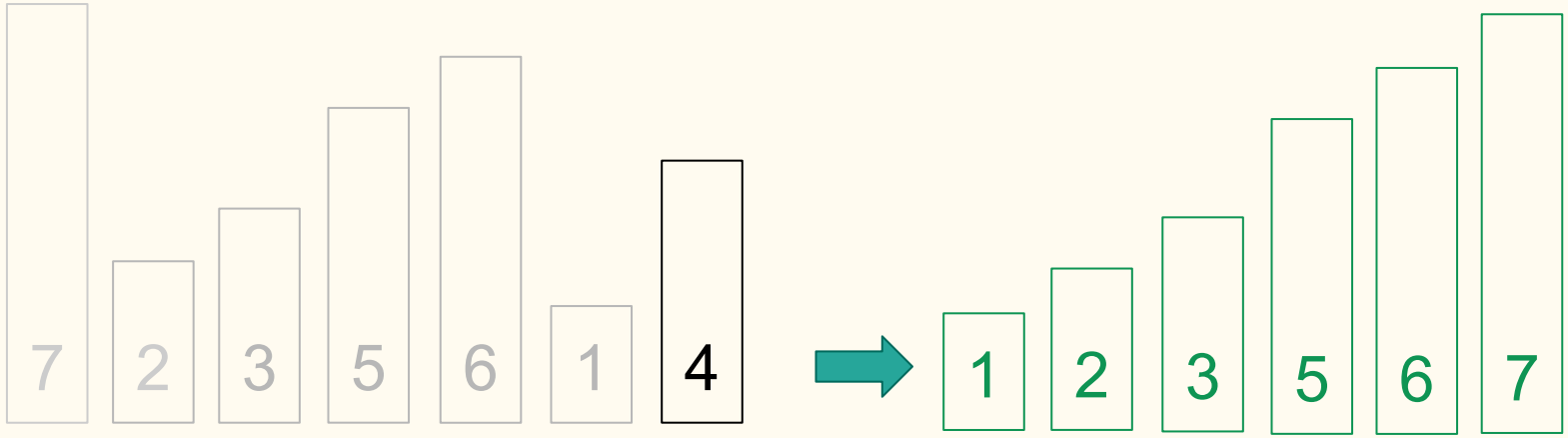


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

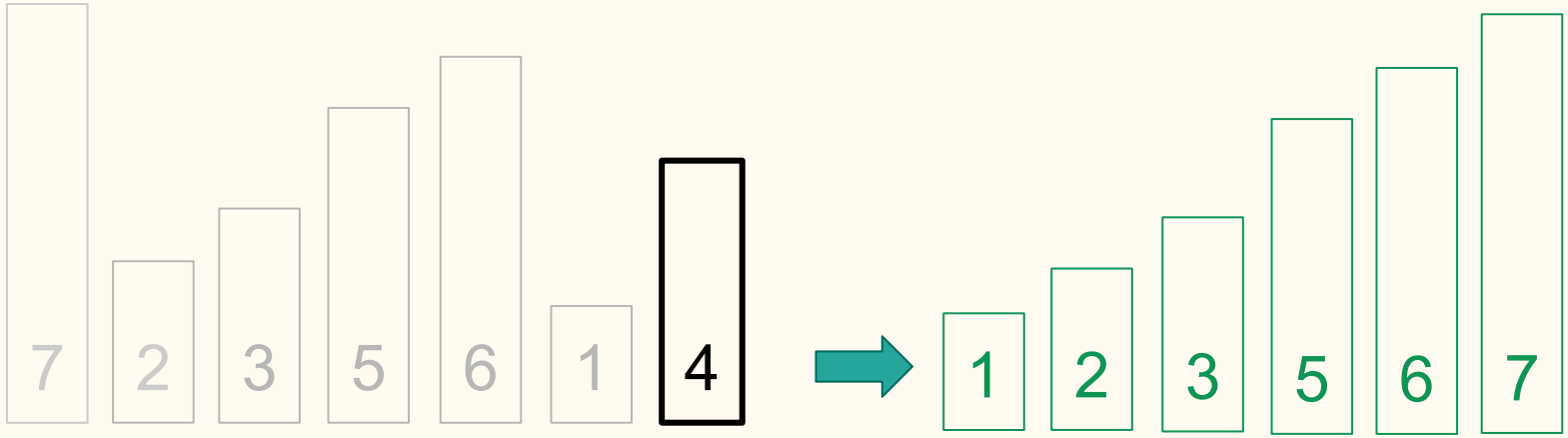


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta

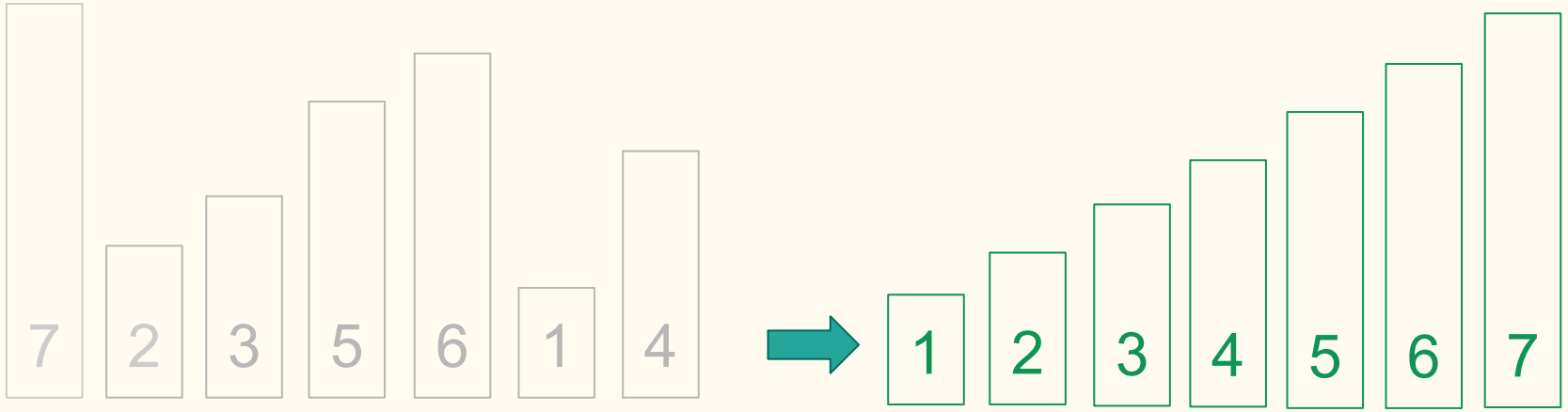


Lista desordenada

Lista ordenada

Ordenação por inserção - Passo a passo

- A partir de uma mão/lista vazia inserir o elemento 1 a 1, na sua posição correta



Lista desordenada

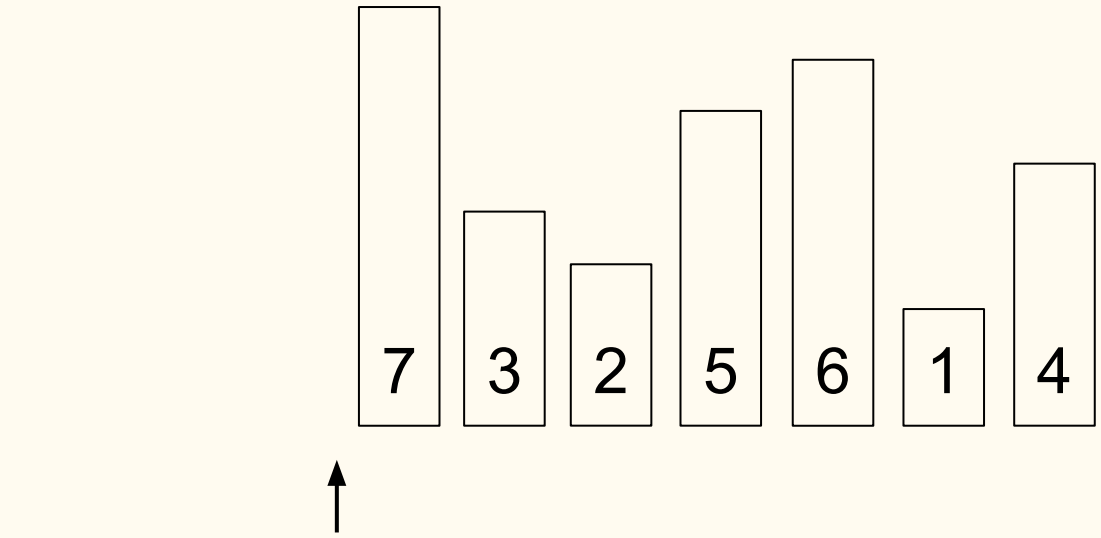
Lista ordenada

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?

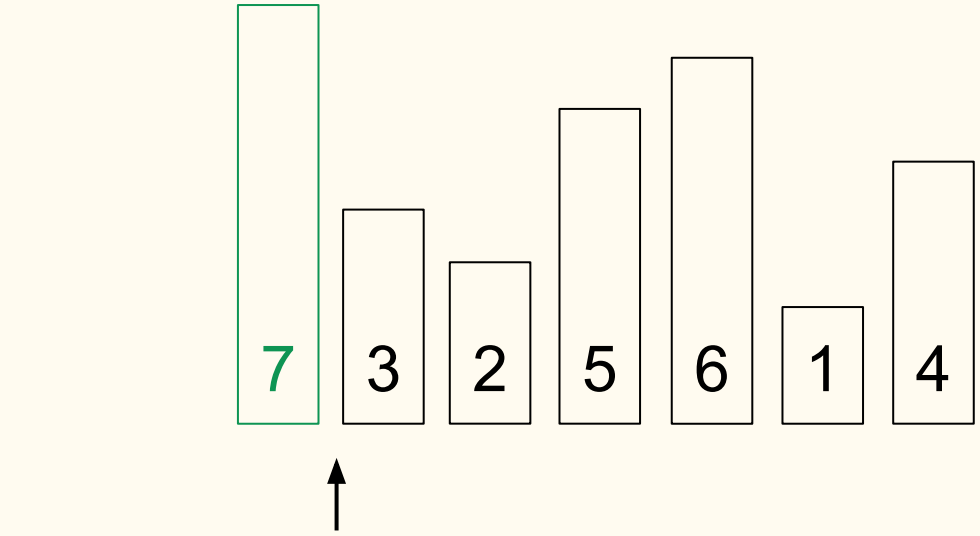
Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Ordenação por inserção - Passo a passo no mesmo vetor

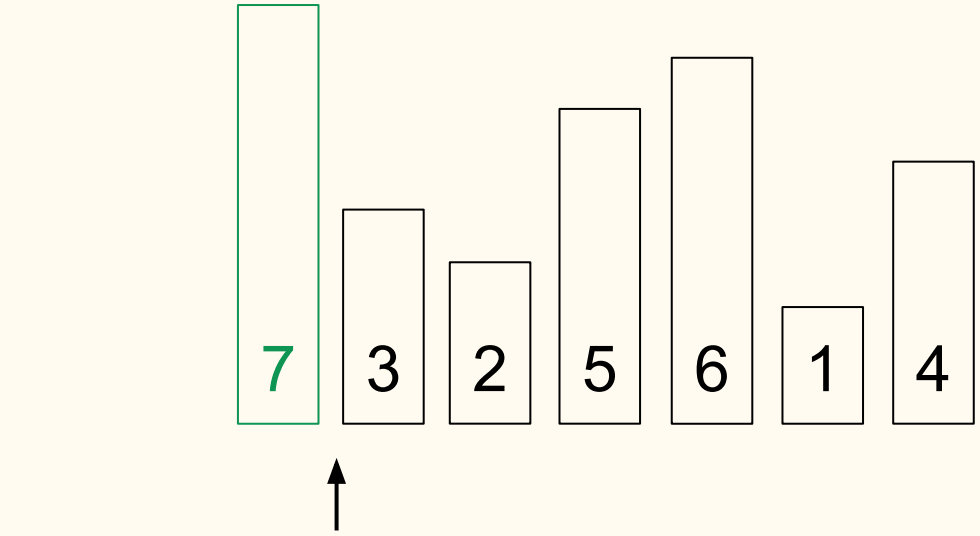
- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



O primeiro elemento está ordenado!

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

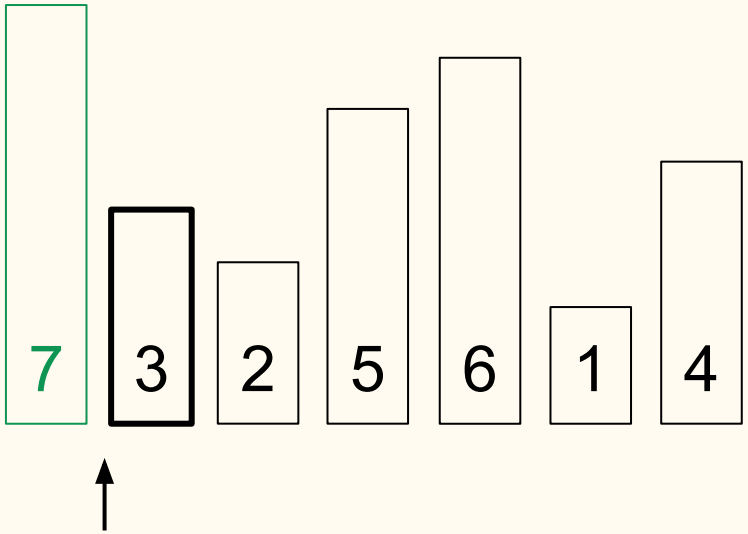
2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

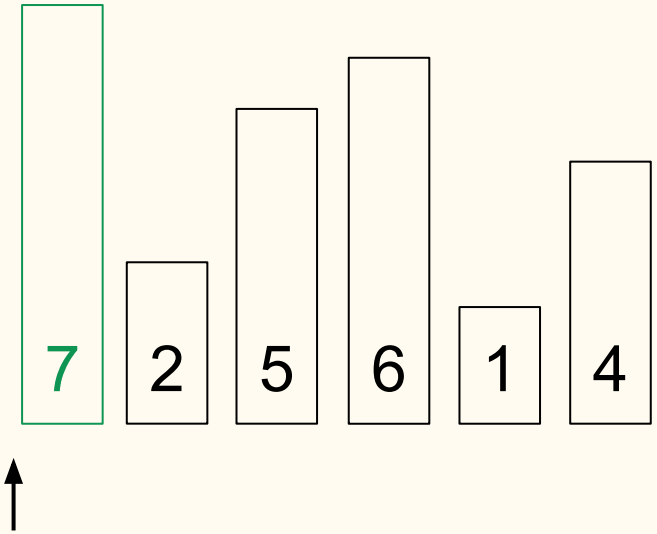
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 3

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Parte ordenada

Parte desordenada

Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

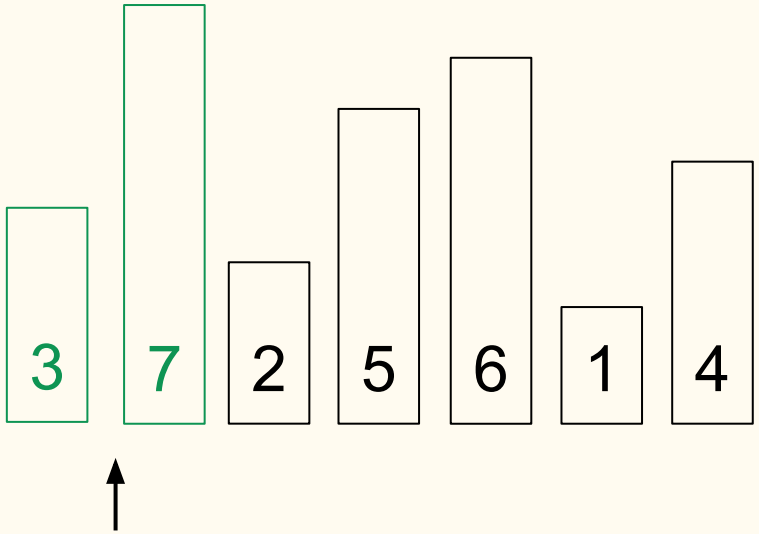
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 3

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Parte ordenada

Parte desordenada

Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

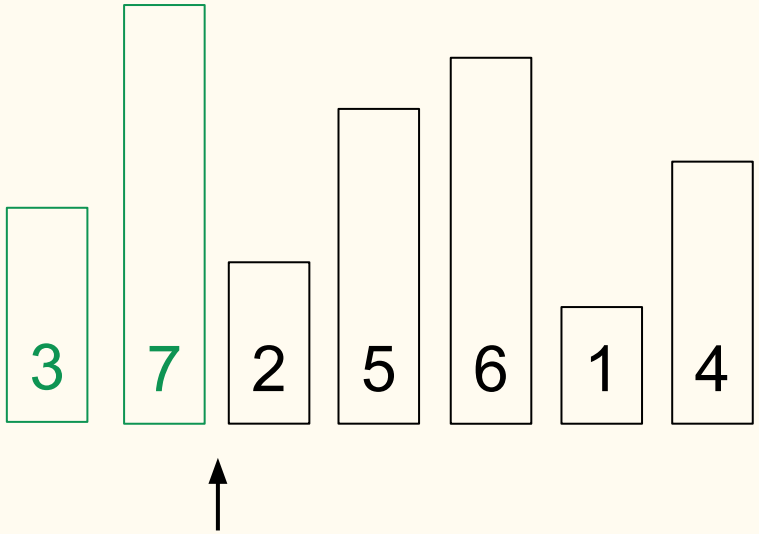
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 3

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

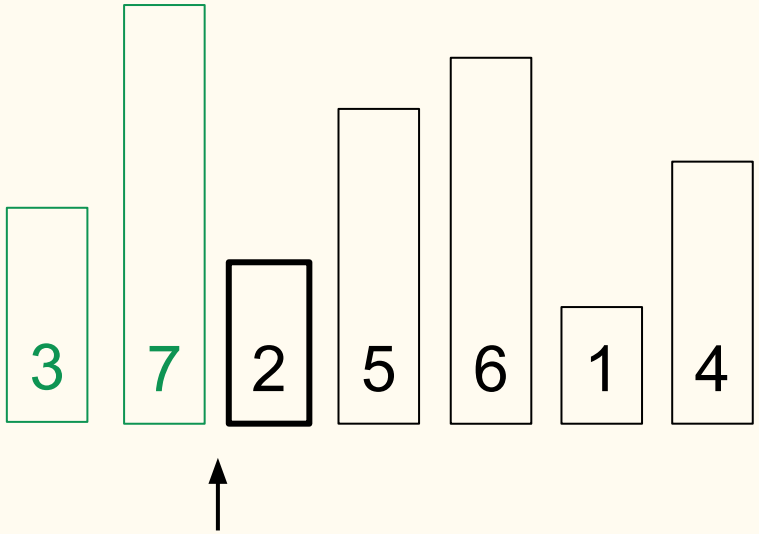
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

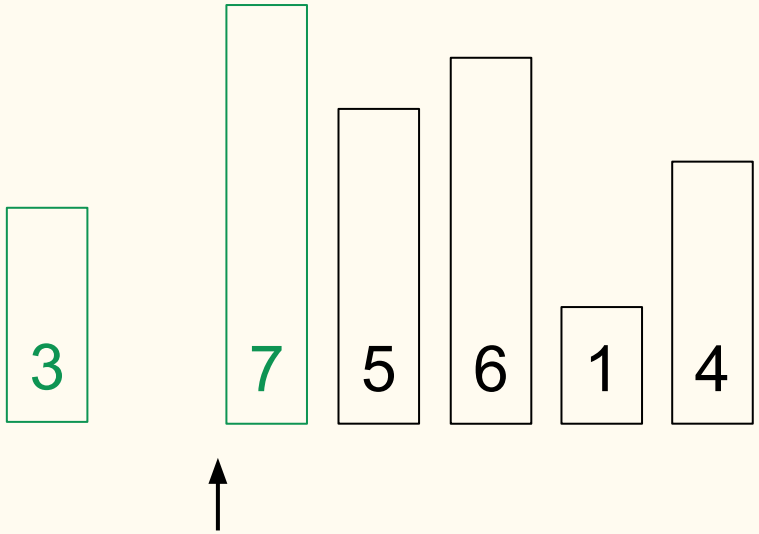
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 2

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

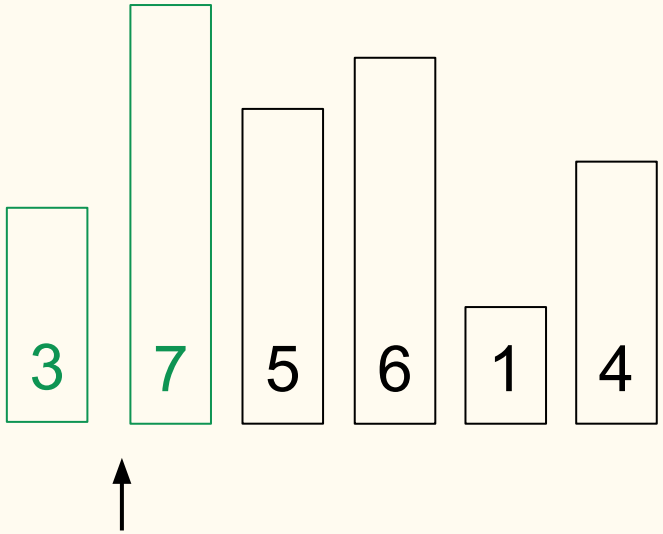
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 2

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

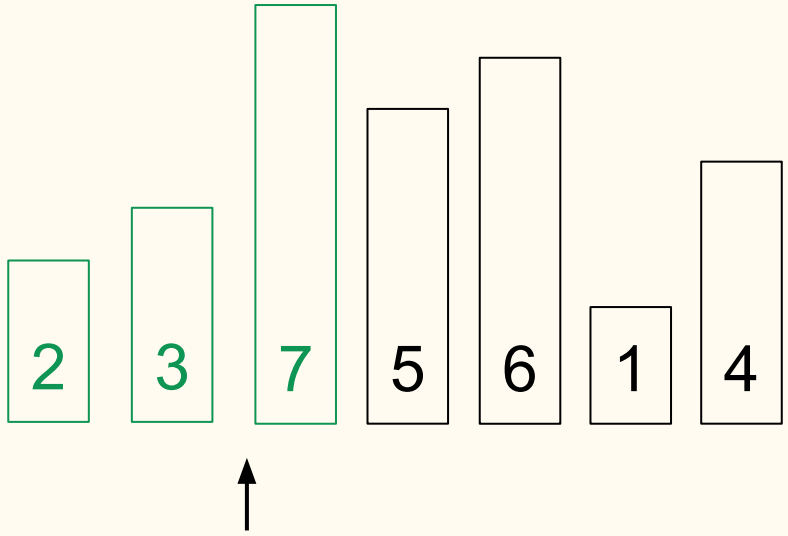
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 2

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

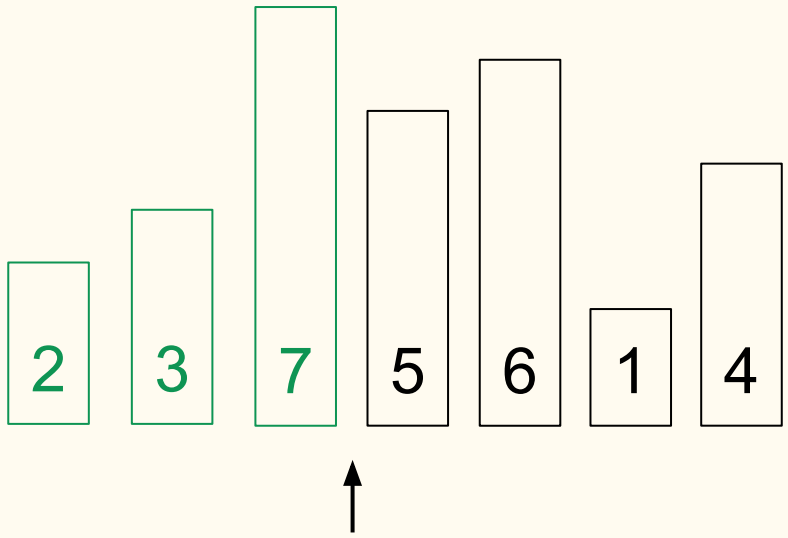
3. Deslocar 1 posição para todos maior que `elemento`

4. **Inserir `elemento`**

`elemento =` 2

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

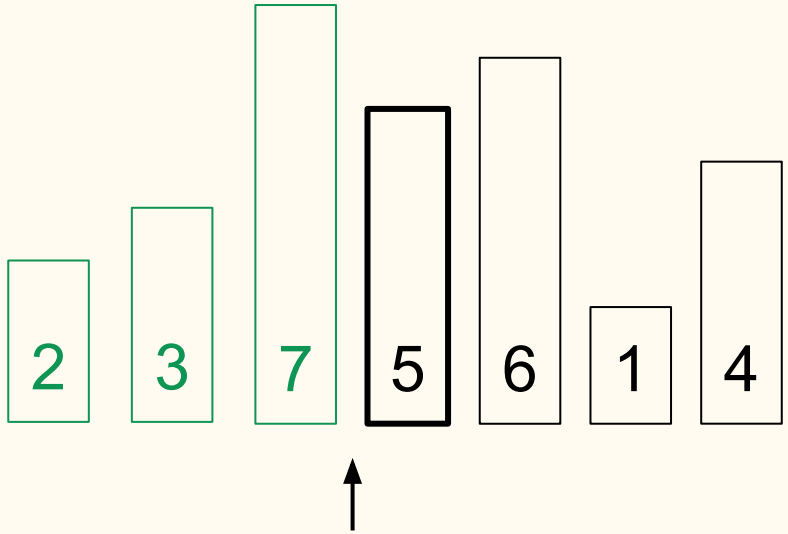
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

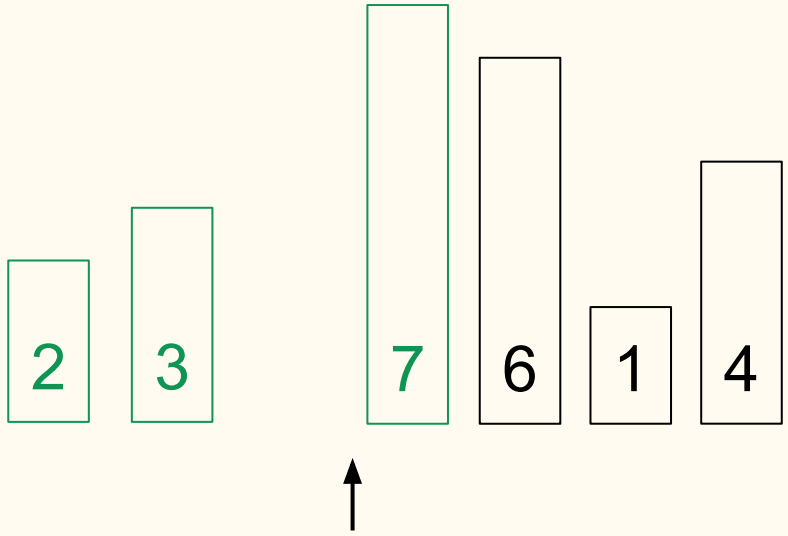
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 5

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

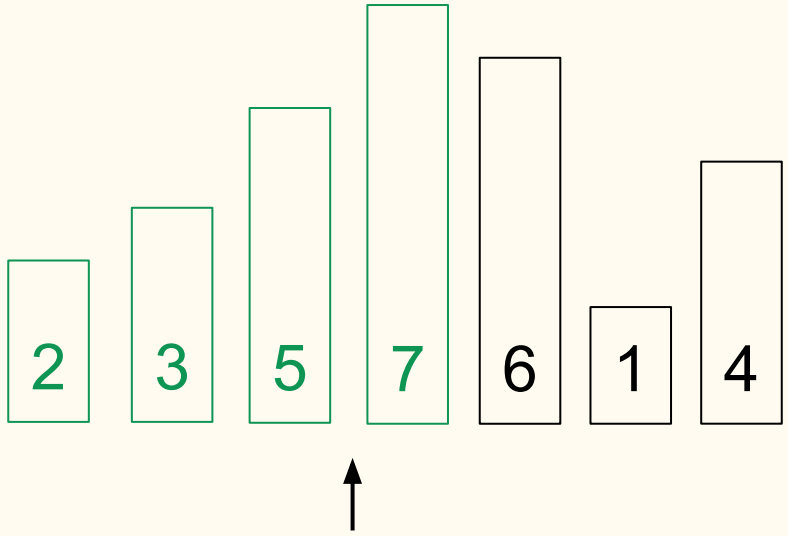
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 5

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

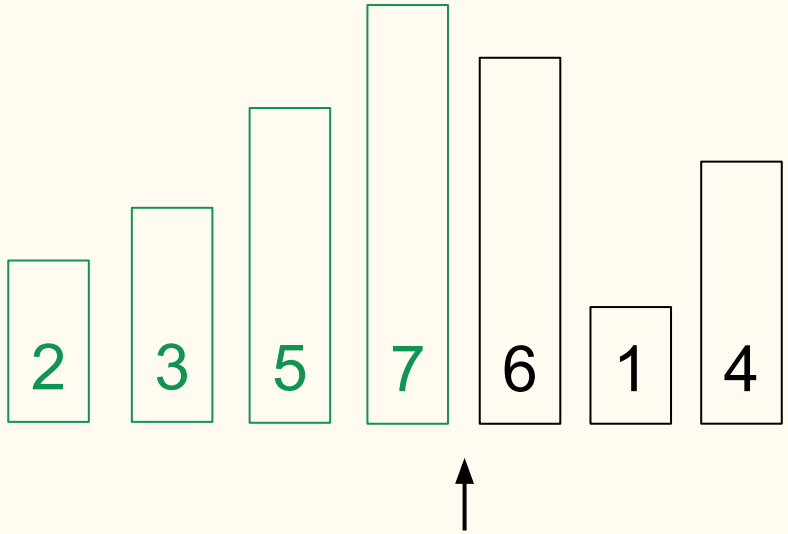
3. Deslocar 1 posição para todos maior que `elemento`

4. **Inserir `elemento`**

`elemento =` 5

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

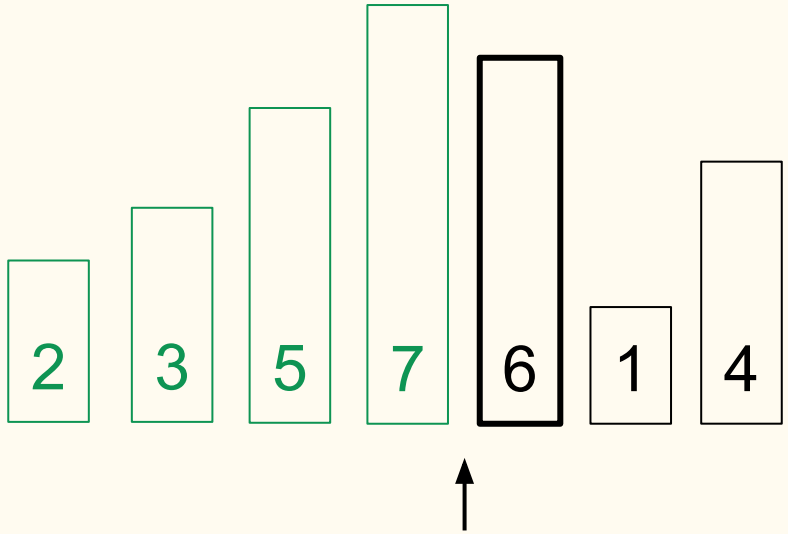
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

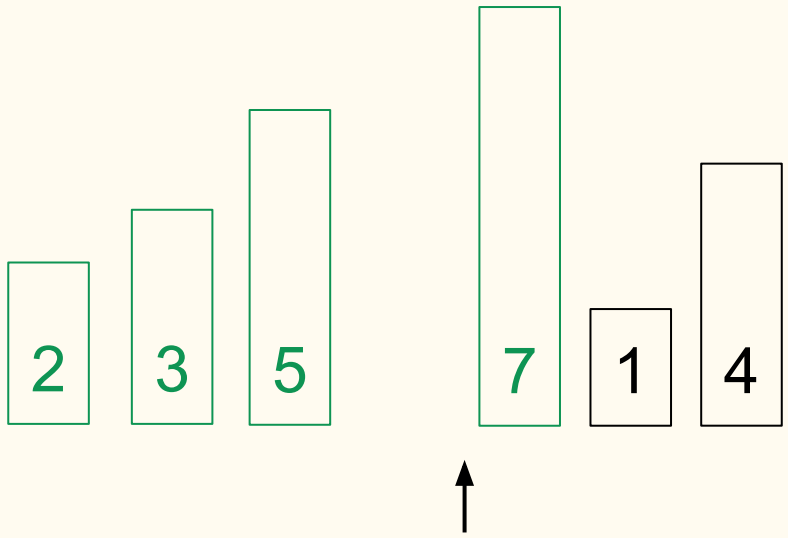
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 6

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

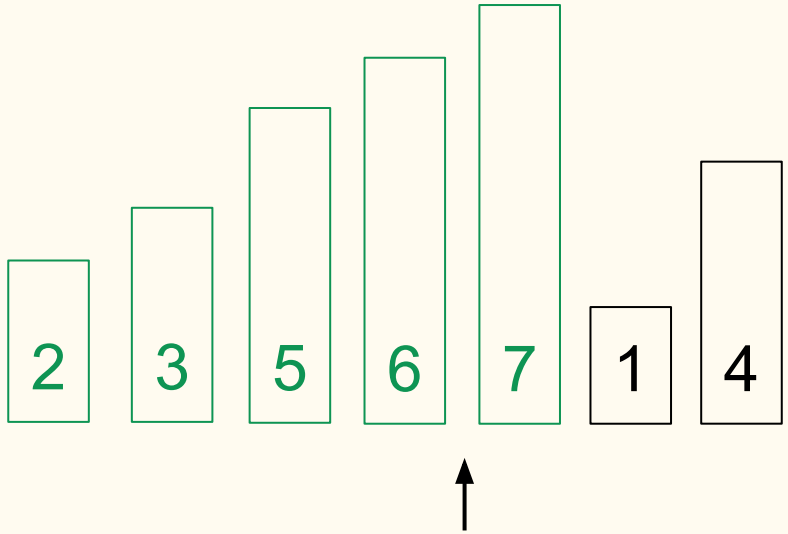
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 6

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

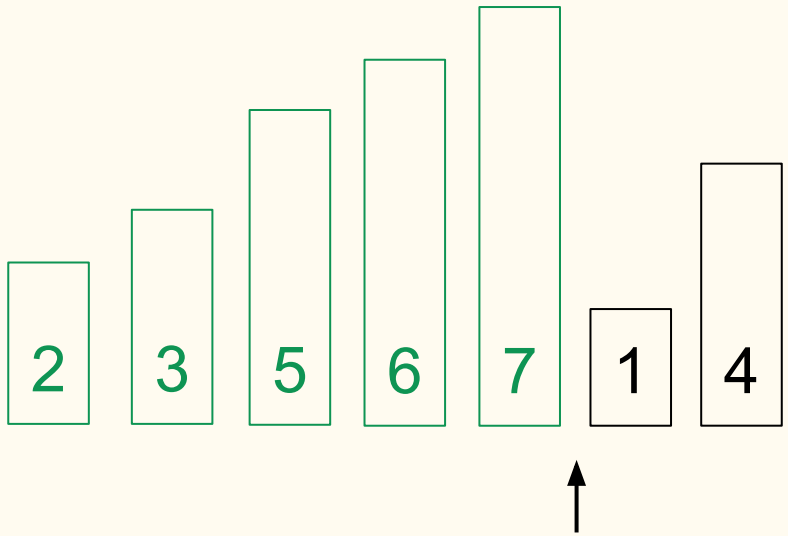
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 6

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

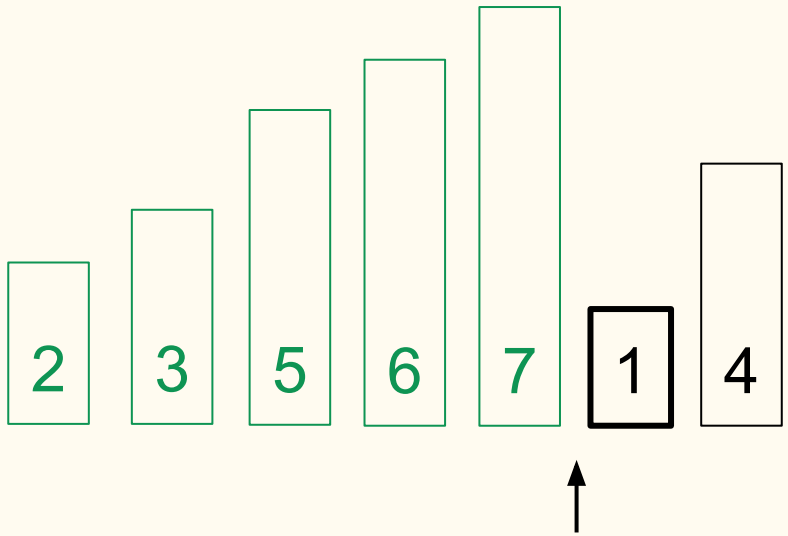
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

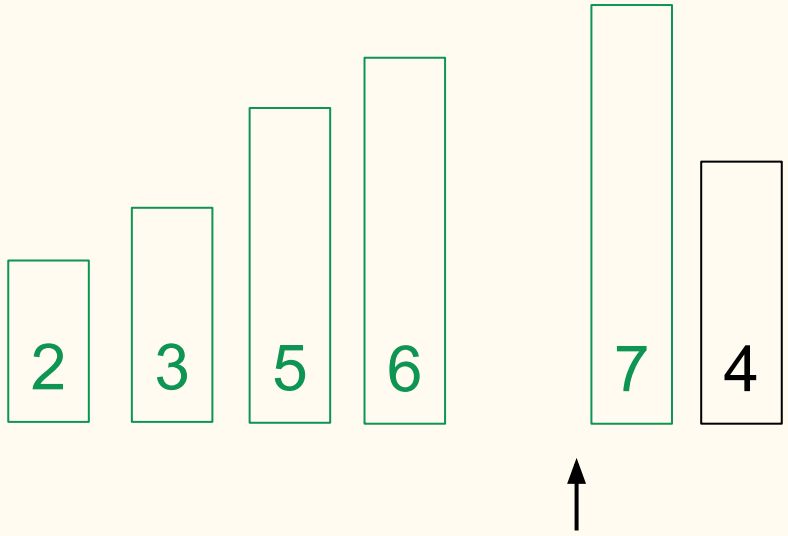
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 1

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

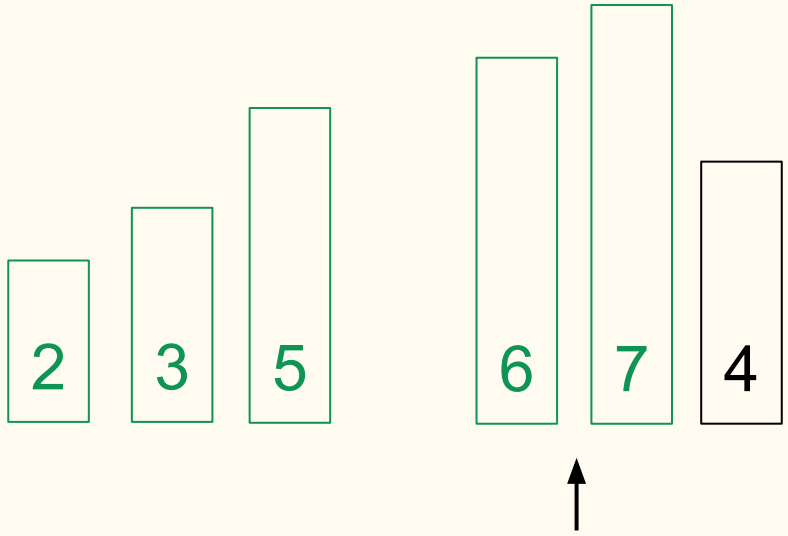
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 1

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

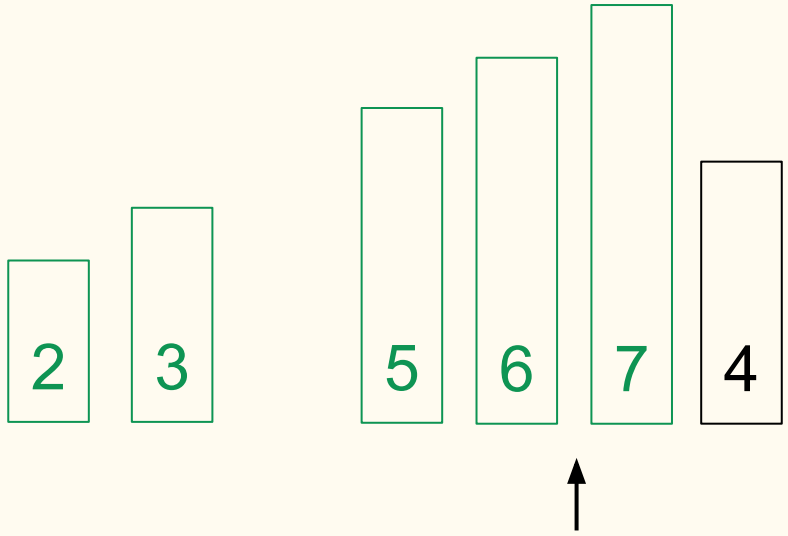
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 1

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

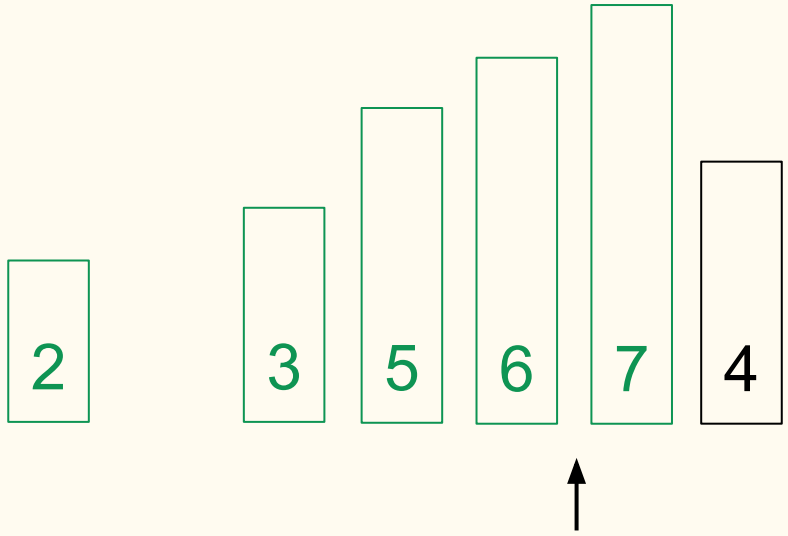
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 1

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

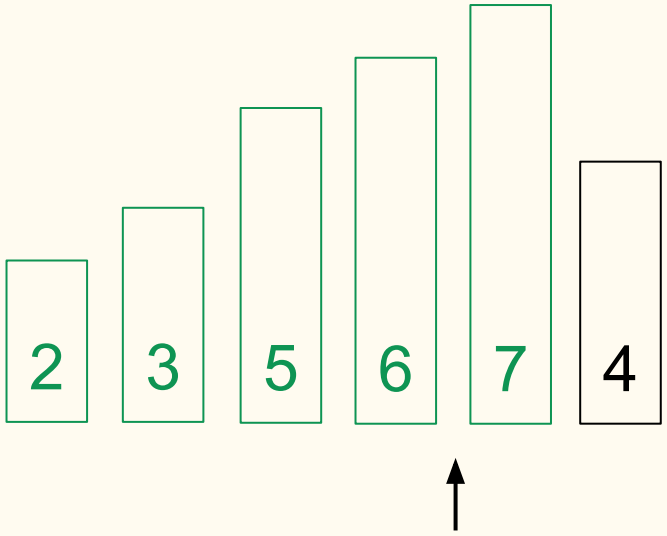
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 1

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

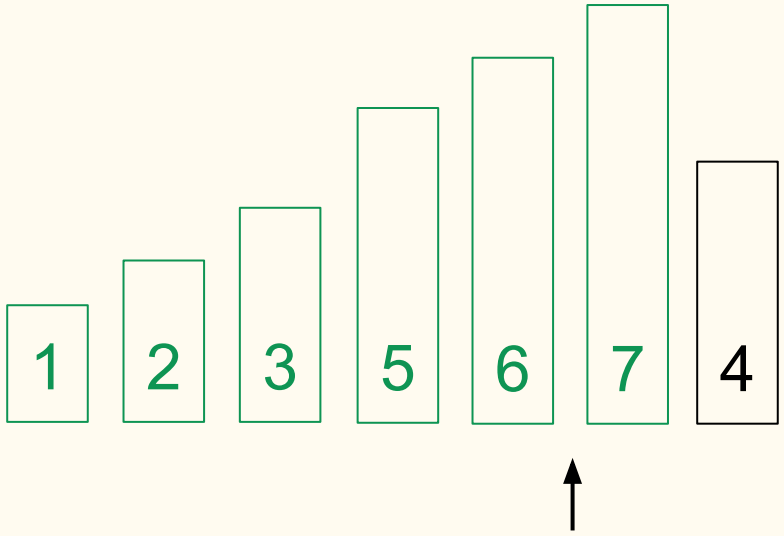
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 1

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

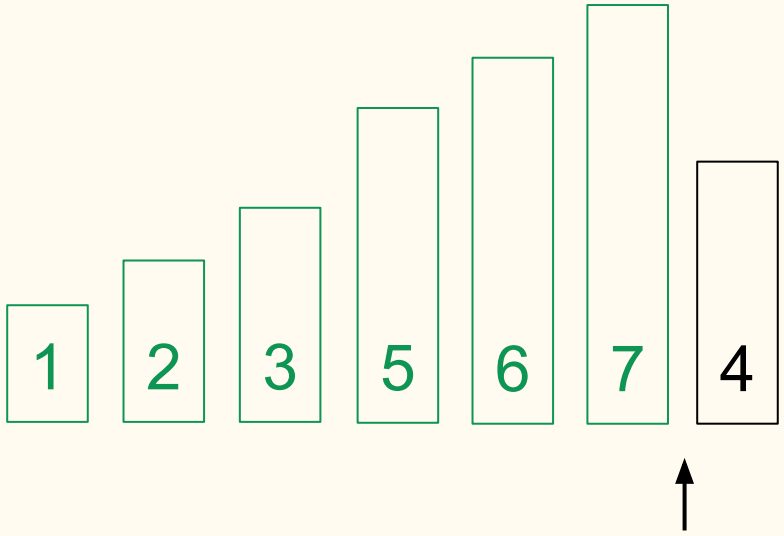
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 1

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

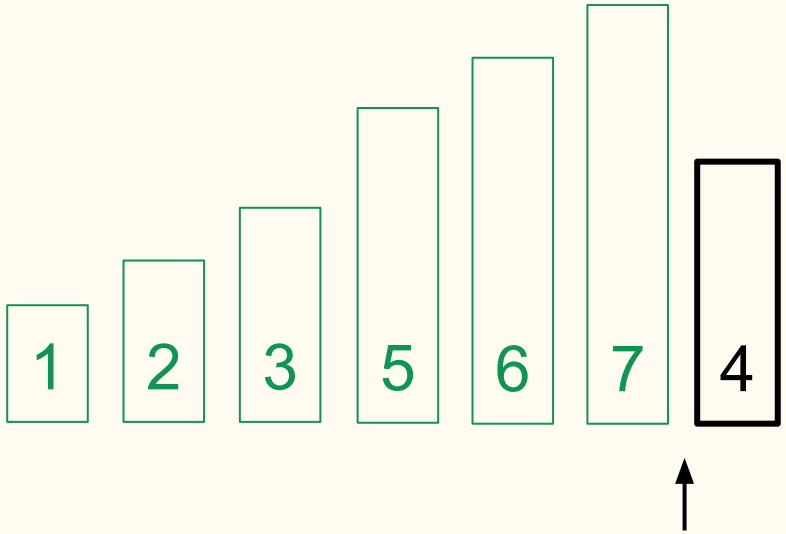
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

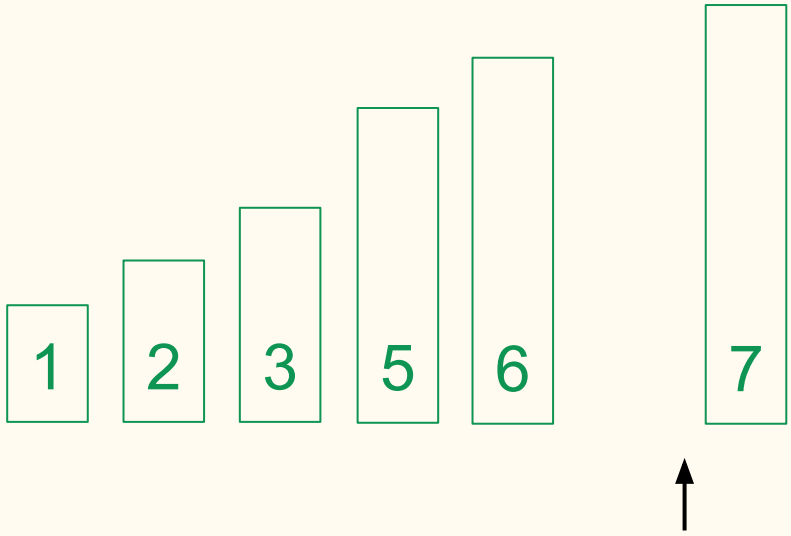
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 4

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Parte ordenada

Parte desordenada

Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

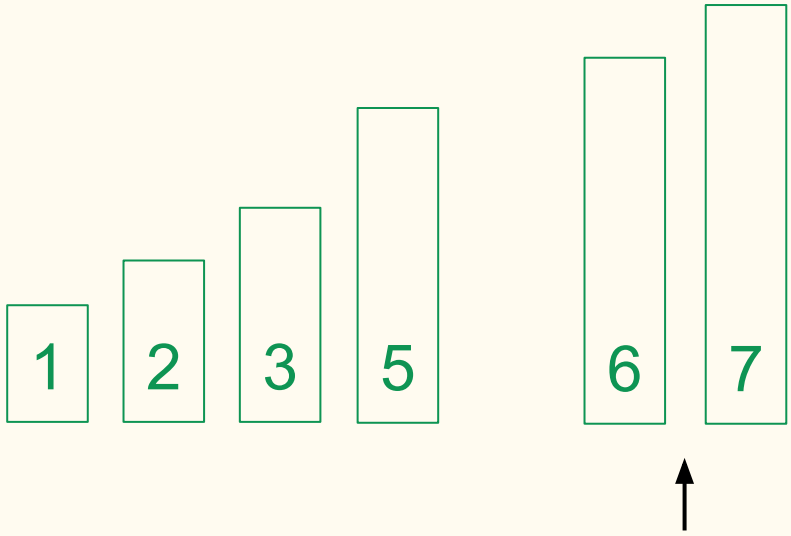
3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =` 4

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Parte ordenada

Parte desordenada

`elemento` = 4

Passo a passo:

1. `elemento`: 1º da parte desordenada

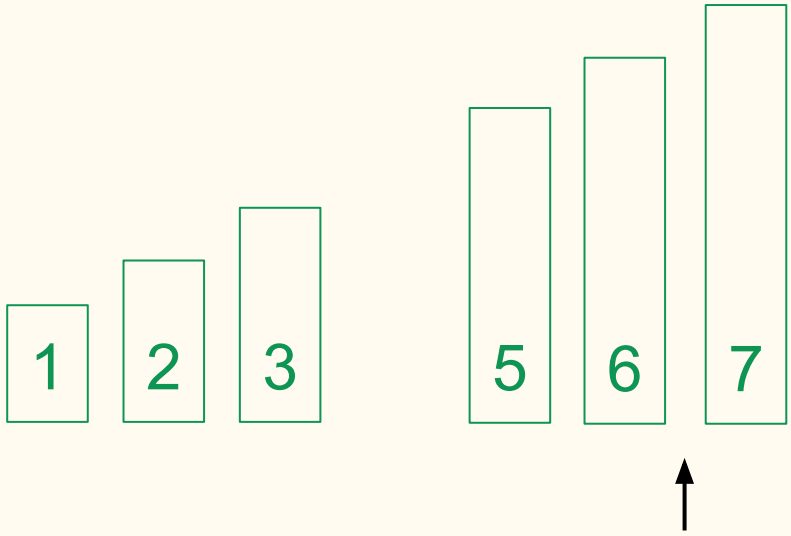
2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

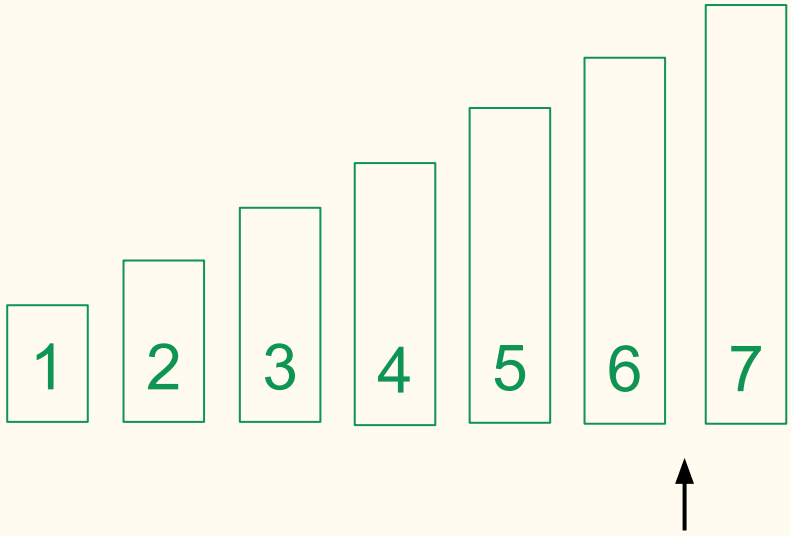
2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Parte ordenada

Parte desordenada

`elemento` = 4

Passo a passo:

1. `elemento`: 1º da parte desordenada

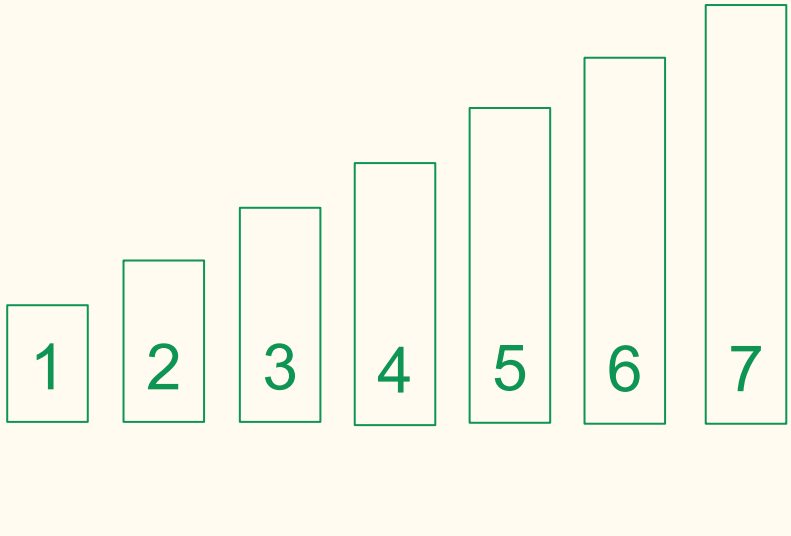
2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Passo a passo no mesmo vetor

- Como executar o mesmo mecanismo no lugar, no mesmo vetor?
  - Marcamos a posição no vetor onde encontra-se a parte ordenada



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

`elemento =`

Ordenação por inserção - Codificando...

Passo a passo:

1. `elemento`: 1º da parte desordenada

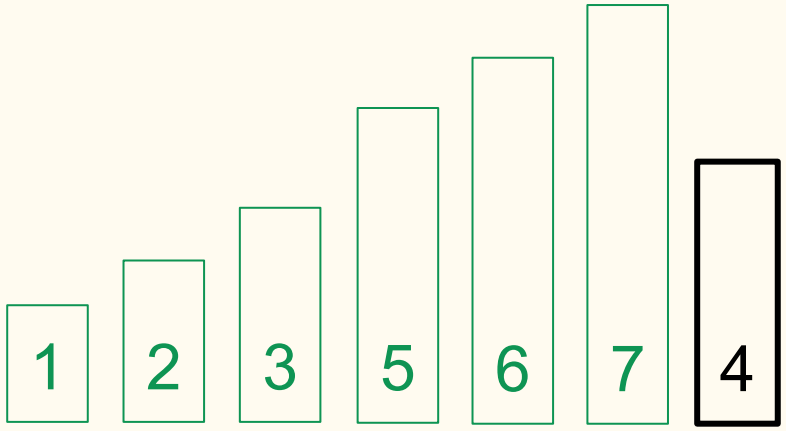
2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Codificando...

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}
```



Passo a passo:

1. `elemento`: 1º da parte desordenada

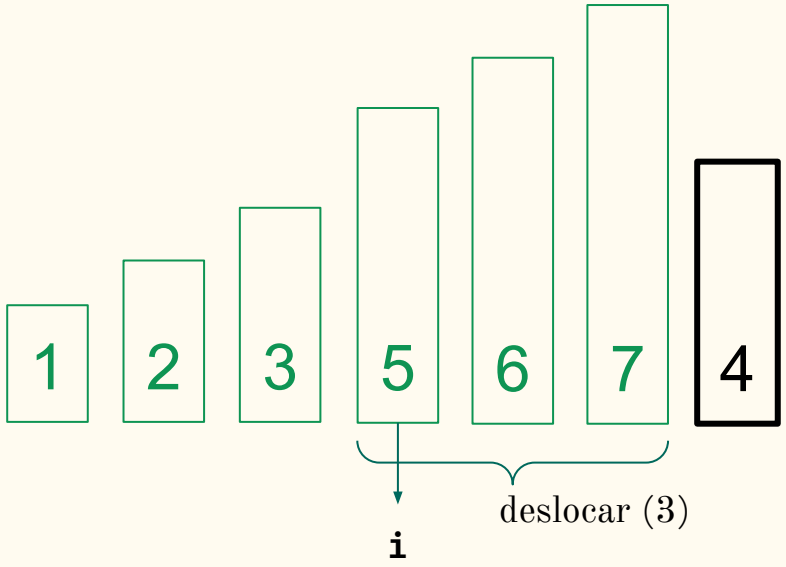
2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Codificando...

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}
```



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Codificando...

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}
```

Passo a passo:

1. `elemento`: 1º da parte desordenada

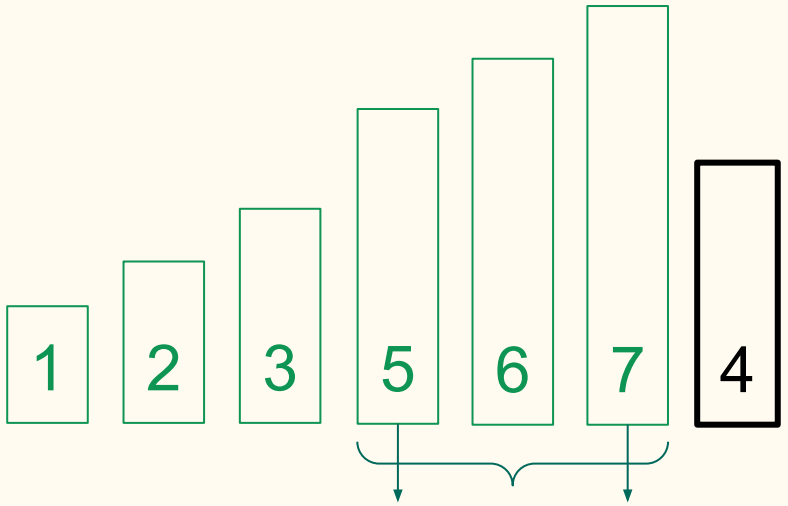
2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Codificando...

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}

 void deslocar_subvetor(int vetor[], int primeiro, int ultimo) {
      int i;
      for (i = ultimo; i >= primeiro; i--)
           vetor[i+1] = vetor[i];
 }
```



Passo a passo:

1. `elemento`: 1º da parte desordenada

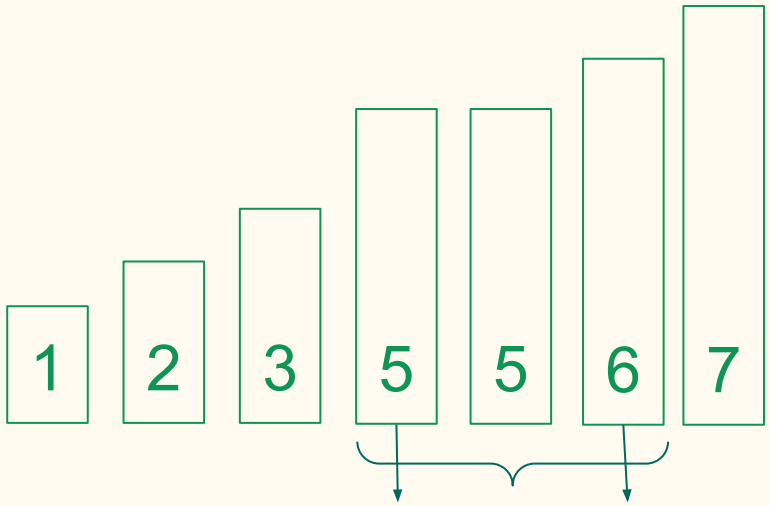
2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Codificando...

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}

 void deslocar_subvetor(int vetor[], int primeiro, int ultimo) {
      int i;
      for (i = ultimo; i >= primeiro; i--)
           vetor[i+1] = vetor[i];
 }
```



Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

Ordenação por inserção - Codificando...

```
int ordenar_insercao(int vetor[], int n) {
     int i, posicao;
     int elemento;
     for (i = 1; i < n; i++) {
          elemento = vetor[i];
          posicao = posicao_elemento(vetor, i-1, elemento);
          deslocar_subvetor(vetor, posicao, i-1);
          vetor[posicao] = elemento;
     }
}
```

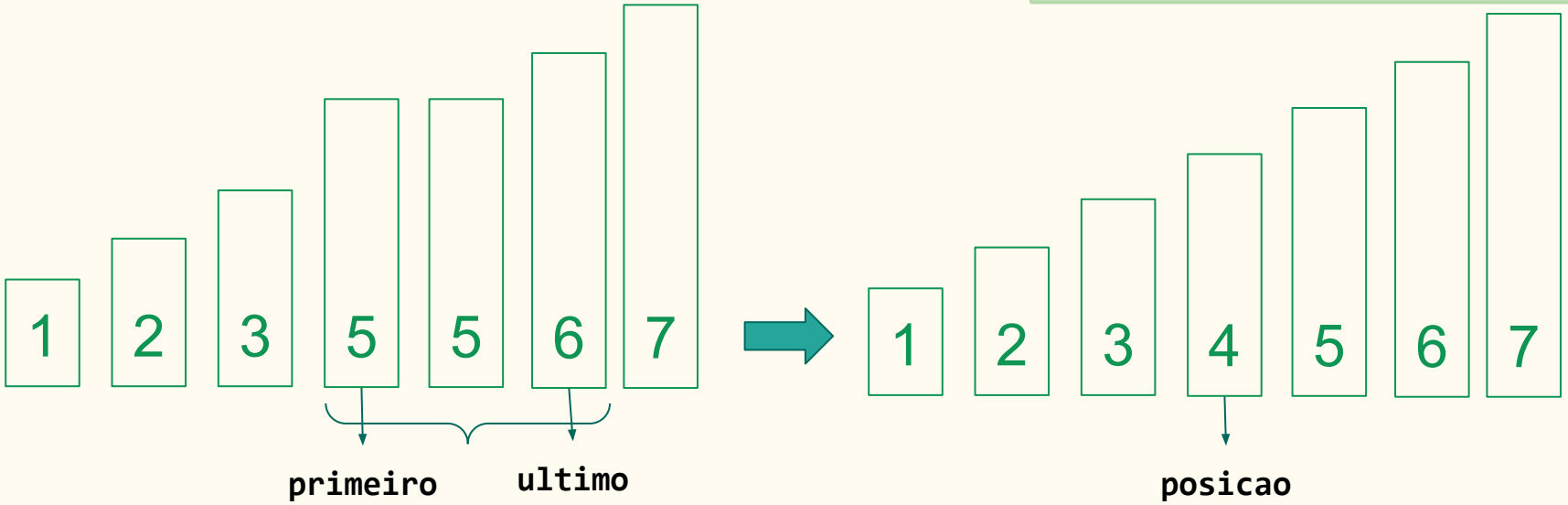
Passo a passo:

1. `elemento`: 1º da parte desordenada

2. Encontrar sua posição na lista ordenada

3. Deslocar 1 posição para todos maior que `elemento`

4. Inserir `elemento`

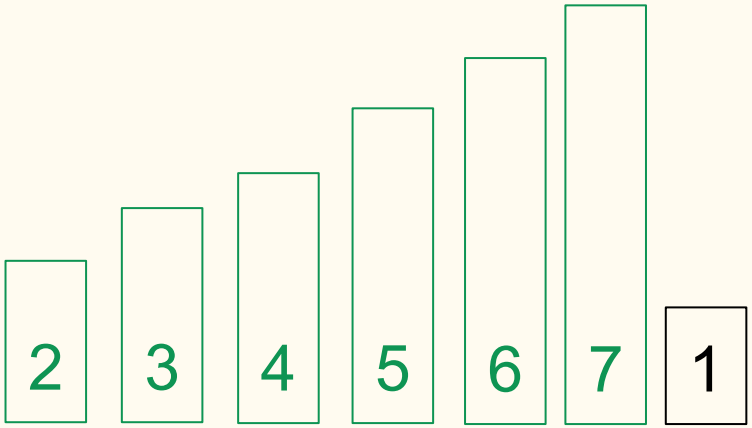


Ordenação por inserção - Complexidade

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}

 void deslocar_subvetor(int vetor[], int primeiro, int ultimo) {
      int i;
      for (i = ultimo; i >= primeiro; i--)
           vetor[i+1] = vetor[i];
 }
```

Ordenação por inserção - Complexidade

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}

 void deslocar_subvetor(int vetor[], int primeiro, int ultimo) {
      int i;
      for (i = ultimo; i >= primeiro; i--)
           vetor[i+1] = vetor[i];
 }
```

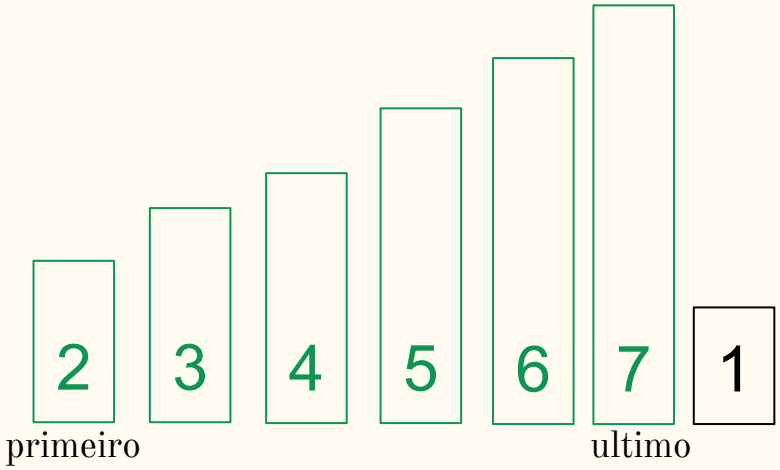


Depende da entrada

i=0

Ordenação por inserção - Complexidade

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}

 void deslocar_subvetor(int vetor[], int primeiro, int ultimo) {
      int i;
      for (i = ultimo; i >= primeiro; i--)
           vetor[i+1] = vetor[i];
 }
```



Depende da entrada

i=0

Elemento é o menor da lista

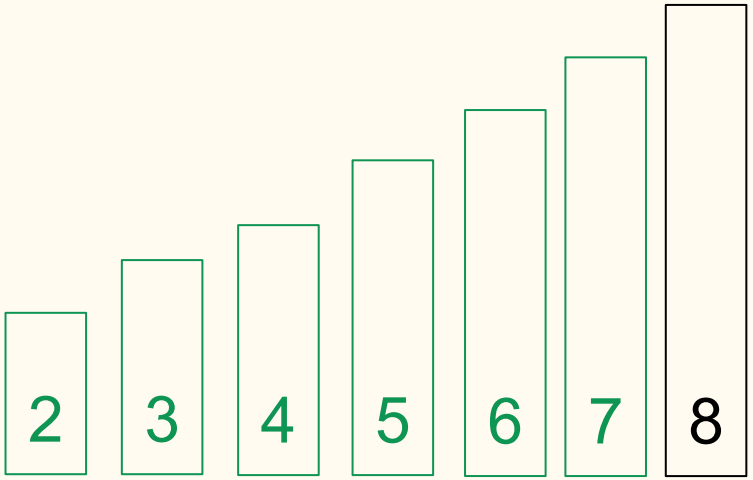
Toda a lista precisa ser deslocada

Comparações: 1

Trocas: ultimo - primeiro

Ordenação por inserção - Complexidade

```
int posicao_elemento(int vetor[], int ultimo, int elemento) {
     int i;
     for (i = 0; i <= ultimo && vetor[i] <= elemento; i++);
     return i;
}

 void deslocar_subvetor(int vetor[], int primeiro, int ultimo) {
      int i;
      for (i = ultimo; i >= primeiro; i--)
           vetor[i+1] = vetor[i];
 }
```



Depende da entrada

i=0

Elemento é o menor da lista

Toda a lista precisa ser deslocada

Comparações: 1

Trocas: ultimo - primeiro

i=ultimo+1

Elemento é o maior da lista

Nenhum deslocamento

Comparações: ultimo

Trocas: 0

(nao satisfaz condição no for)

Ordenação por inserção - Complexidade

```
int ordenar_insercao(int vetor[], int n) {
     int i, posicao;
     int elemento;
     for (i = 1; i < n; i++) {
          elemento = vetor[i];
          posicao = posicao_elemento(vetor, i-1, elemento);
          deslocar_subvetor(vetor, posicao, i-1);
          vetor[posicao] = elemento;
     }
}
```

Ordenado:

i=1
- comparação 1 (posicao=1)
- deslocamento 0

i=2
- comparação 2 (posicao=2)
- deslocamento 0

i=3
- comparação 3 (posicao=3)
- deslocamento 0

.....

i=n-1
- comparações n-1
- deslocamento 0

Ordenação por inserção - Complexidade

```
int ordenar_insercao(int vetor[], int n) {
     int i, posicao;
     int elemento;
     for (i = 1; i < n; i++) {
          elemento = vetor[i];
          posicao = posicao_elemento(vetor, i-1, elemento);
          deslocar_subvetor(vetor, posicao, i-1);
          vetor[posicao] = elemento;
     }
}
```

Ordenado:

i=1
- comparação 1 (posicao=1)
- deslocamento 0

i=2
- comparação 2 (posicao=2)
- deslocamento 0

i=3
- comparação 3 (posicao=3)
- deslocamento 0

.....

i=n-1
- comparações n-1
- deslocamento 0

Comparações:$O(n^2)$
Trocas/deslocamentos: $O(1)$

Ordenação por inserção - Complexidade

```
int ordenar_insercao(int vetor[], int n) {
     int i, posicao;
     int elemento;
     for (i = 1; i < n; i++) {
          elemento = vetor[i];
          posicao = posicao_elemento(vetor, i-1, elemento);
          deslocar_subvetor(vetor, posicao, i-1);
          vetor[posicao] = elemento;
     }
}
```

Ordenado:

i=1

comparação 1 (posicao=1)

atribuições 0 +2

i=2

comparação 2 (posicao=2)

atribuições 0 +2

i=3

comparação 3 (posicao=3)

atribuições 0 +2

.....

i=n-1

comparações n-1

atribuições 0+2

Comparações:$O(n^2)$

Trocas/atribuições: $2(n-1)=O(n)$

Ordenação por inserção - Complexidade

```
int ordenar_insercao(int vetor[], int n) {
     int i, posicao;
     int elemento;
     for (i = 1; i < n; i++) {
          elemento = vetor[i];
          posicao = posicao_elemento(vetor, i-1, elemento);
          deslocar_subvetor(vetor, posicao, i-1);
          vetor[posicao] = elemento;
     }
}
```

Ordenado Decrescente:

i=1
- comparação 1 (posicao=0)
- deslocamento 1

i=2
- comparação 1 (posicao=0)
- deslocamento 2

i=3
- comparação 1 (posicao=0)
- deslocamento 3

.....

i=n-1
- comparações 1
- deslocamento n-1

Comparações: n-1 = O(n)
Trocas/atribuições: $O(n^2)$

Ordenação por inserção - Complexidade

```
int ordenar_insercao(int vetor[], int n) {
     int i, posicao;
     int elemento;
     for (i = 1; i < n; i++) {
          elemento = vetor[i];
          posicao = posicao_elemento(vetor, i-1, elemento);
          deslocar_subvetor(vetor, posicao, i-1);
          vetor[posicao] = elemento;
     }
}
```

Ordenado Decrescente:

i=1
- comparação 1 (posicao=0)
- deslocamento 1

i=2
- comparação 1 (posicao=0)
- deslocamento 2

i=3
- comparação 1 (posicao=0)
- deslocamento 3

.....

i=n-1
- comparações 1
- deslocamento n-1

Comparações: $n-1 = O(n)$

Trocas/atribuições:

$O(n^2)+2(n-1) = O(n^2)$

Ordenação por inserção - Complexidade

- Coeficiente quadrático menor que demais apresentados aqui
  - Faz substituição no deslocamento (1 operação), não troca (3 operações)
  - Veja diferença entre n trocas x deslocamentos

  - Operações de deslocamento: $1 + n + 1 = n + 2$

```
elemento = vetor[i]
v[i+1]=v[i] (x n)
vetor[posicao]=elemento
```

  - Operações de trocas: $3n$

```
Void troca(int *a, int *b) {
      int aux = *a;
      *a = *b;
      *b = aux;
}
```

Ordenação por inserção - Complexidade

Desafio: Como melhorar esse algoritmo mantendo o mesmo invariante?

-Invariante: A cada iteração i, v[0 - (i-1)] está inicialmente ordenado, insere i na posição correta; ao final v[0 - i] está ordenado.

Dica: a função de busca pela posição onde i deve ser inserida pode ser melhorada

Roteiro

Introdução

Métodos iterativos

- Ordenação por trocas
- Método das bolhas
- Ordenação por inserção

Métodos recursivos - Divisão e Conquista

- Merge Sort
- Quick Sort

Complexidade algorítmica dos algoritmos de classificação

Complexidade para os métodos iterativos (vistos até aqui)

	Melhor caso		Pior caso	
	Trocas	Comparações	Trocas	Comparações
Trocas		$O(n^2)$		$O(n^2)$
Bolhas		$O(n^2)$		$O(n^2)$
Inserção				

Complexidade para os métodos iterativos (vistos até aqui)

	Melhor caso		Pior caso	
	Trocas	Comparações	Trocas	Comparações
Trocas	O(1)	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bolhas	O(1)	$O(n^2)$	$O(n^2)$	$O(n^2)$
Inserção				

Complexidade para os métodos iterativos (vistos até aqui)

	Melhor caso		Pior caso	
	Trocas	Comparações	Trocas	Comparações
Trocas	O(1)	$O(n^2)$	$O(n^2)$	$O(n^2)$
Bolhas	O(1)	$O(n^2)$	$O(n^2)$	$O(n^2)$
Inserção	O(n)	$O(n^2)$	$O(n^2)$	O(n)

Roteiro

Introdução

Métodos iterativos
- Ordenação por trocas
- Método das bolhas
- Ordenação por inserção

Métodos recursivos - Divisão e Conquista
- Merge Sort
- Quick Sort

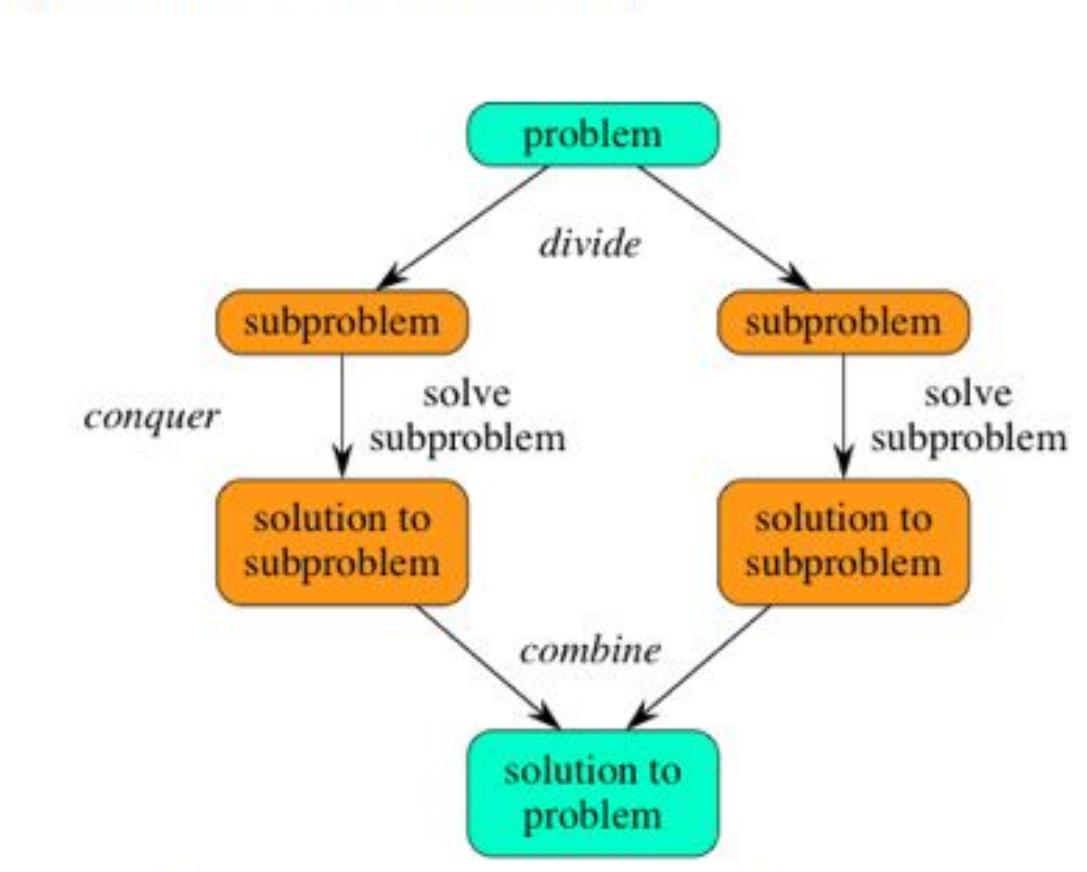
Complexidade algorítmica dos algoritmos de classificação

Divisão e Conquista

- A estratégia parte do princípio que é mais fácil resolver problemas menores

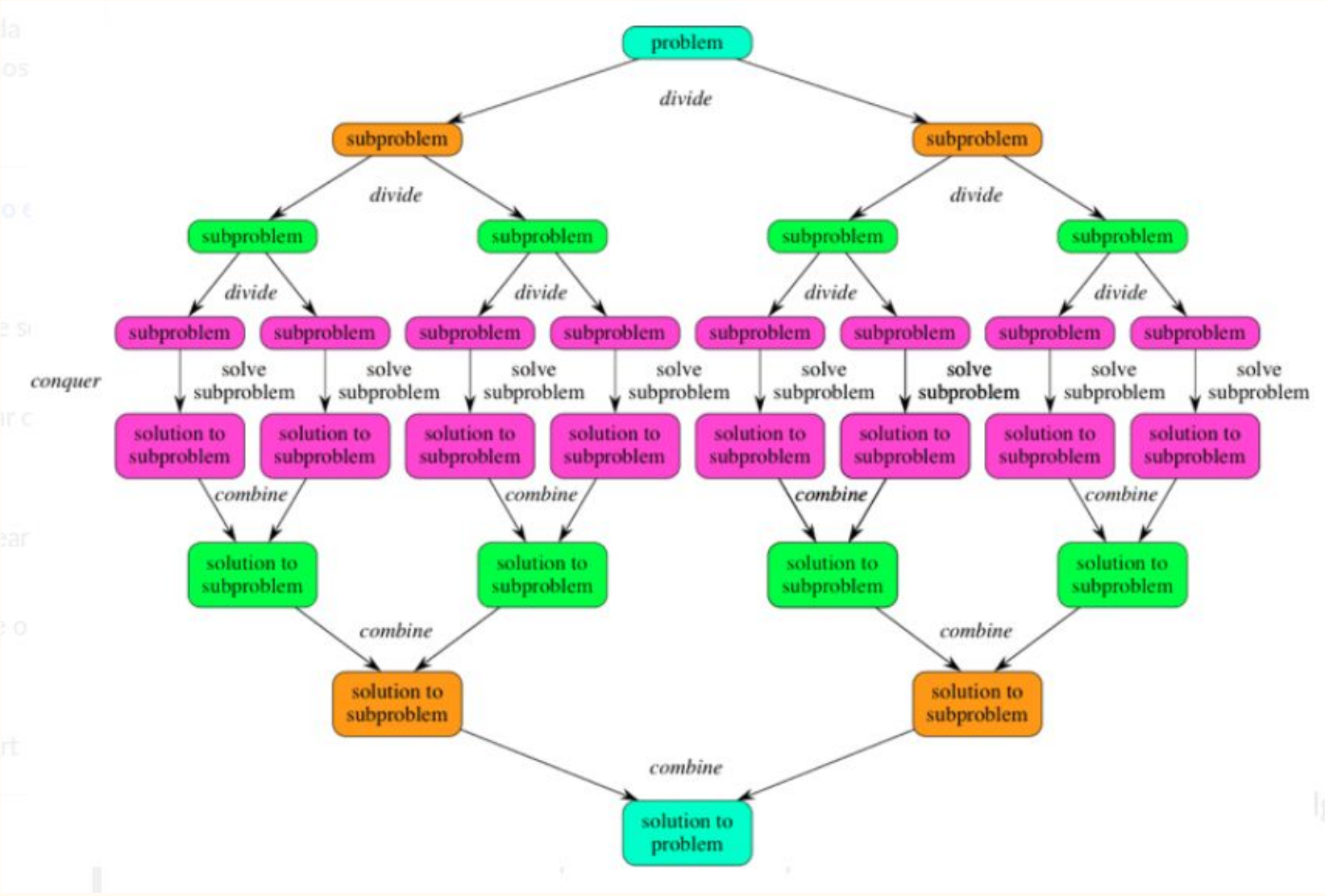
- Com isso podemos resolver o problema em duas etapas:
  - **Divisão:** quebramos um problema em vários subproblemas menores
  - **Conquista:** Combinamos a solução dos problemas menores

Divisão e Conquista



https://pt.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms

Divisão e Conquista



Roteiro

Introdução

Métodos iterativos

- Ordenação por trocas
- Método das bolhas
- Ordenação por inserção

Métodos recursivos - Divisão e Conquista

- **Merge Sort**
- Quick Sort

Complexidade algorítmica dos algoritmos de classificação

Merge Sort - Ideia principal

- Recebemos um vetor de tamanho n

1. Dividimos em dois subvetores com tamanho n/2
   - vetor de início a meio
   - vetor de meio a fim

2. Ordenamos os vetores menores
   usando o mesmo método, dividindo em problema menor
3. Combinamos as soluções dos 2 vetores menor
   - função chamada **intercalar**

4. O caso base é vetor de tamanho 0 ou 1.
   - Isto é, já está ordenado trivialmente.

Merge Sort - Estruturando a ideia

```
void merge-sort(int vetor[], int ini, int fim) {
        int meio;
        if (ini < fim) {
        1  meio = (ini + fim) / 2;
           merge-sort(vetor, ini, meio);
           merge-sort(vetor, meio + 1, fim);
           intercalar(vetor, ini, meio, fim);
        }
}
```

Merge Sort - Estruturando a ideia

```
void merge-sort(int vetor[], int ini, int fim) {
        int meio;
        if (ini < fim) {
1       meio = (ini + fim) / 2;
2       merge-sort(vetor, ini, meio);
        merge-sort(vetor, meio + 1, fim);
           intercalar(vetor, ini, meio, fim);
        }
}
```

Merge Sort - Estruturando a ideia

```
void merge-sort(int vetor[], int ini, int fim) {
        int meio;
        if (ini < fim) {
        1 meio = (ini + fim) / 2;
          merge-sort(vetor, ini, meio);
        2
          merge-sort(vetor, meio + 1, fim);
        3 intercalar(vetor, ini, meio, fim);
        }
}
```

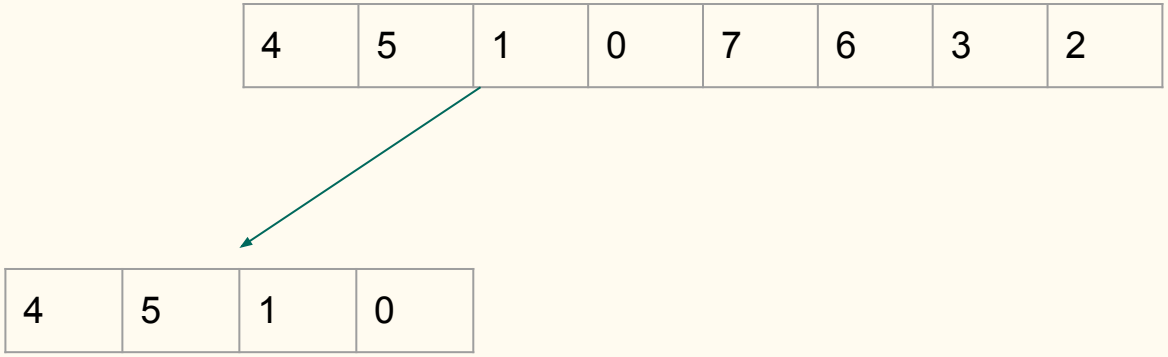
Merge Sort - Estruturando a ideia

```
void merge-sort(int vetor[], int ini, int fim) {
        int meio;
    4   if (ini < fim) {
    1     meio = (ini + fim) / 2;
    2     merge-sort(vetor, ini, meio);
          merge-sort(vetor, meio + 1, fim);
    3     intercalar(vetor, ini, meio, fim);
        }
}
```

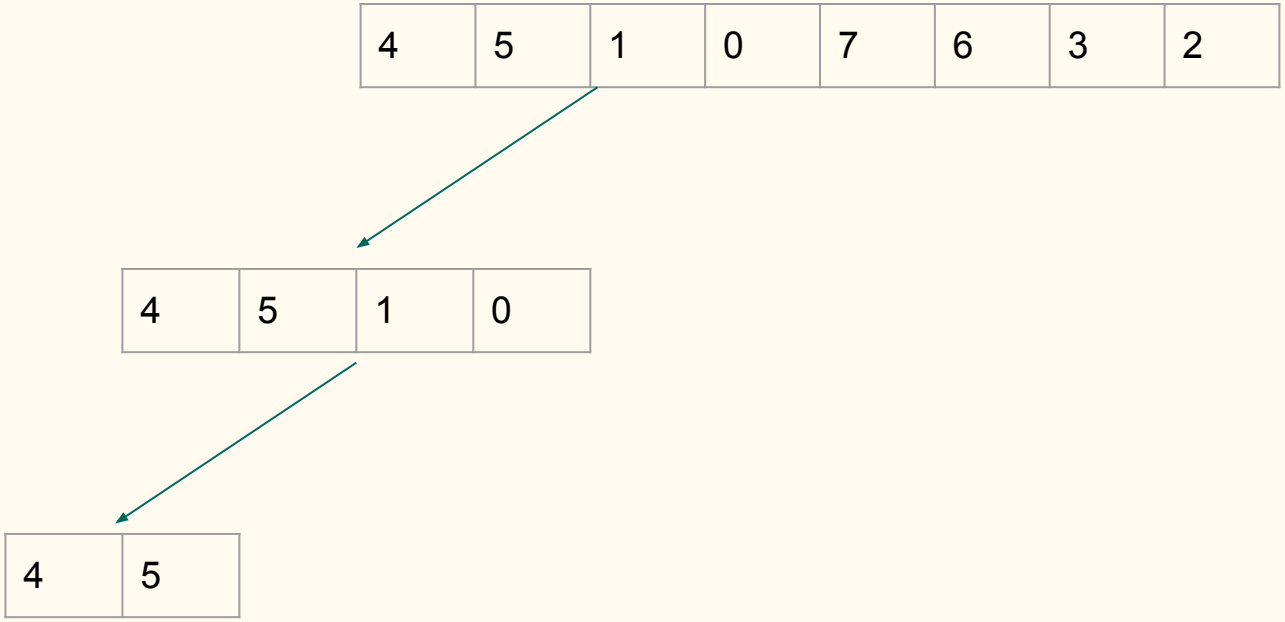
Merge Sort - Passo a passo

4	5	1	0	7	6	3	2

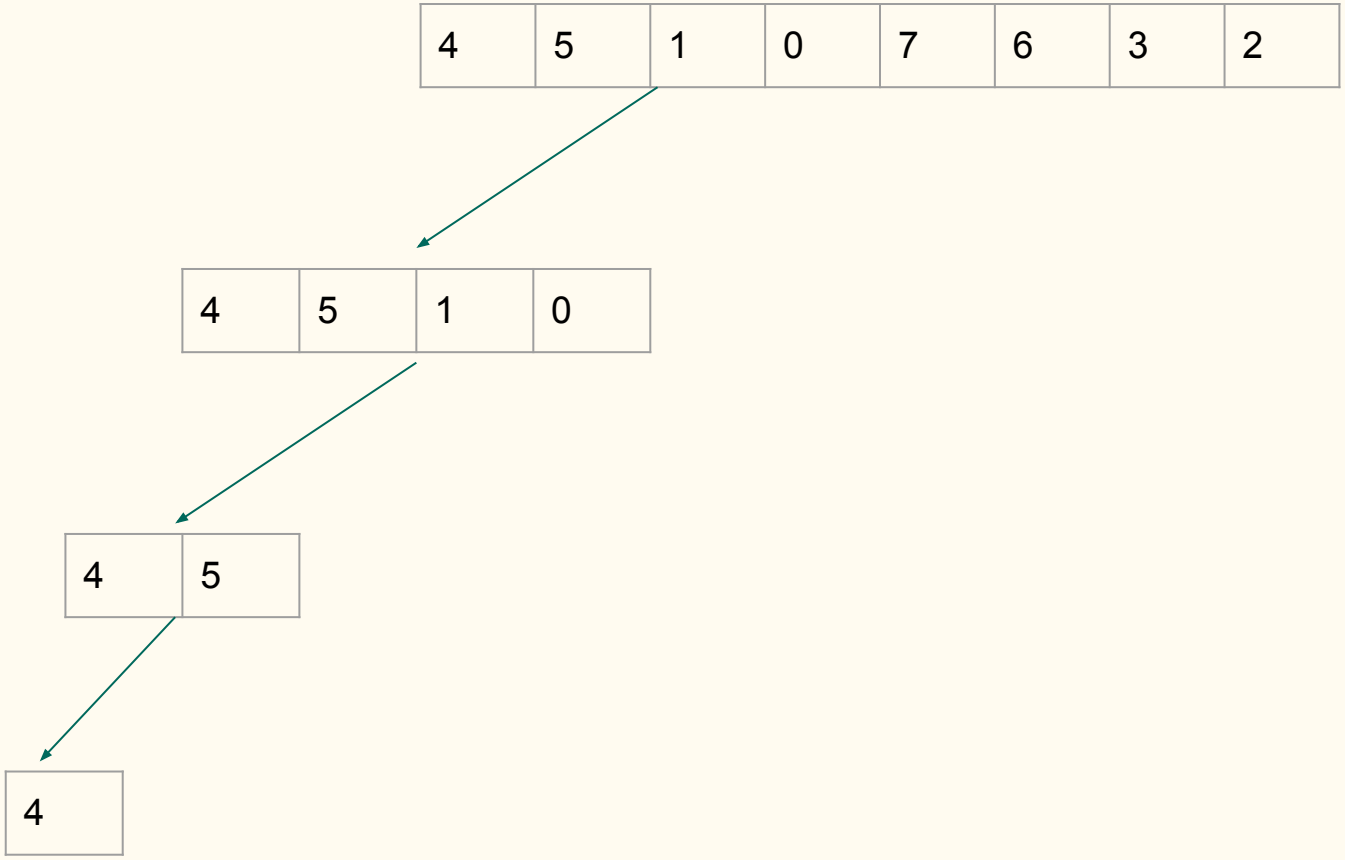
Merge Sort - Passo a passo



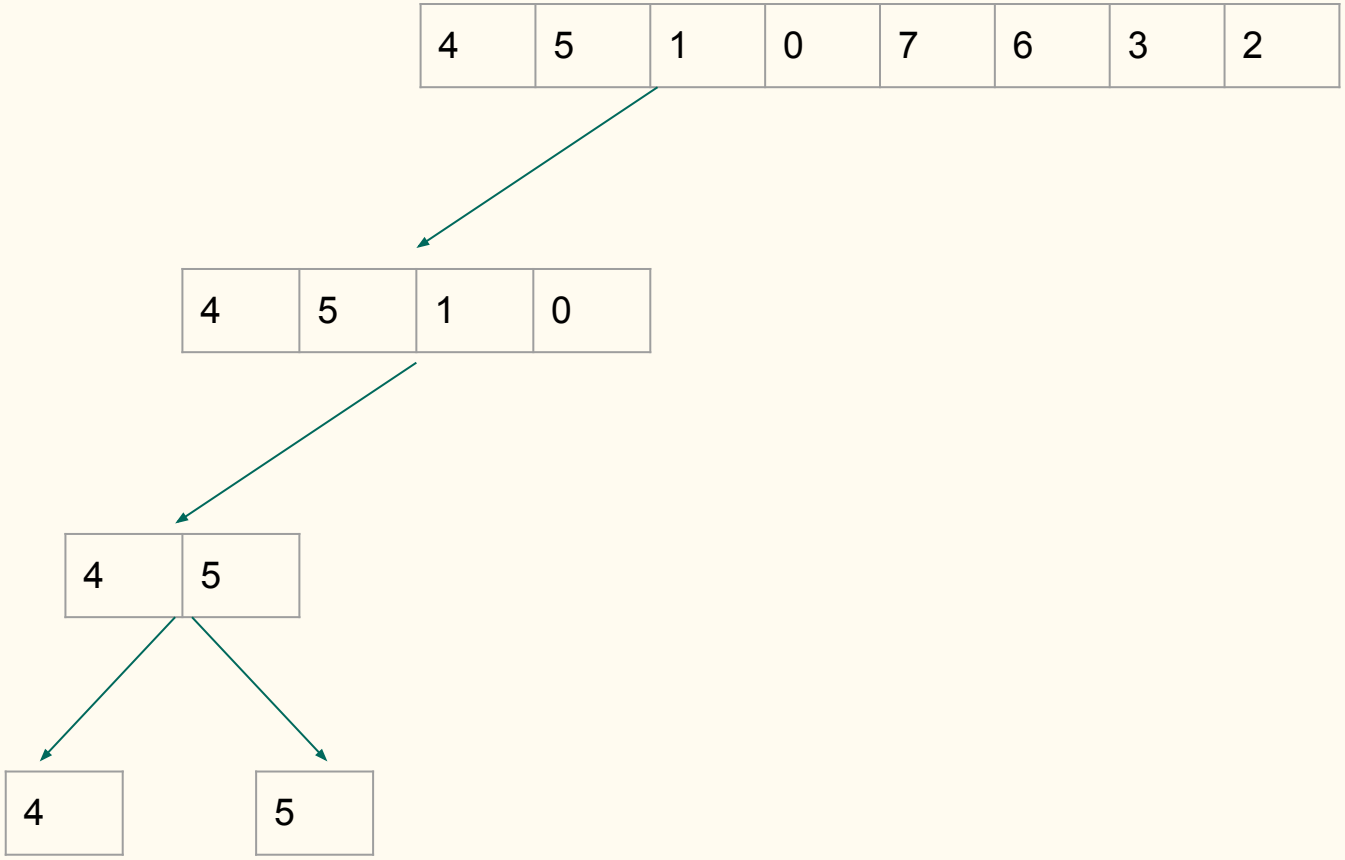
Merge Sort - Passo a passo



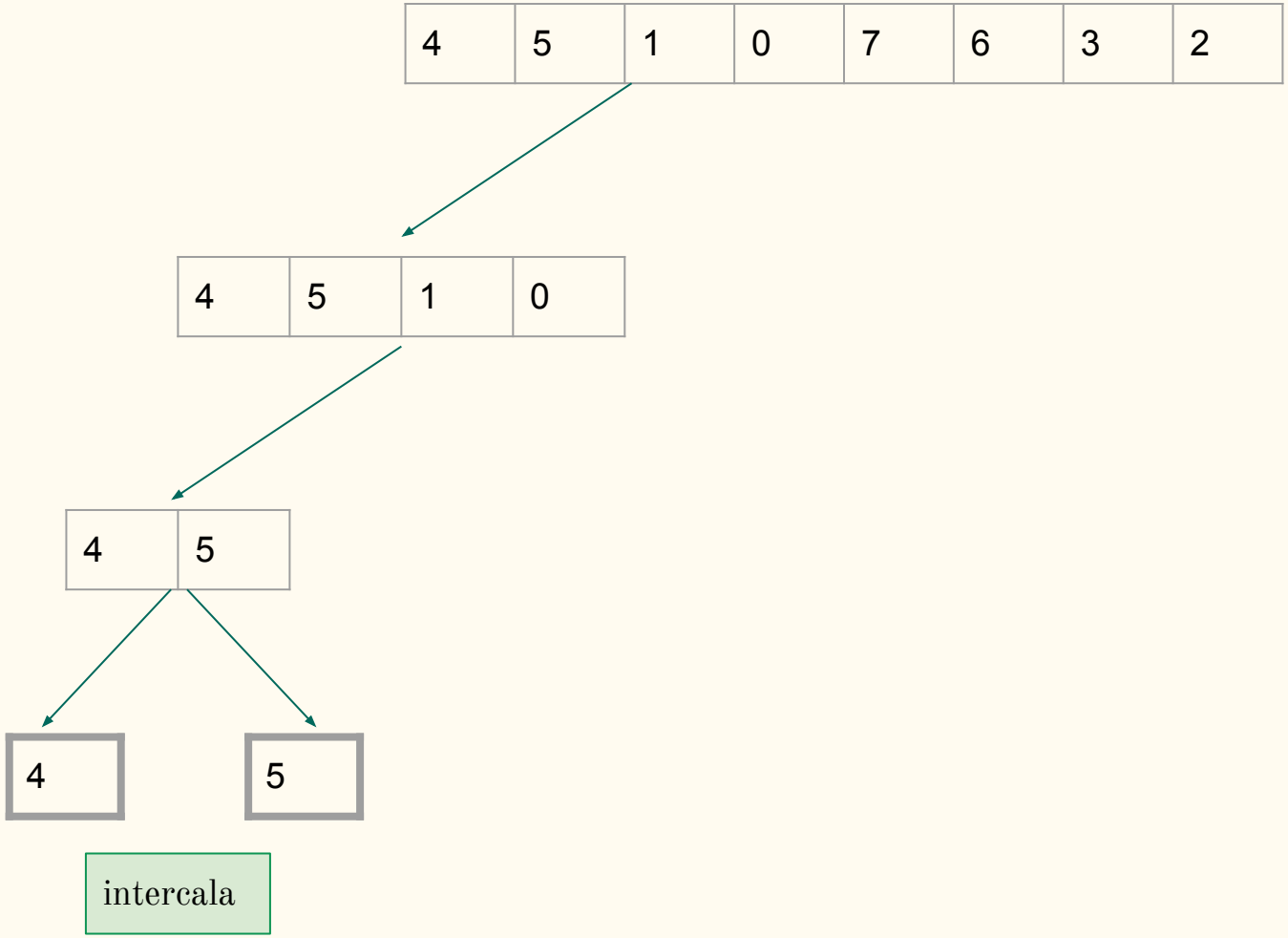
Merge Sort - Passo a passo



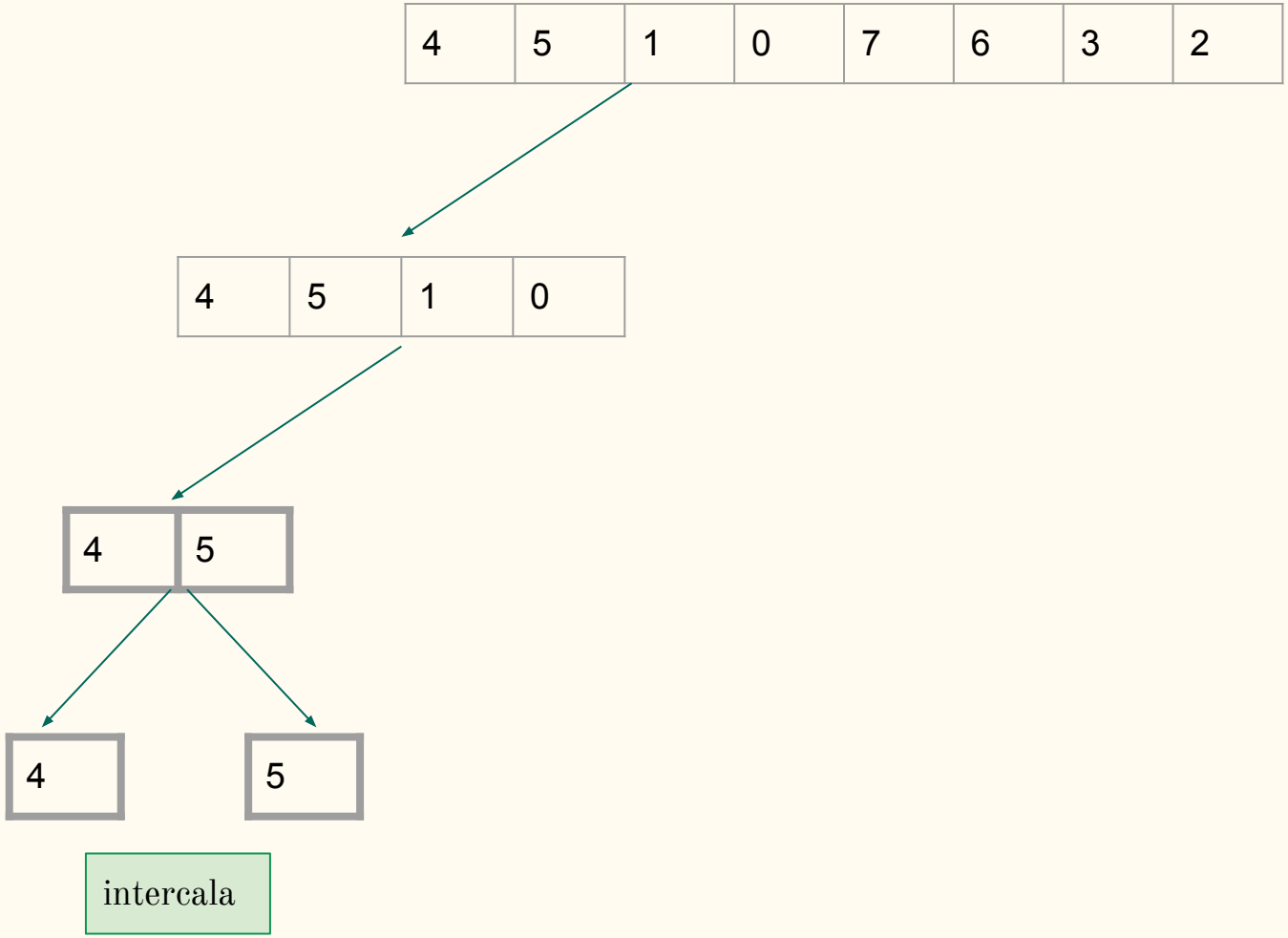
Merge Sort - Passo a passo



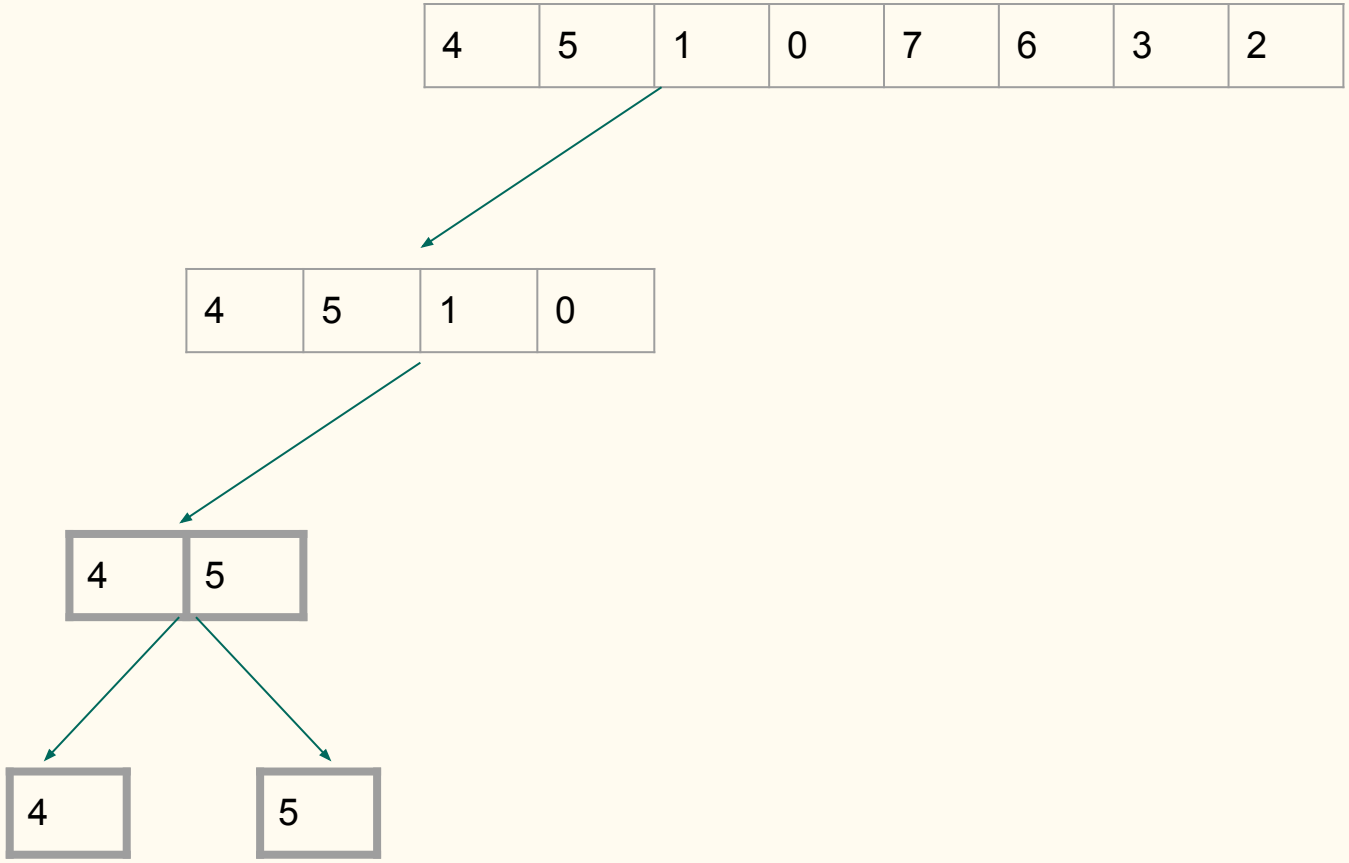
Merge Sort - Passo a passo



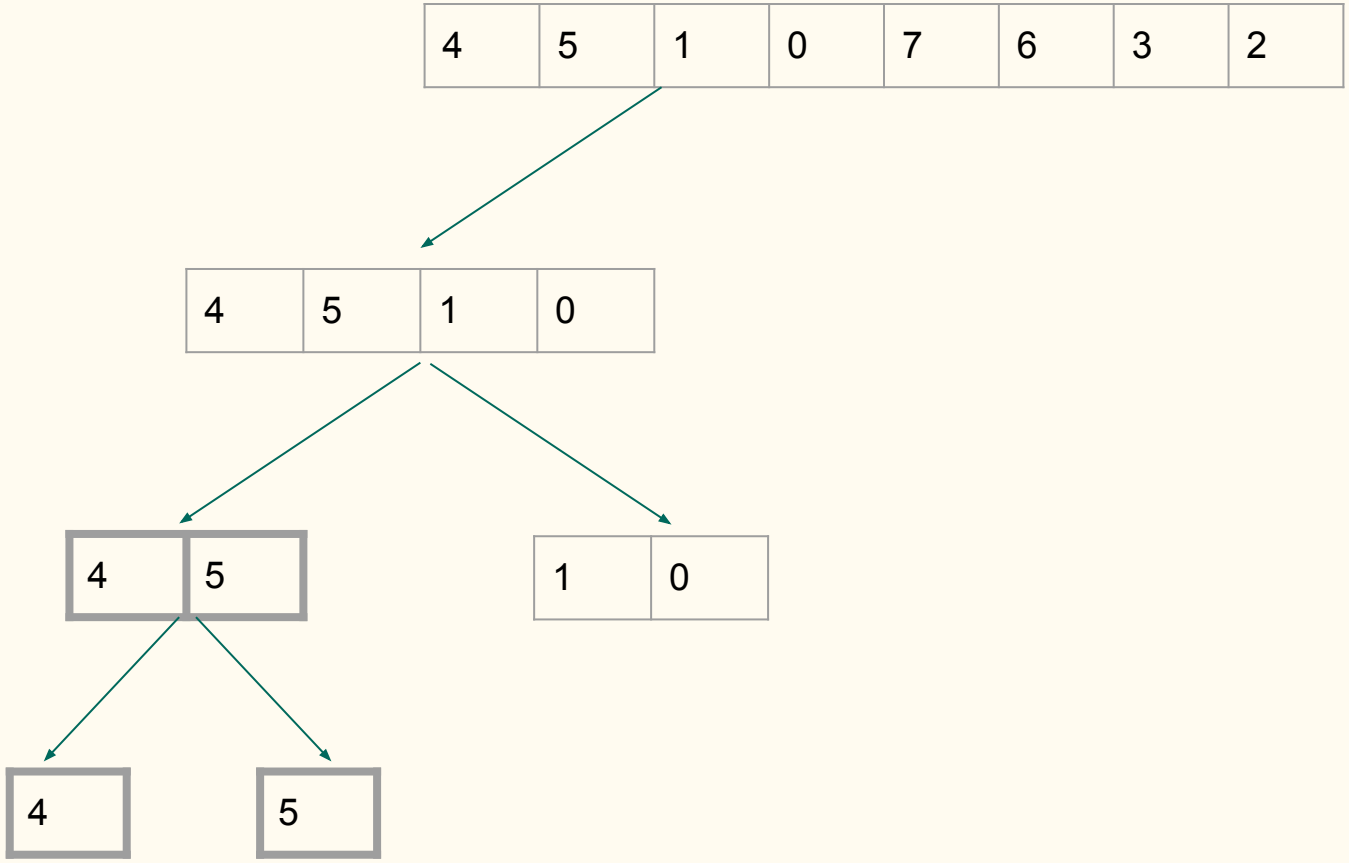
Merge Sort - Passo a passo



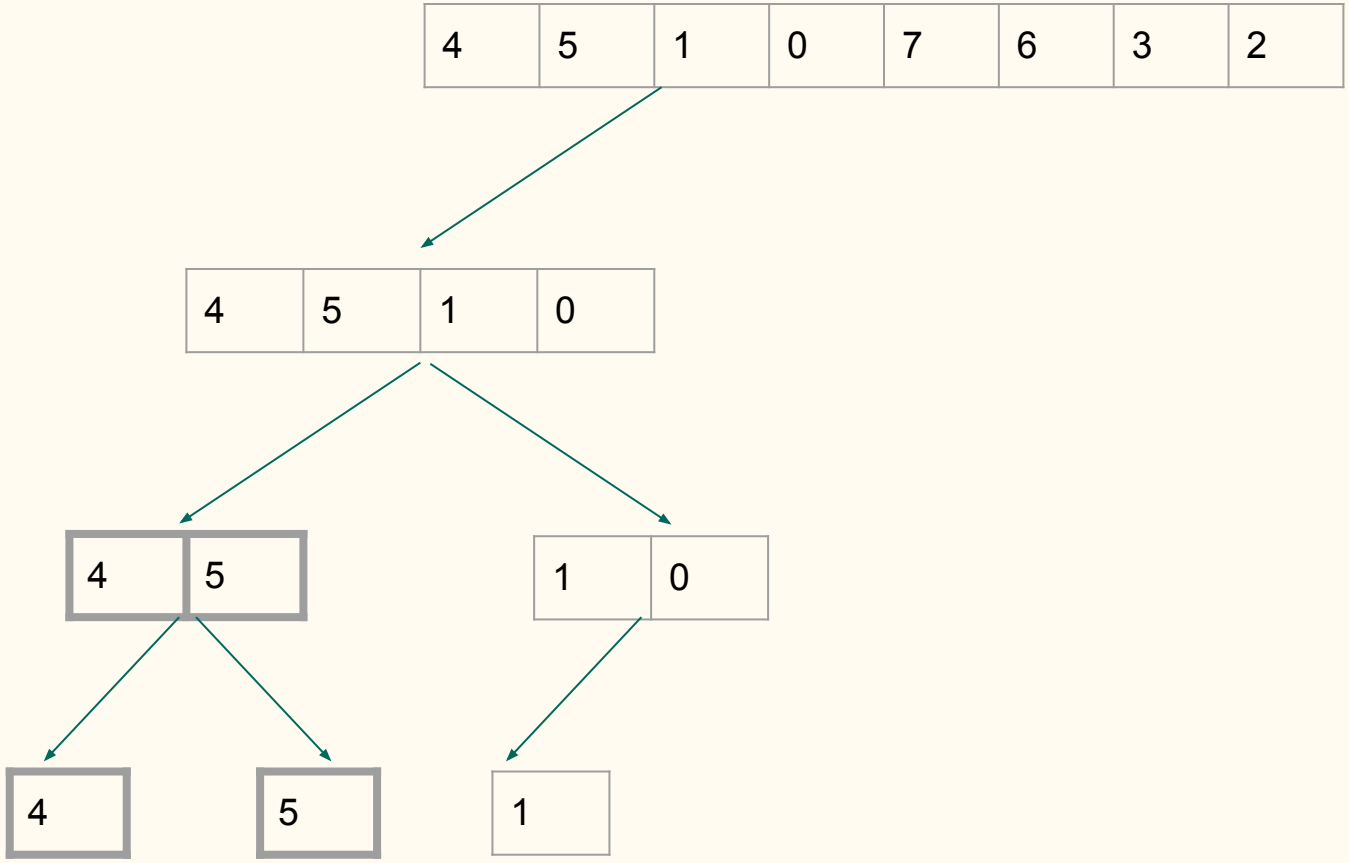
Merge Sort - Passo a passo



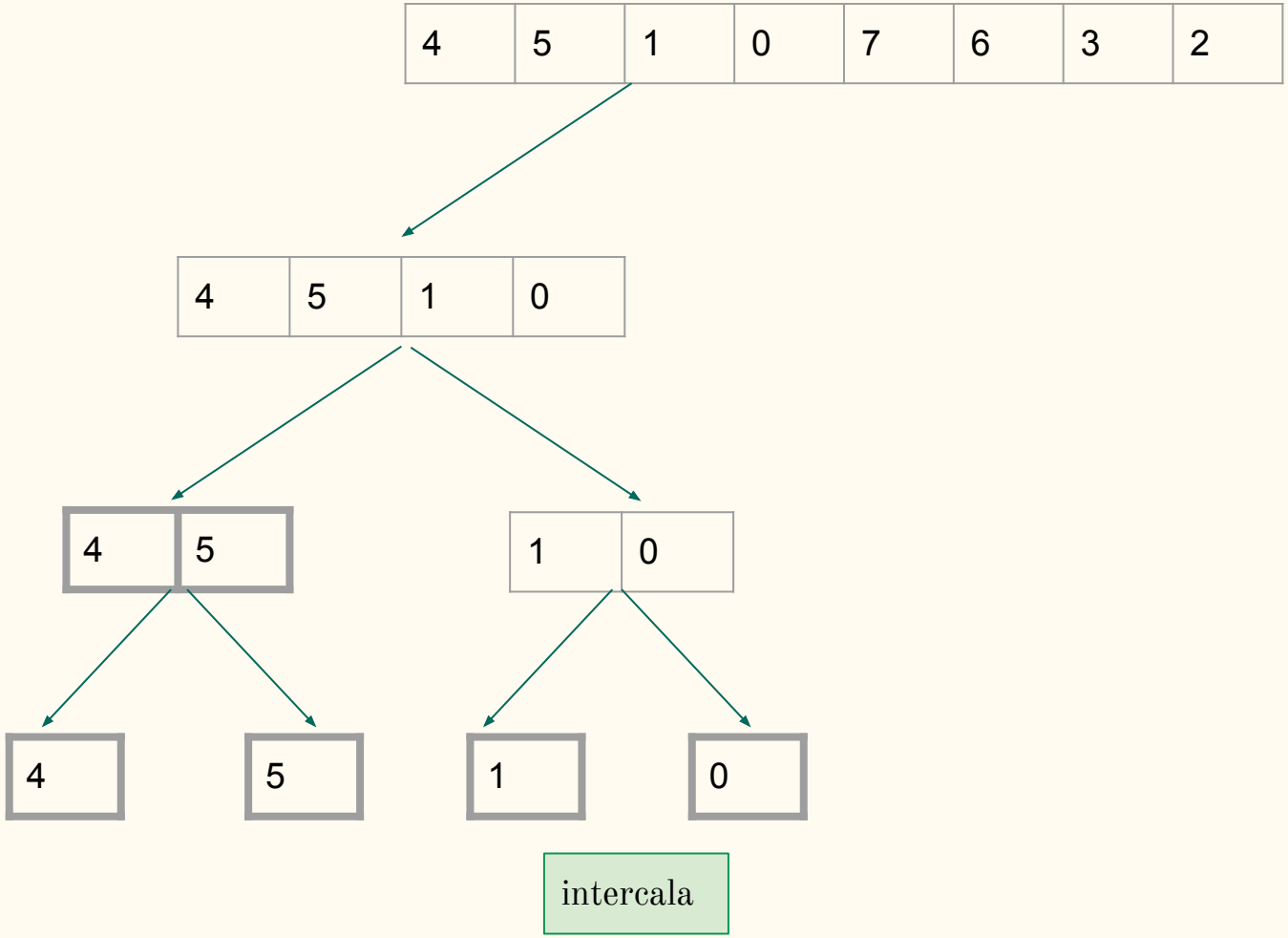
Merge Sort - Passo a passo



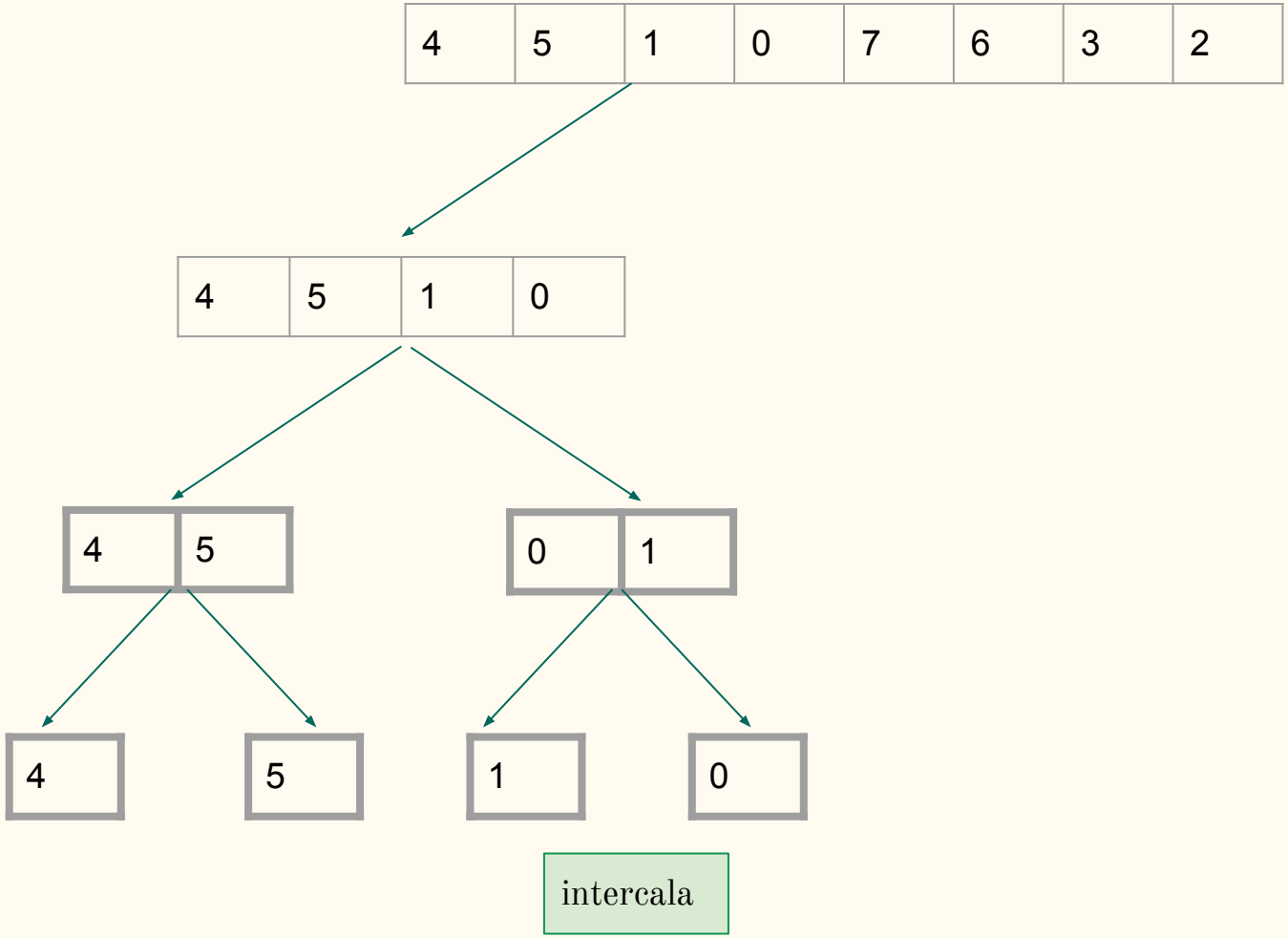
Merge Sort - Passo a passo



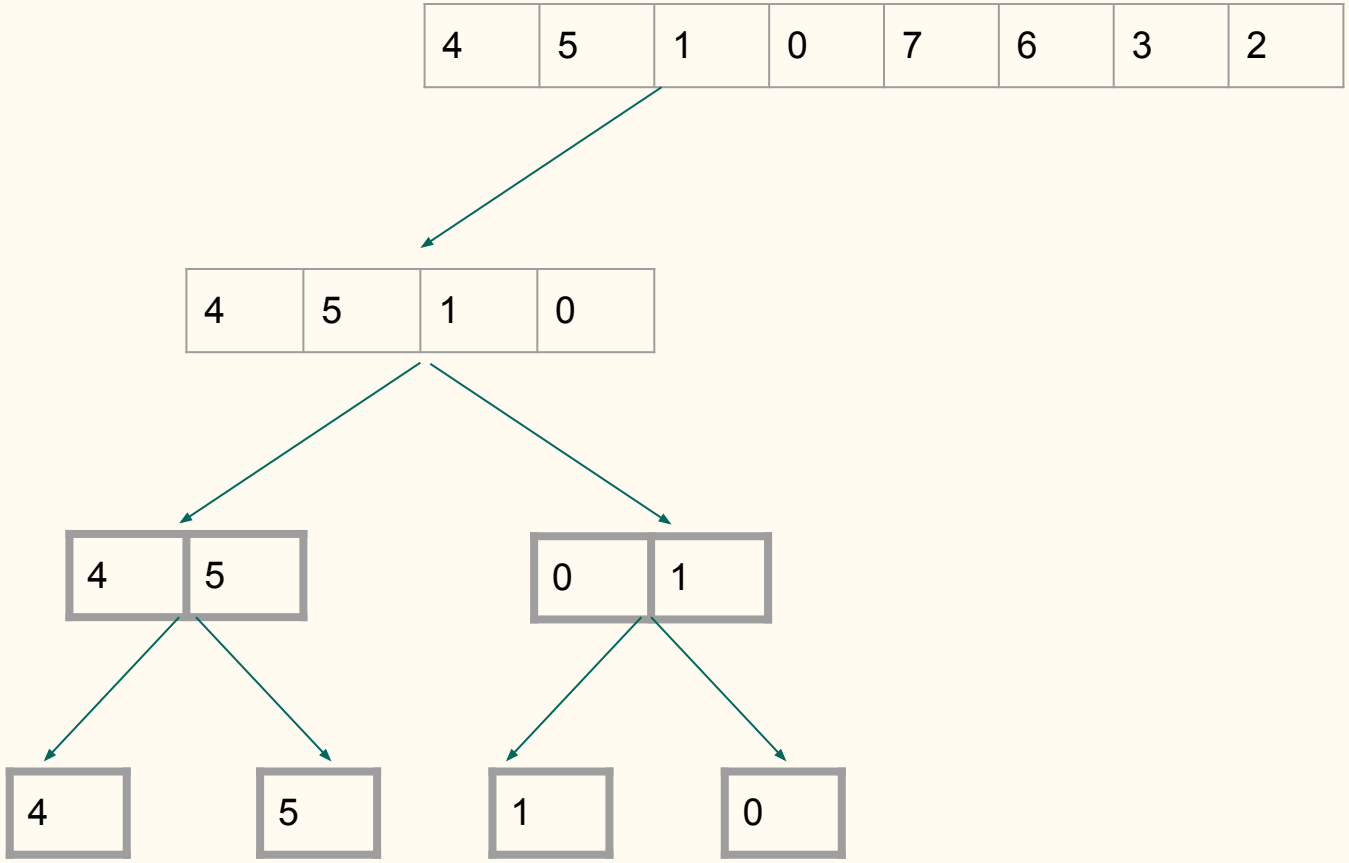
Merge Sort - Passo a passo



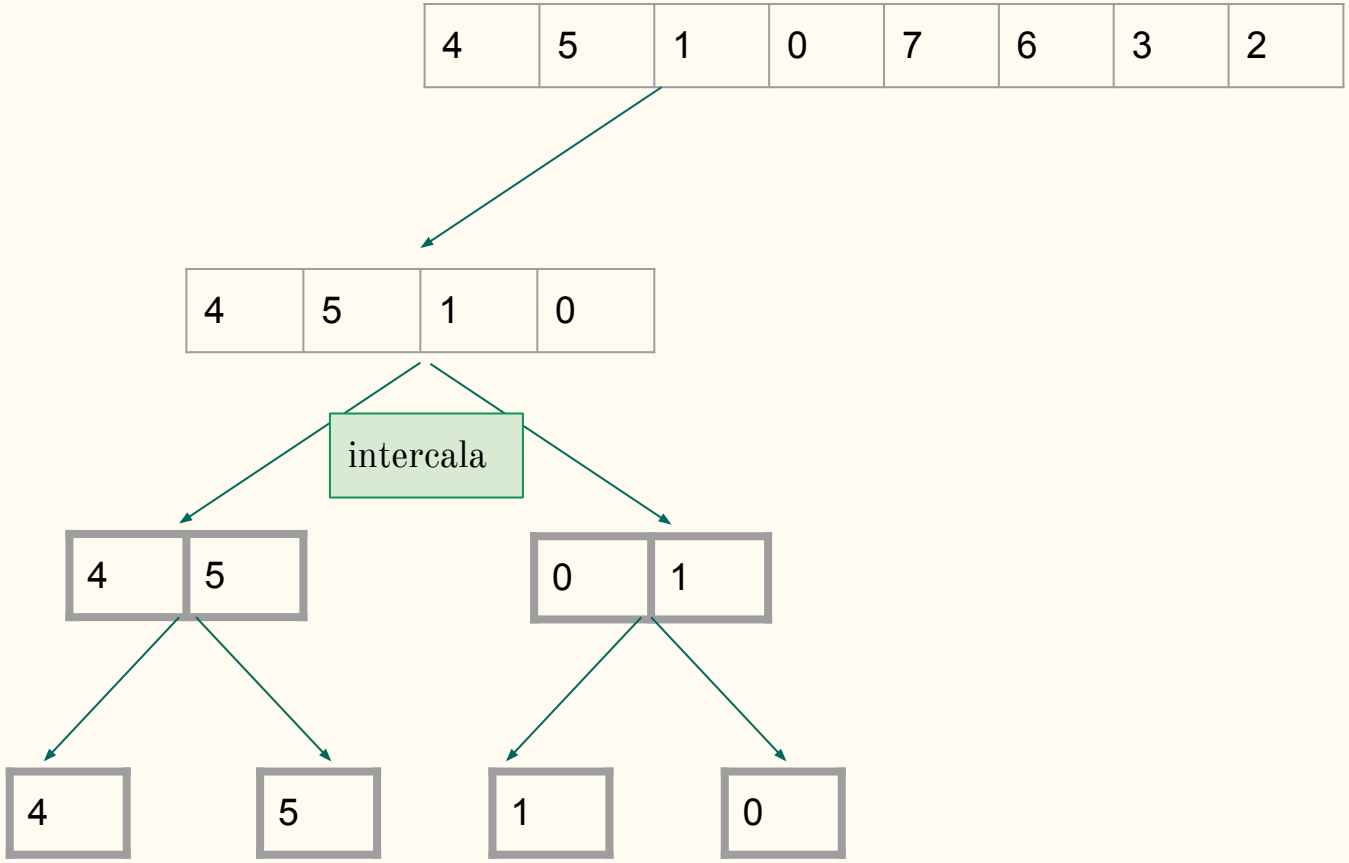
Merge Sort - Passo a passo



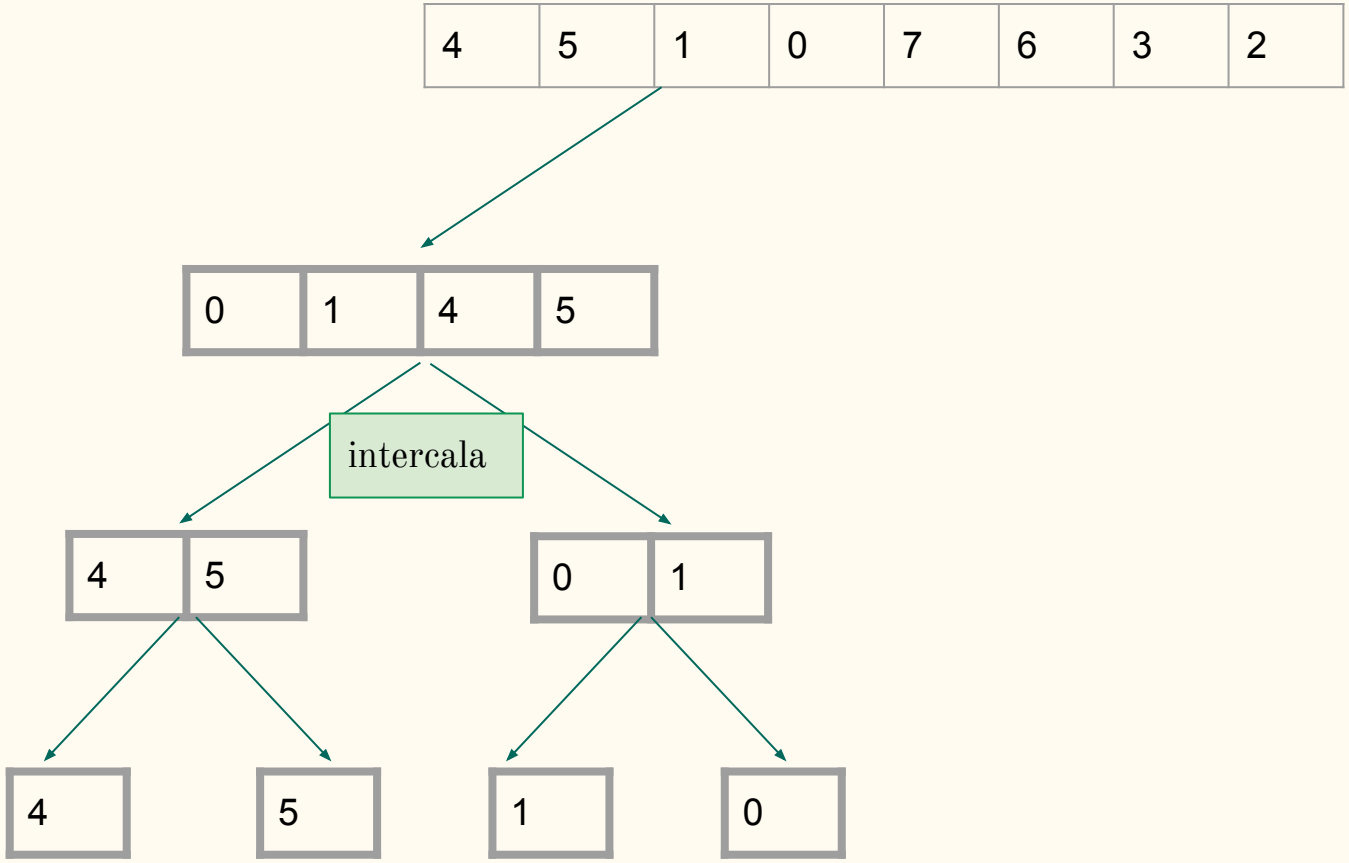
Merge Sort - Passo a passo



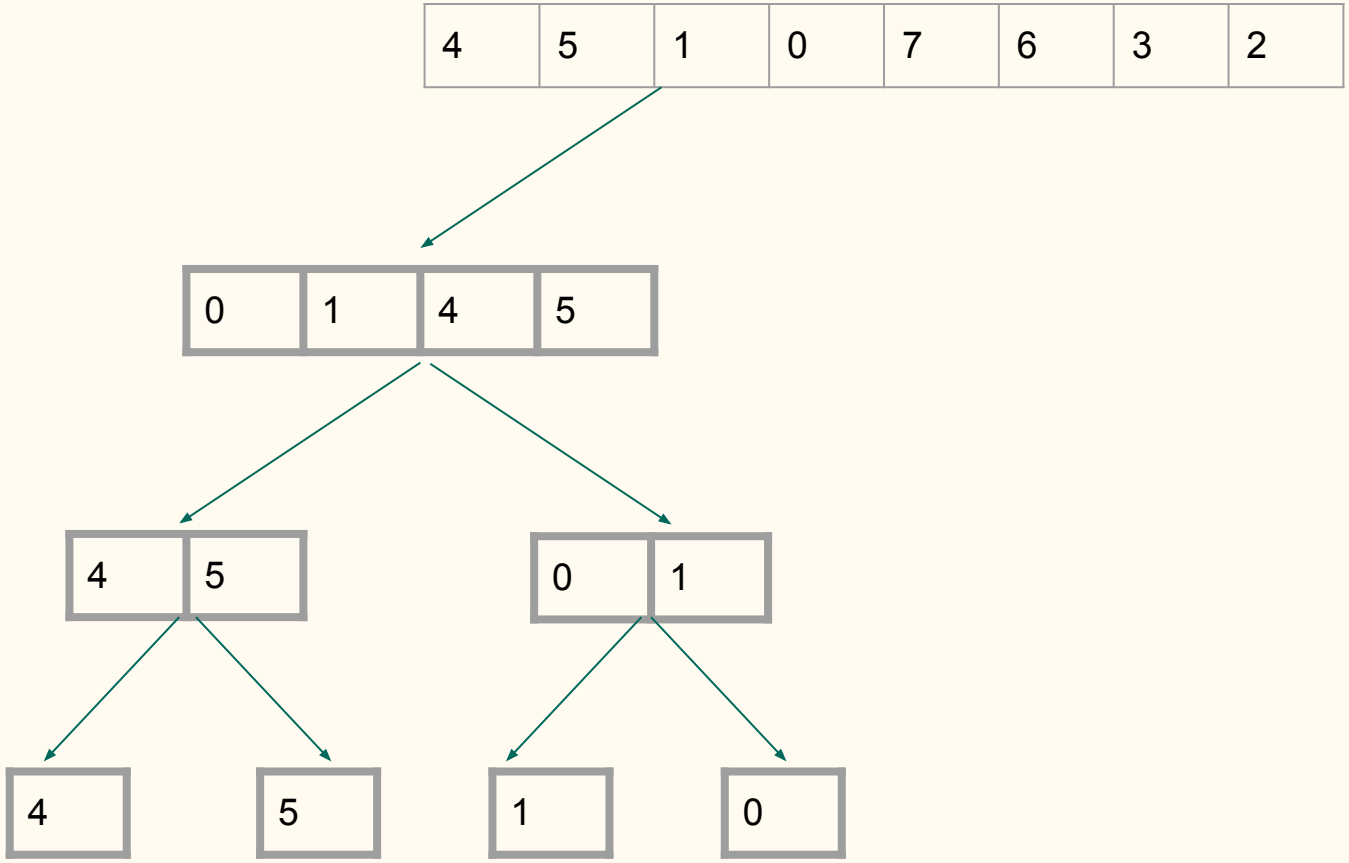
Merge Sort - Passo a passo



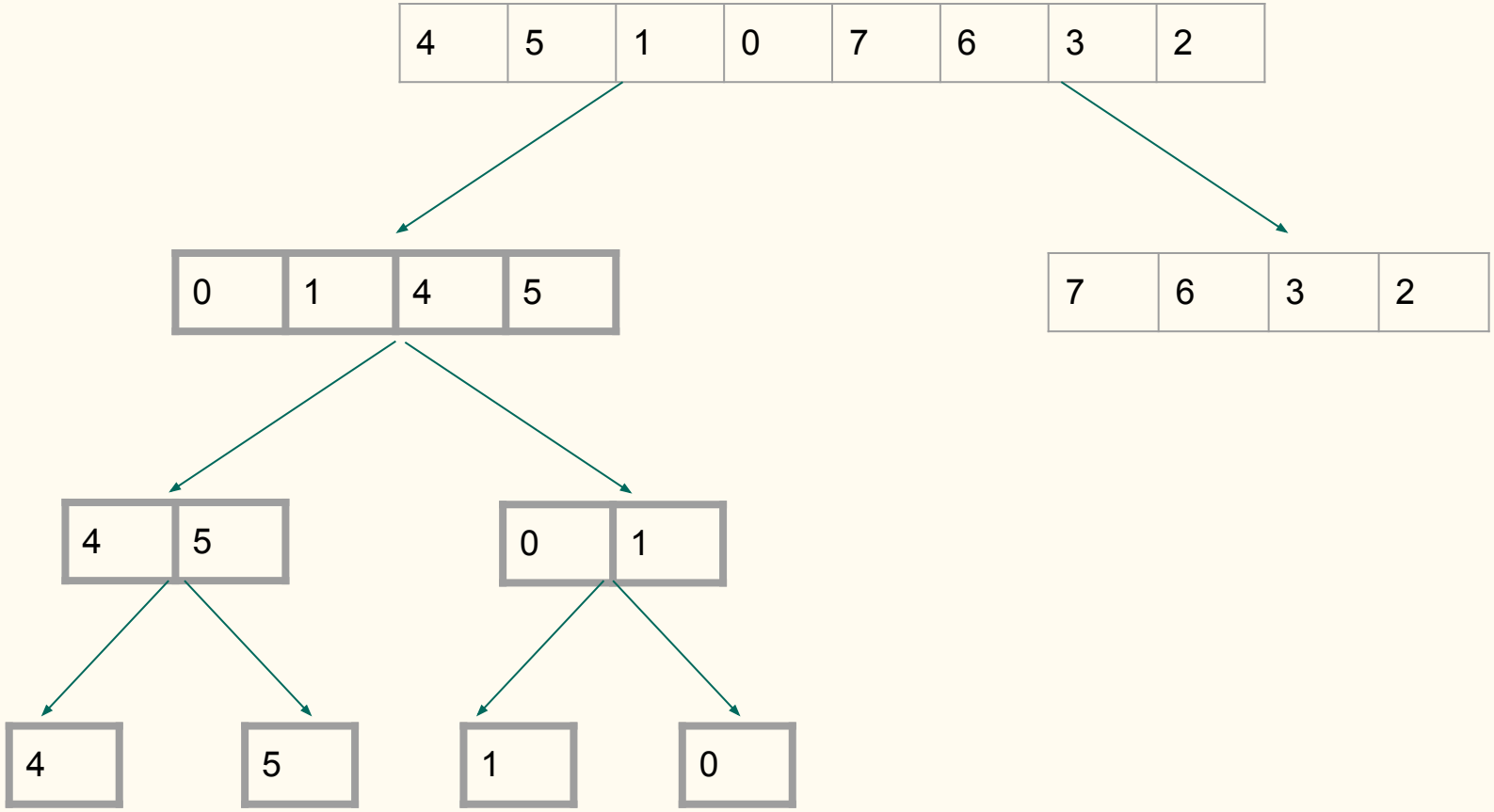
Merge Sort - Passo a passo



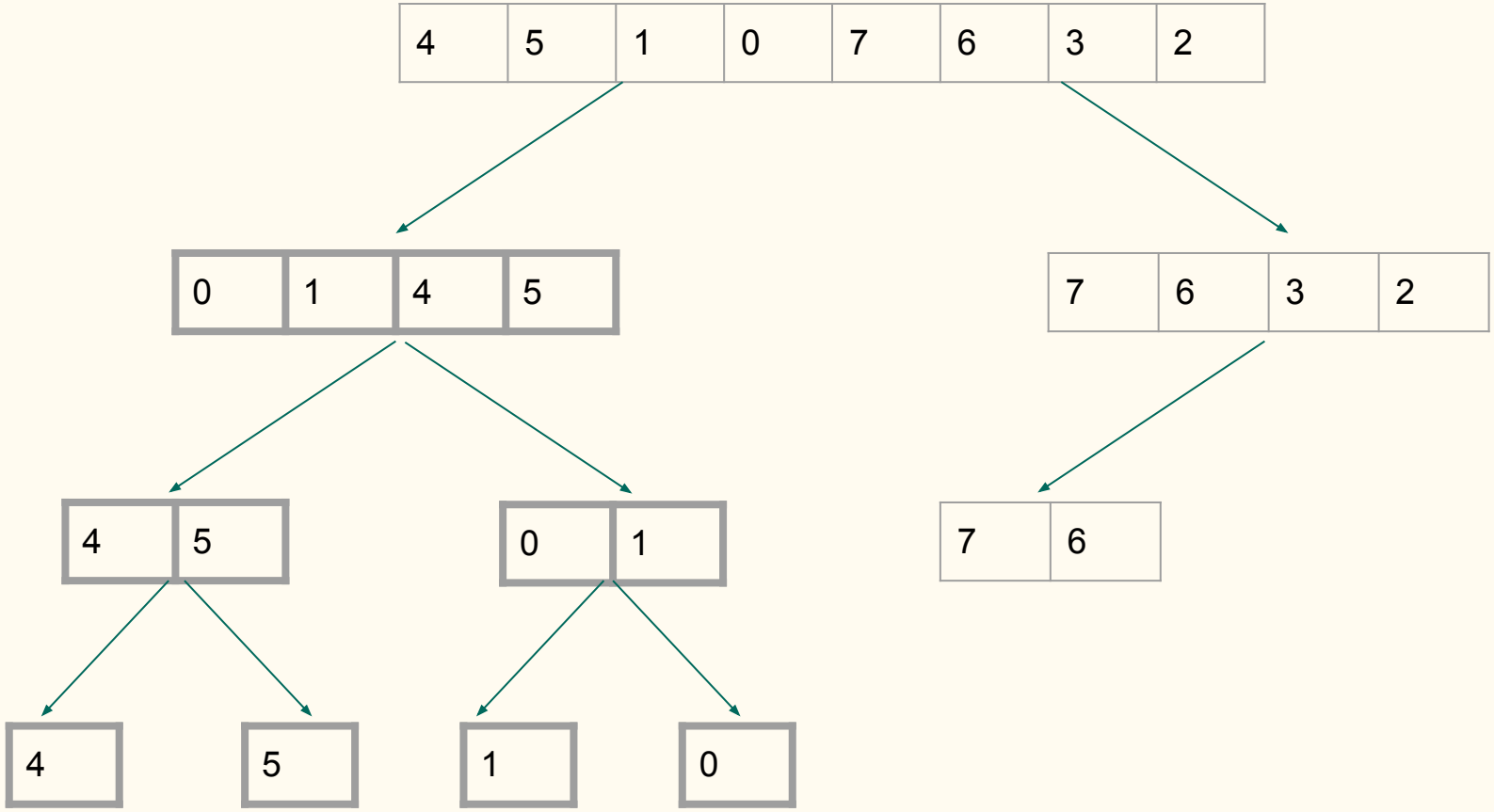
Merge Sort - Passo a passo



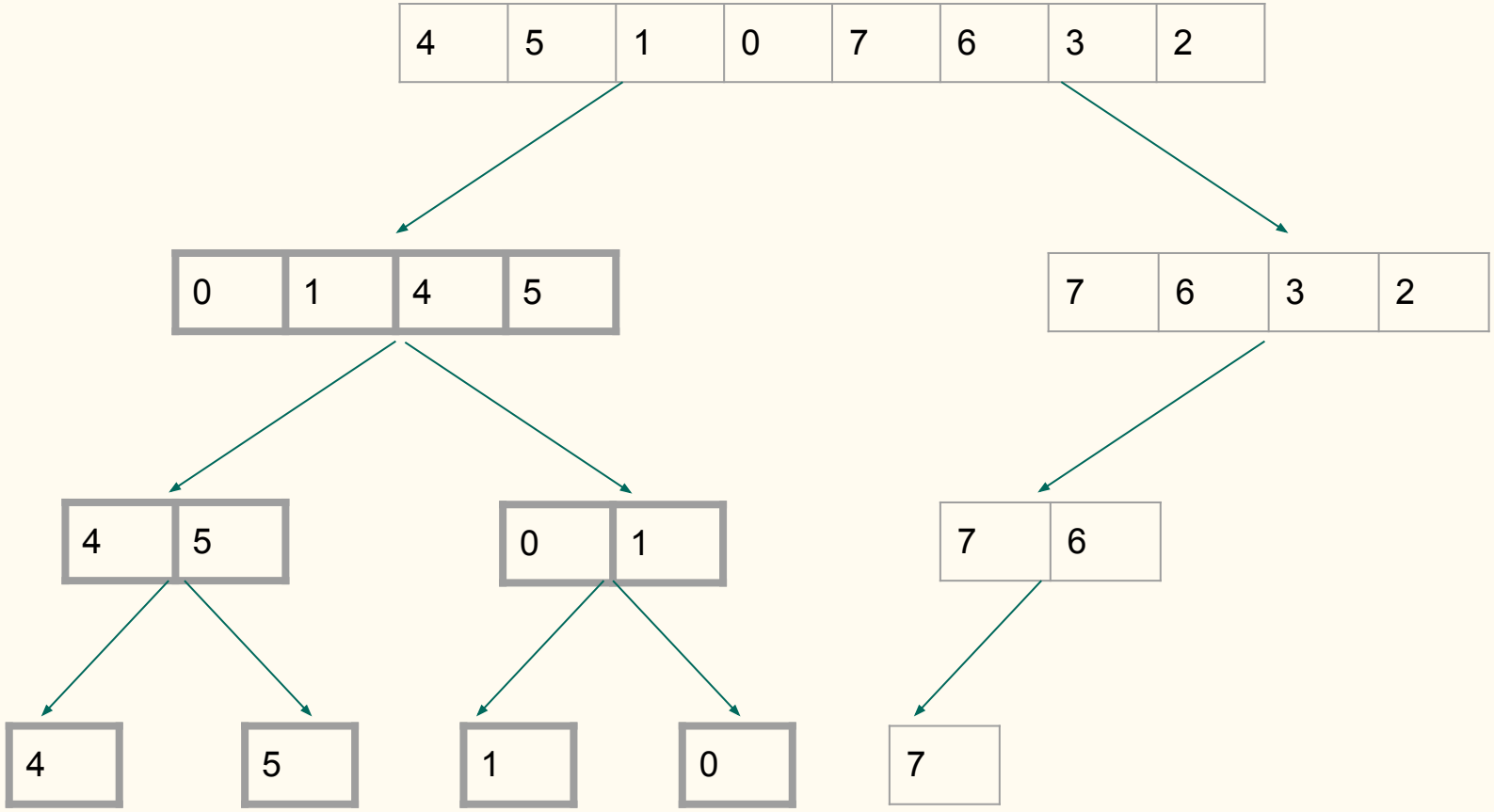
Merge Sort - Passo a passo



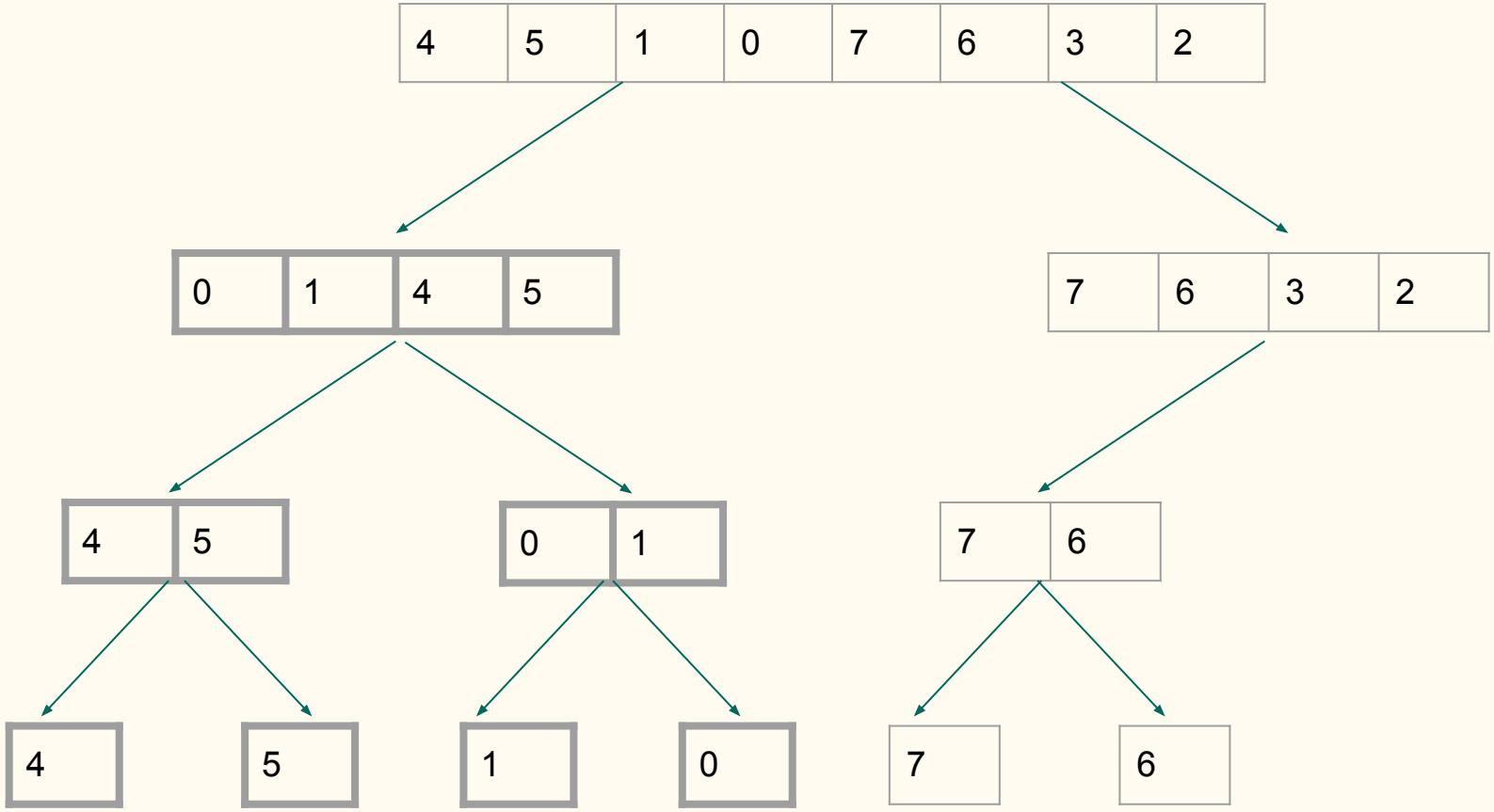
Merge Sort - Passo a passo



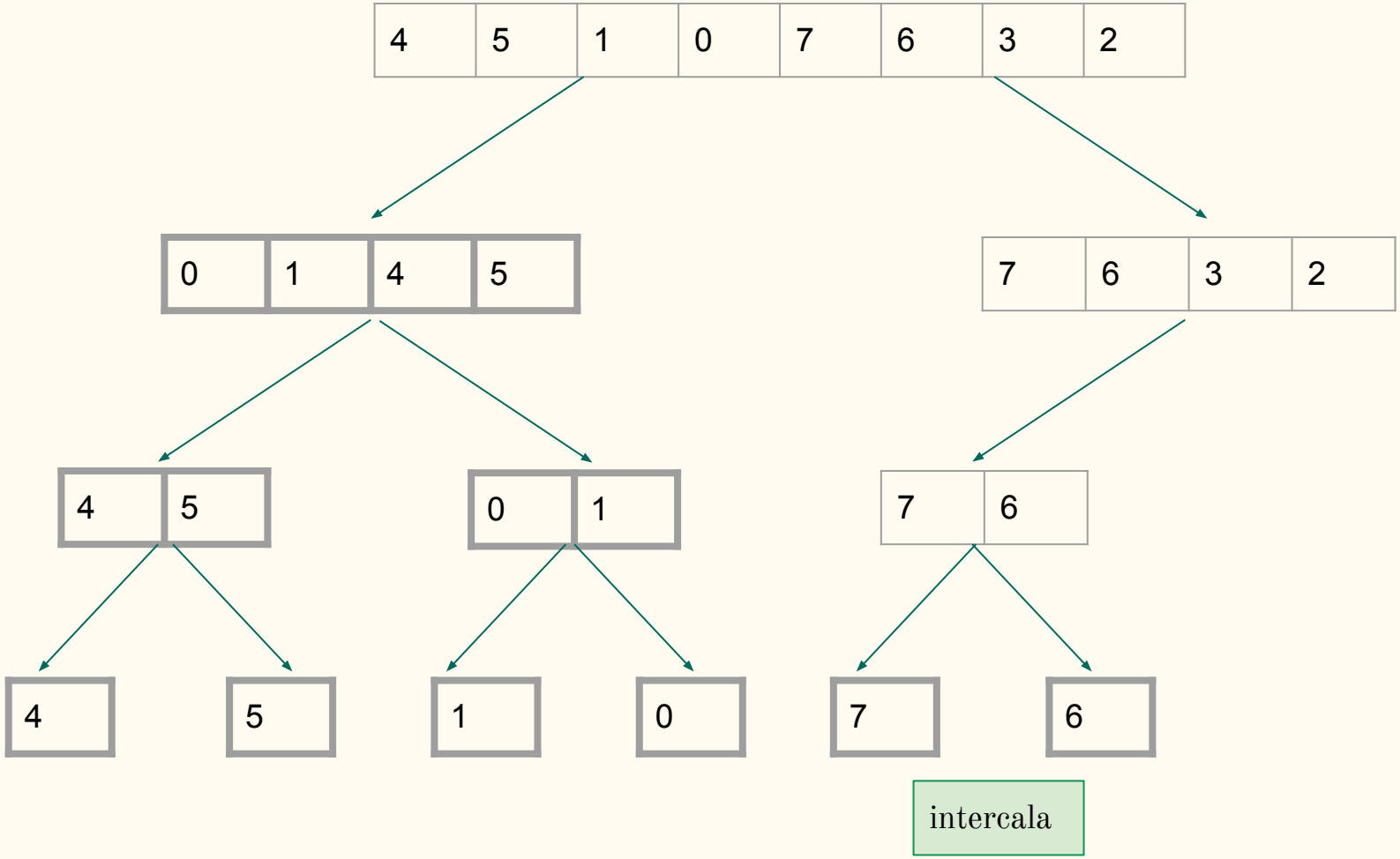
Merge Sort - Passo a passo



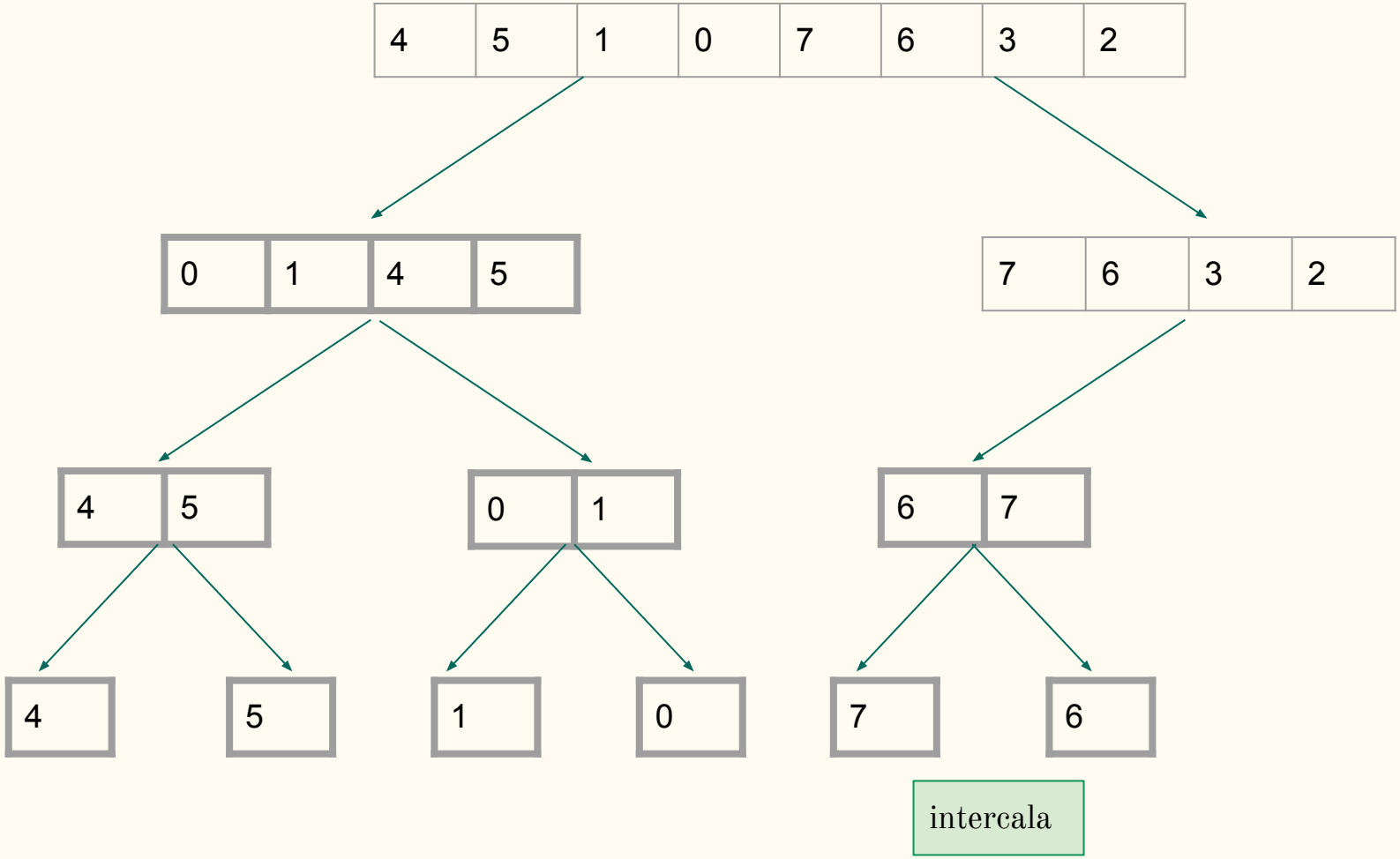
Merge Sort - Passo a passo



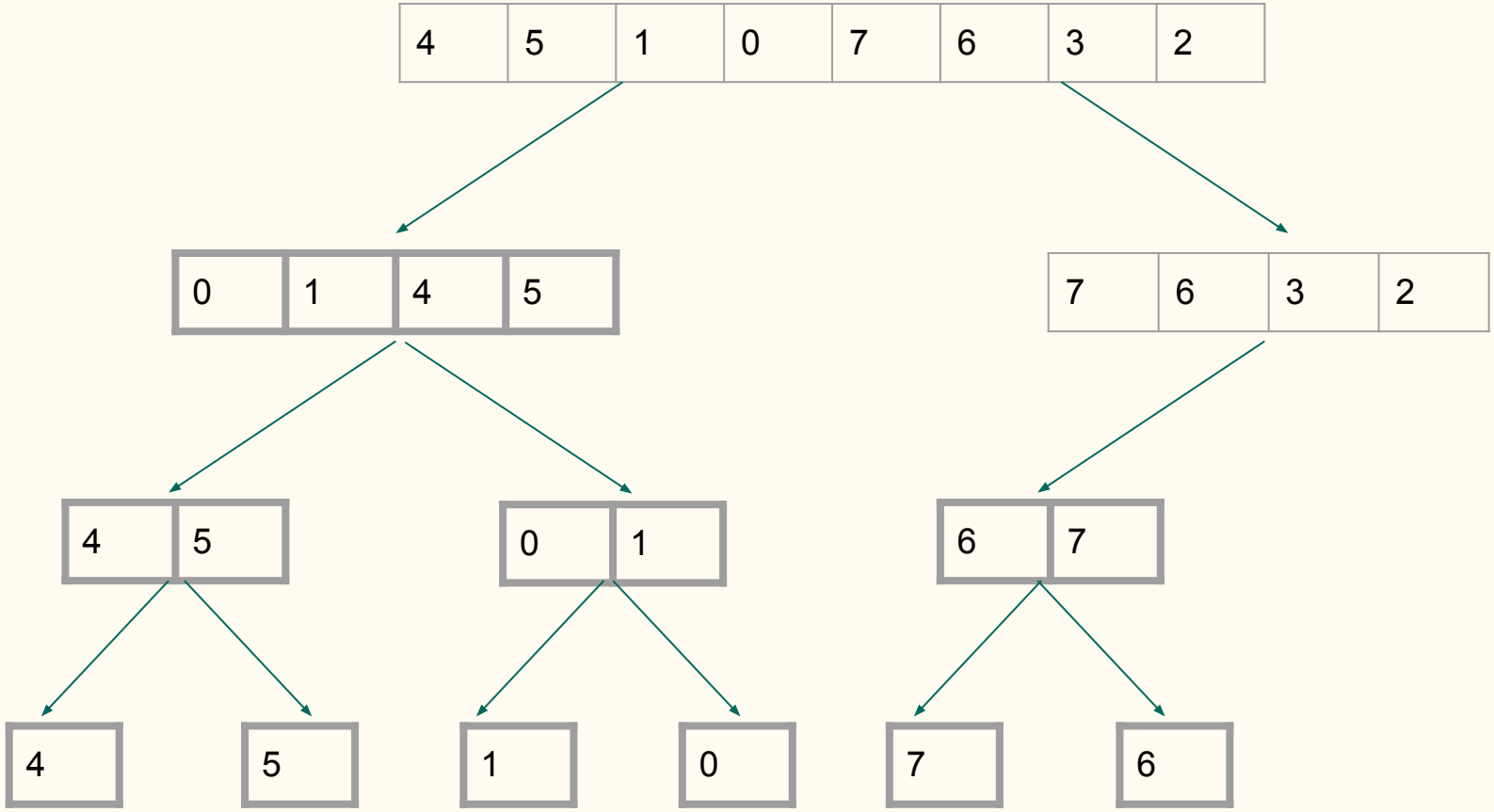
Merge Sort - Passo a passo



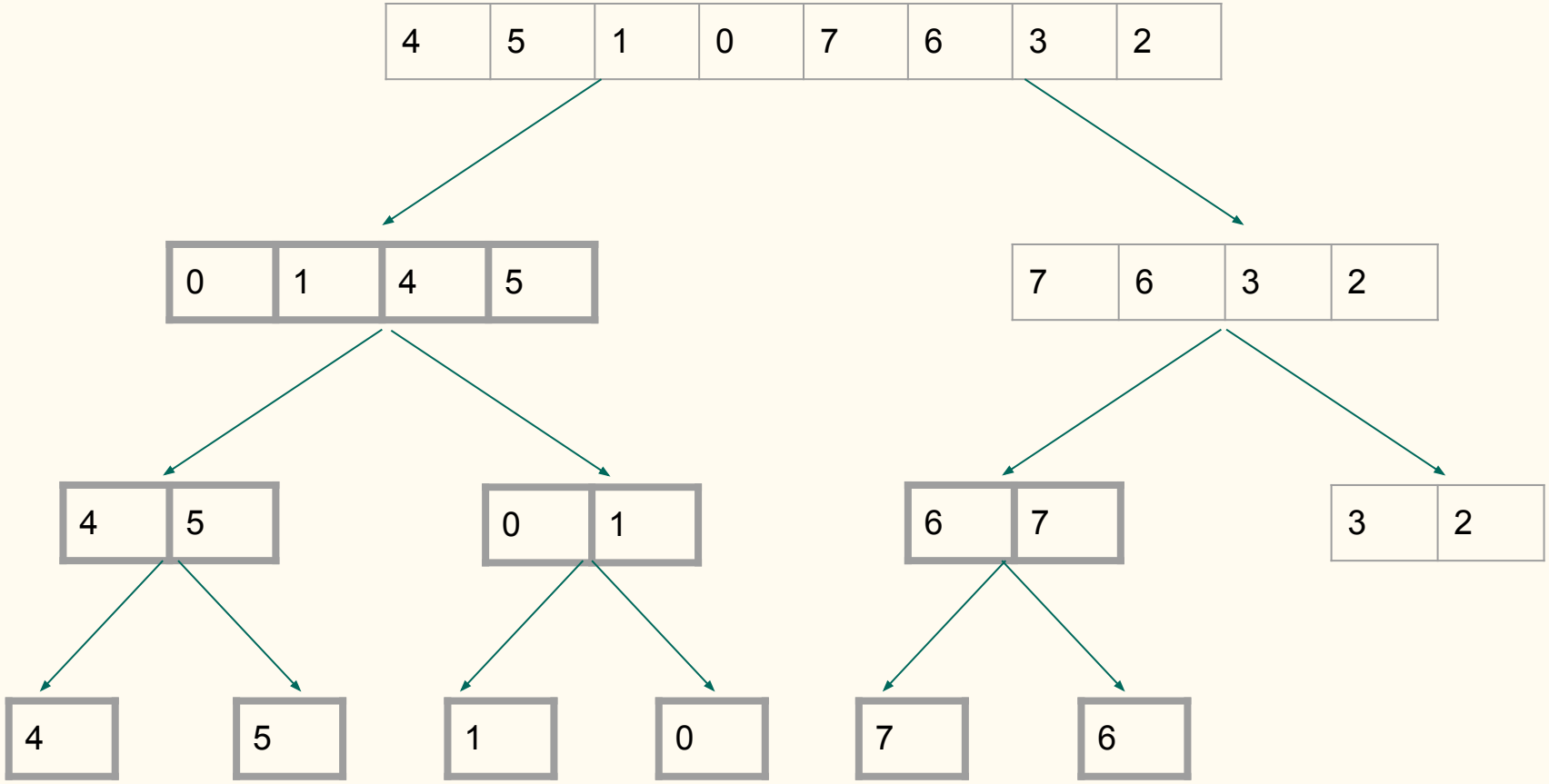
Merge Sort - Passo a passo



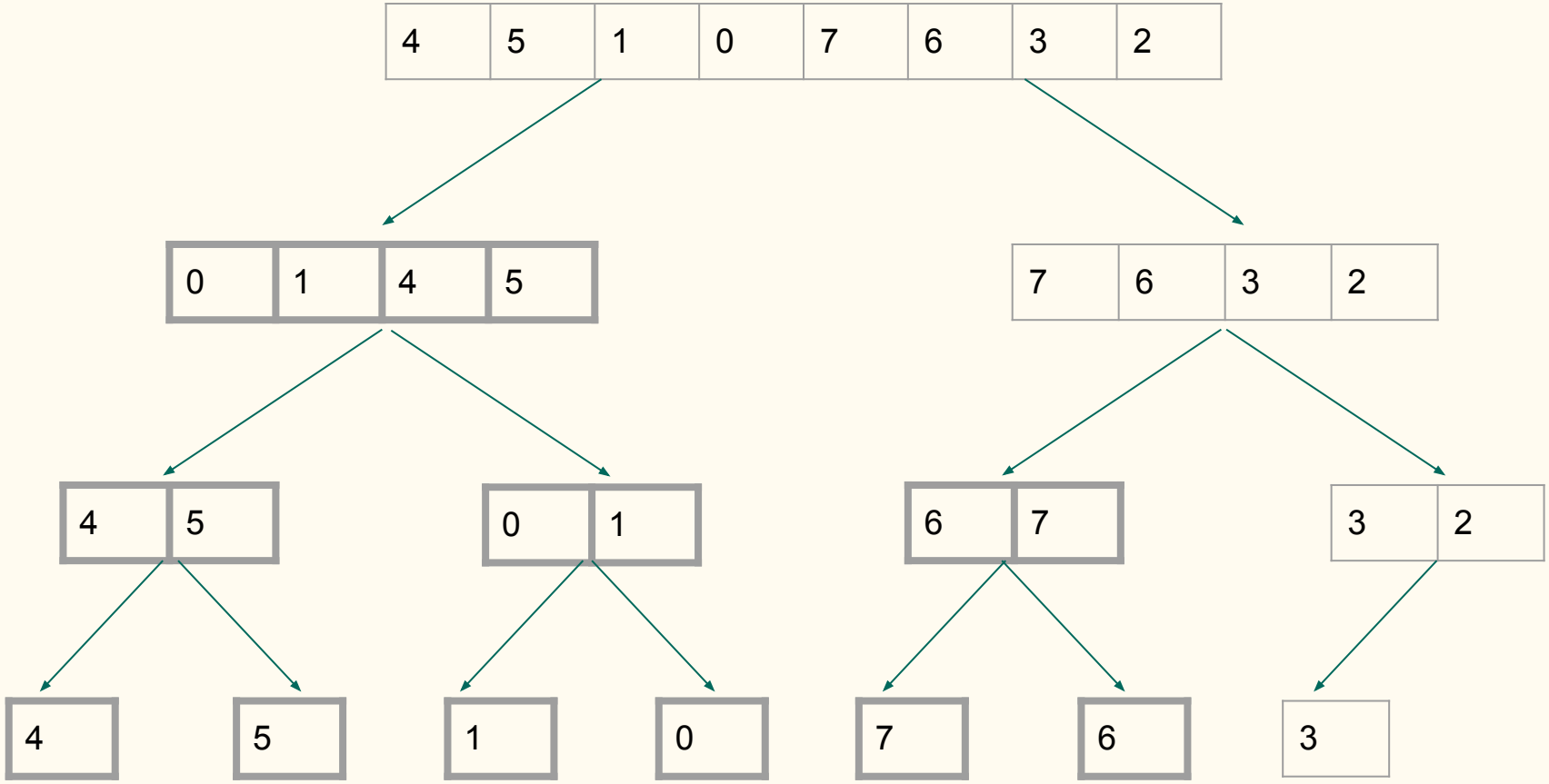
Merge Sort - Passo a passo



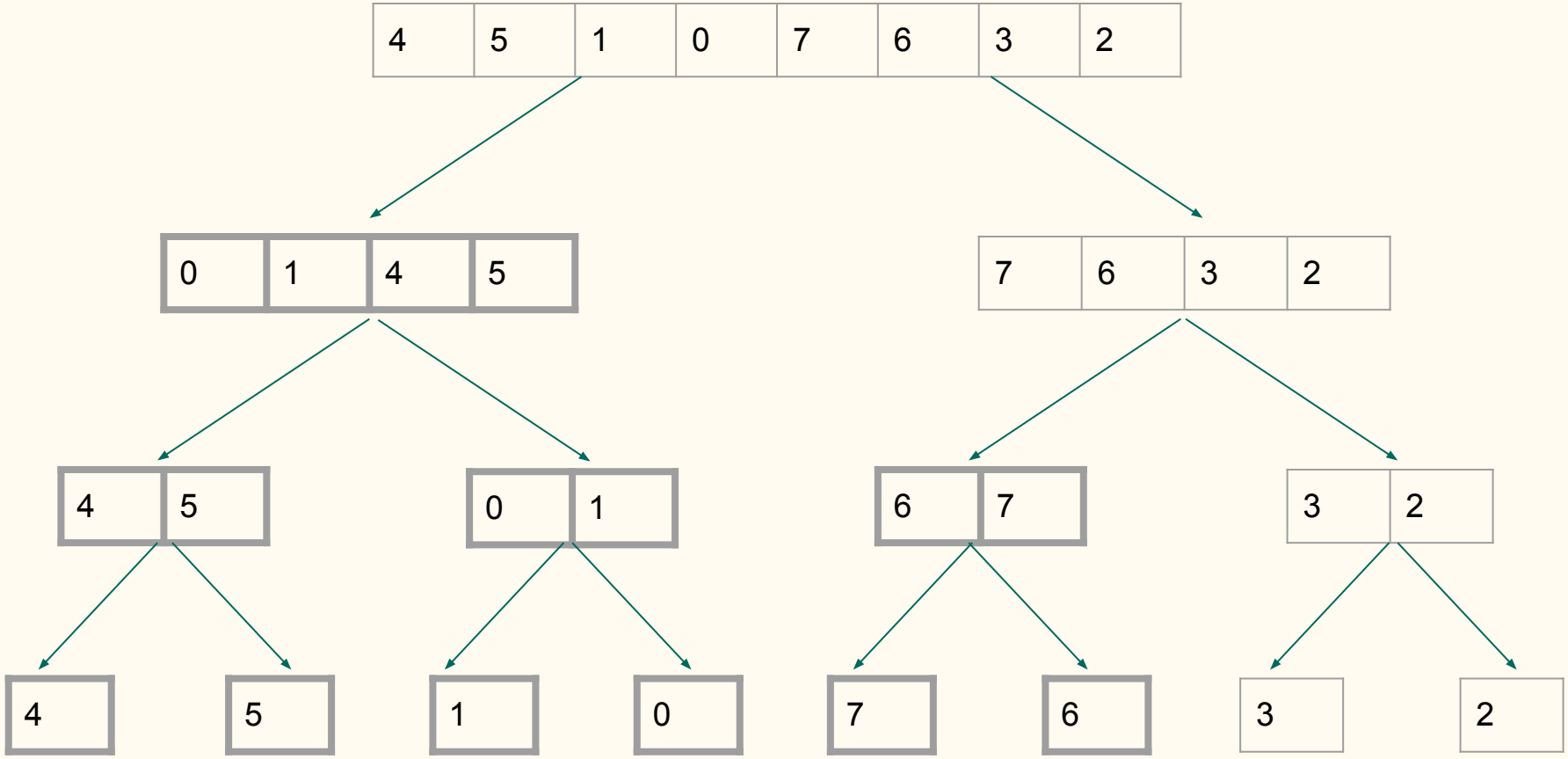
Merge Sort - Passo a passo



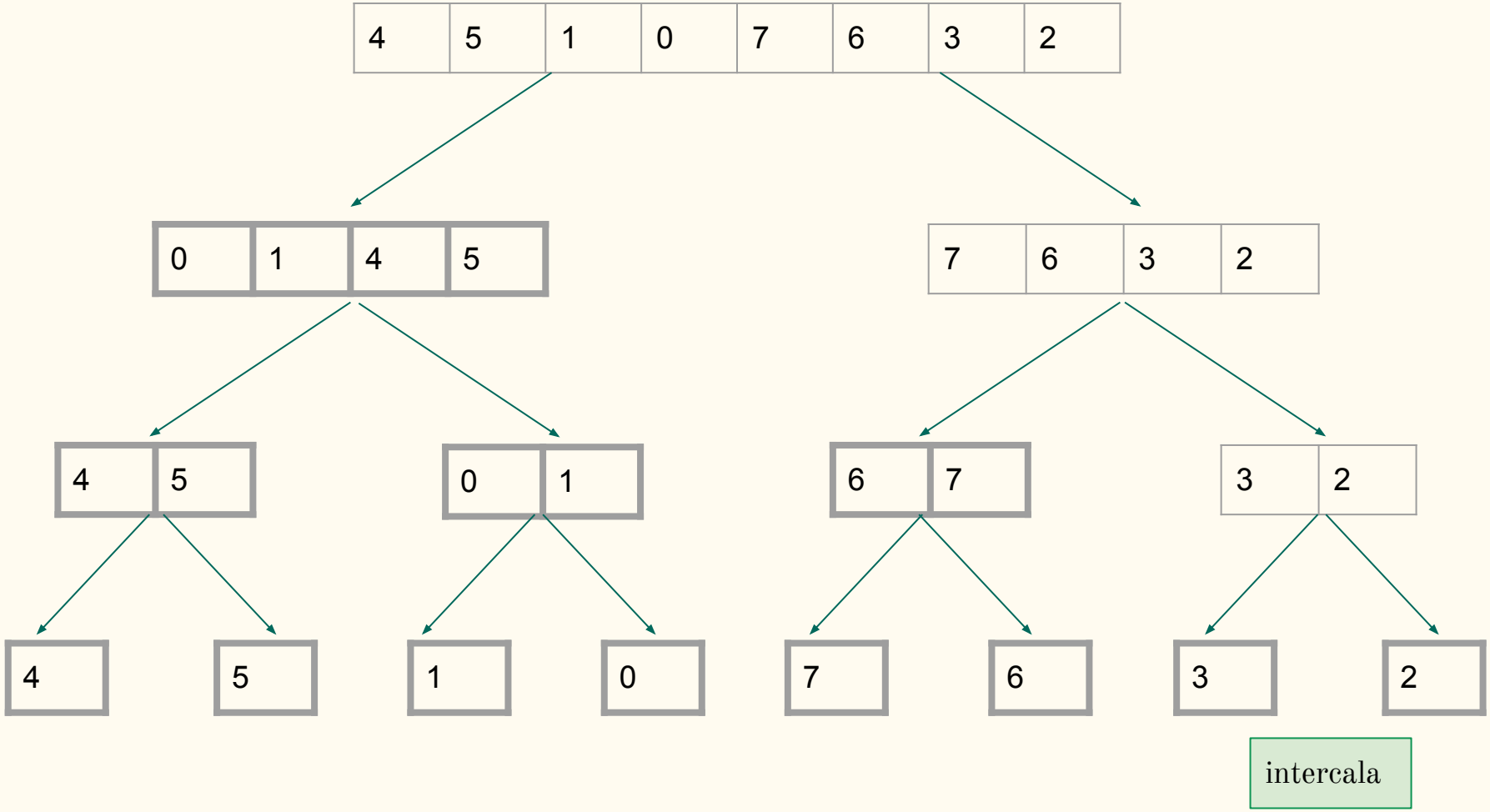
Merge Sort - Passo a passo



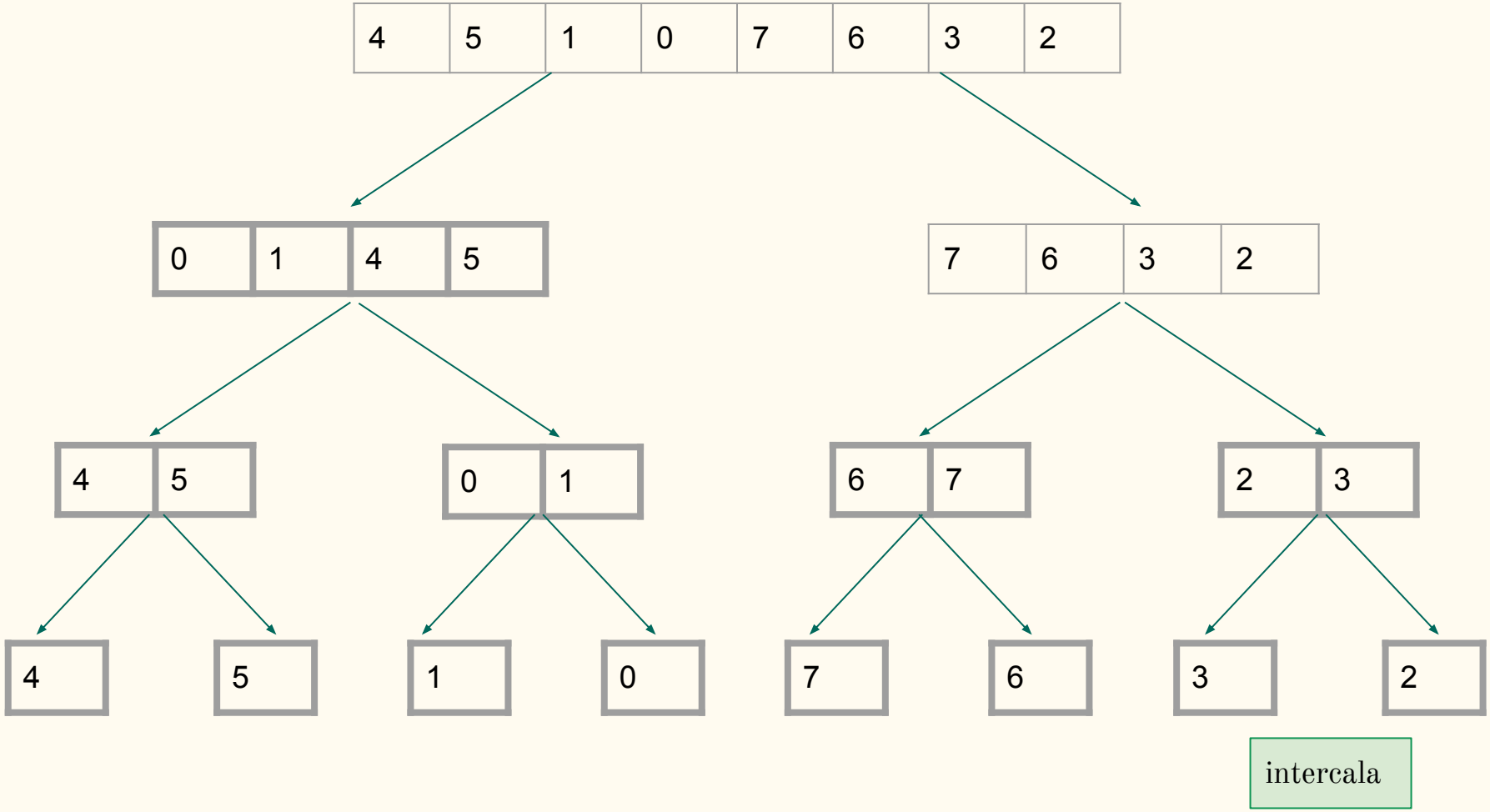
Merge Sort - Passo a passo



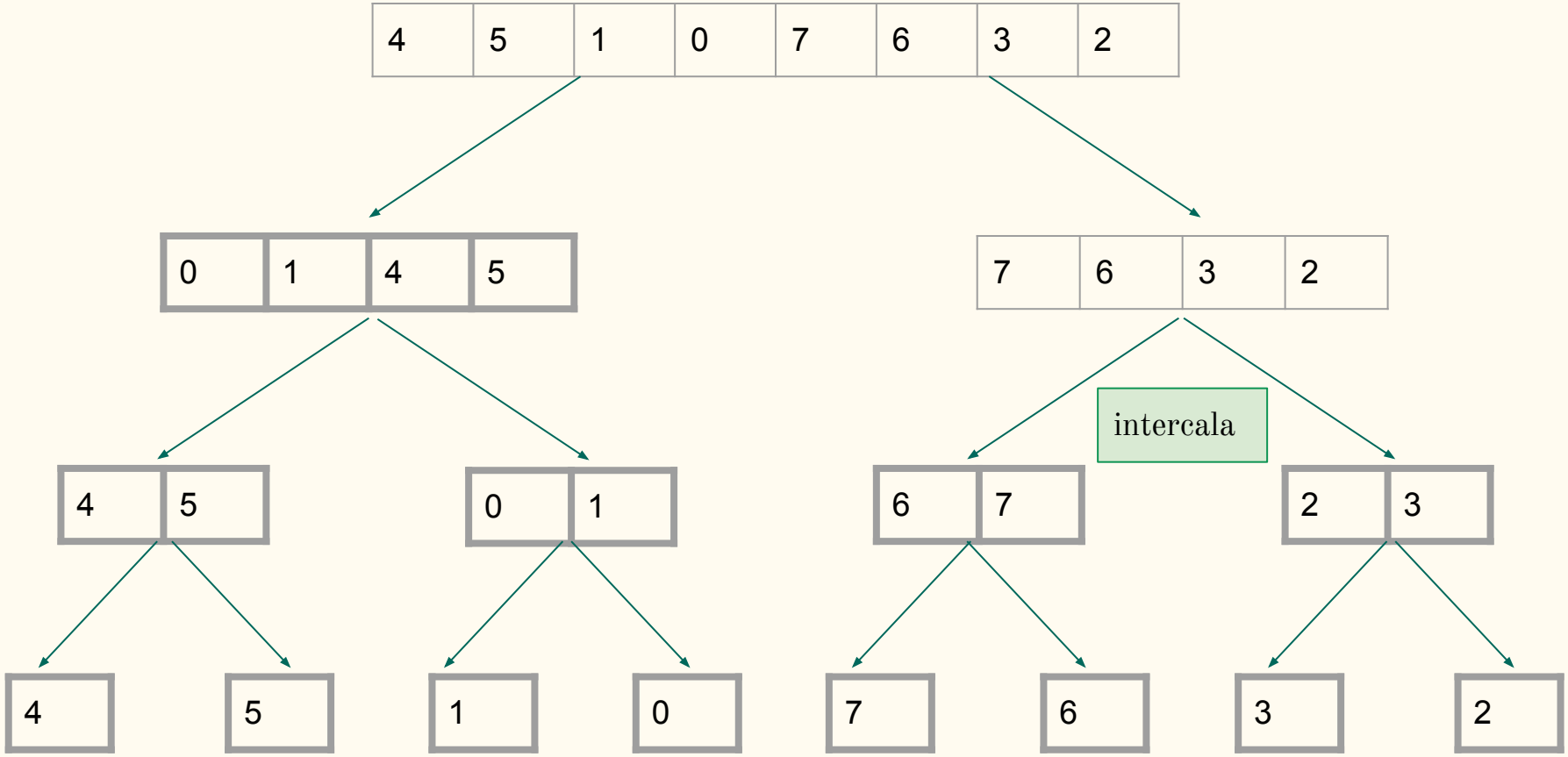
Merge Sort - Passo a passo



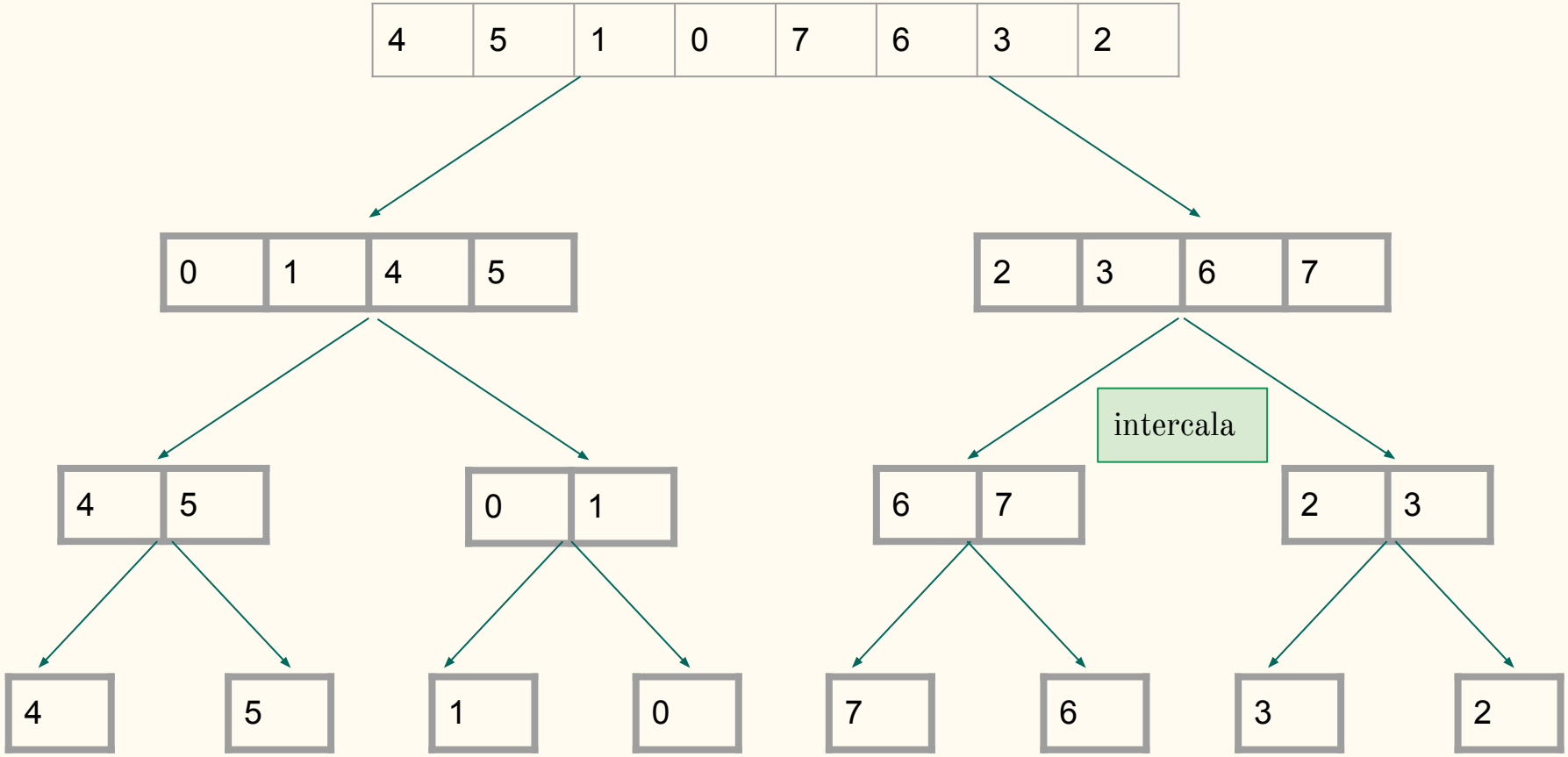
Merge Sort - Passo a passo



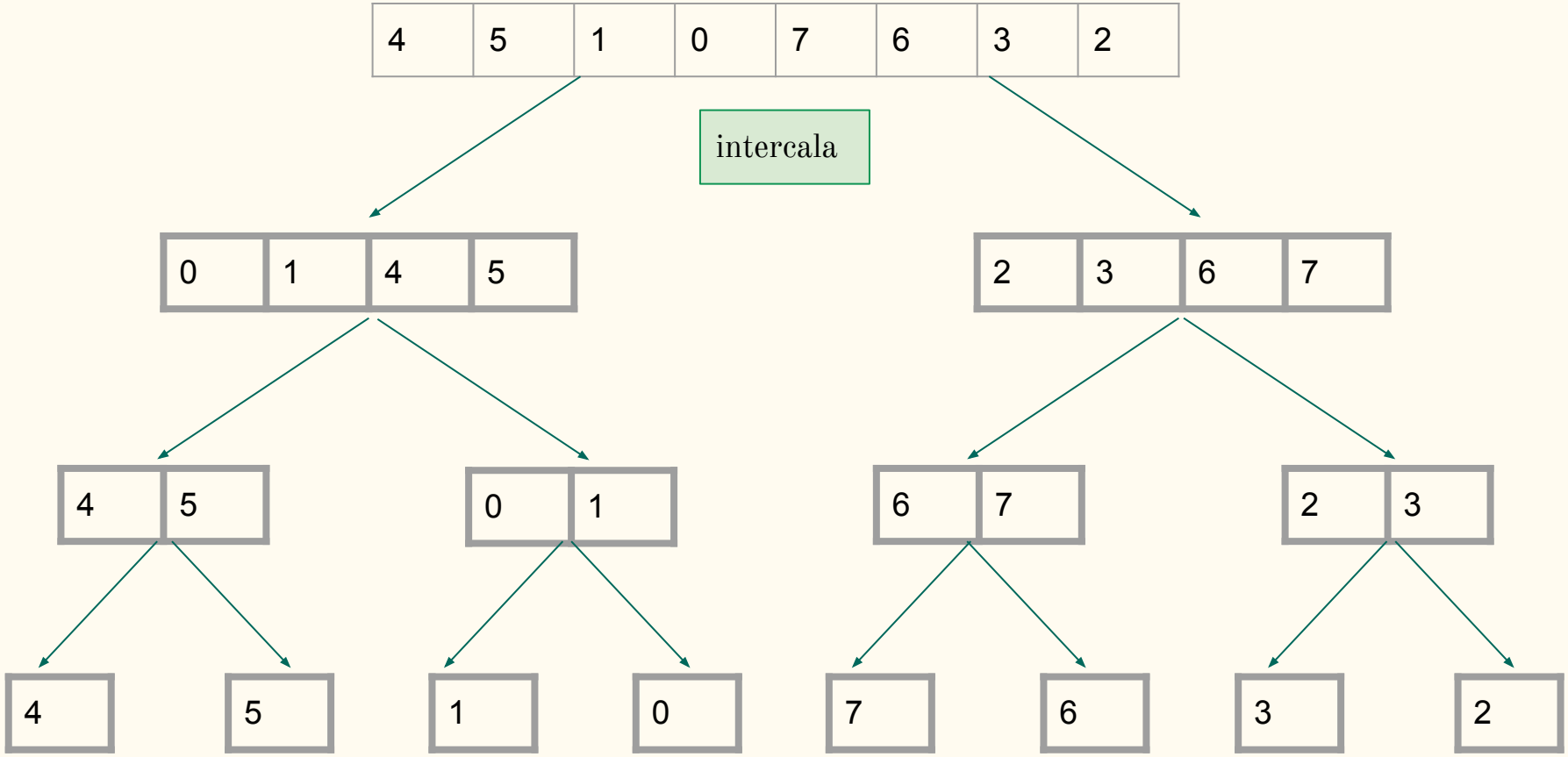
Merge Sort - Passo a passo



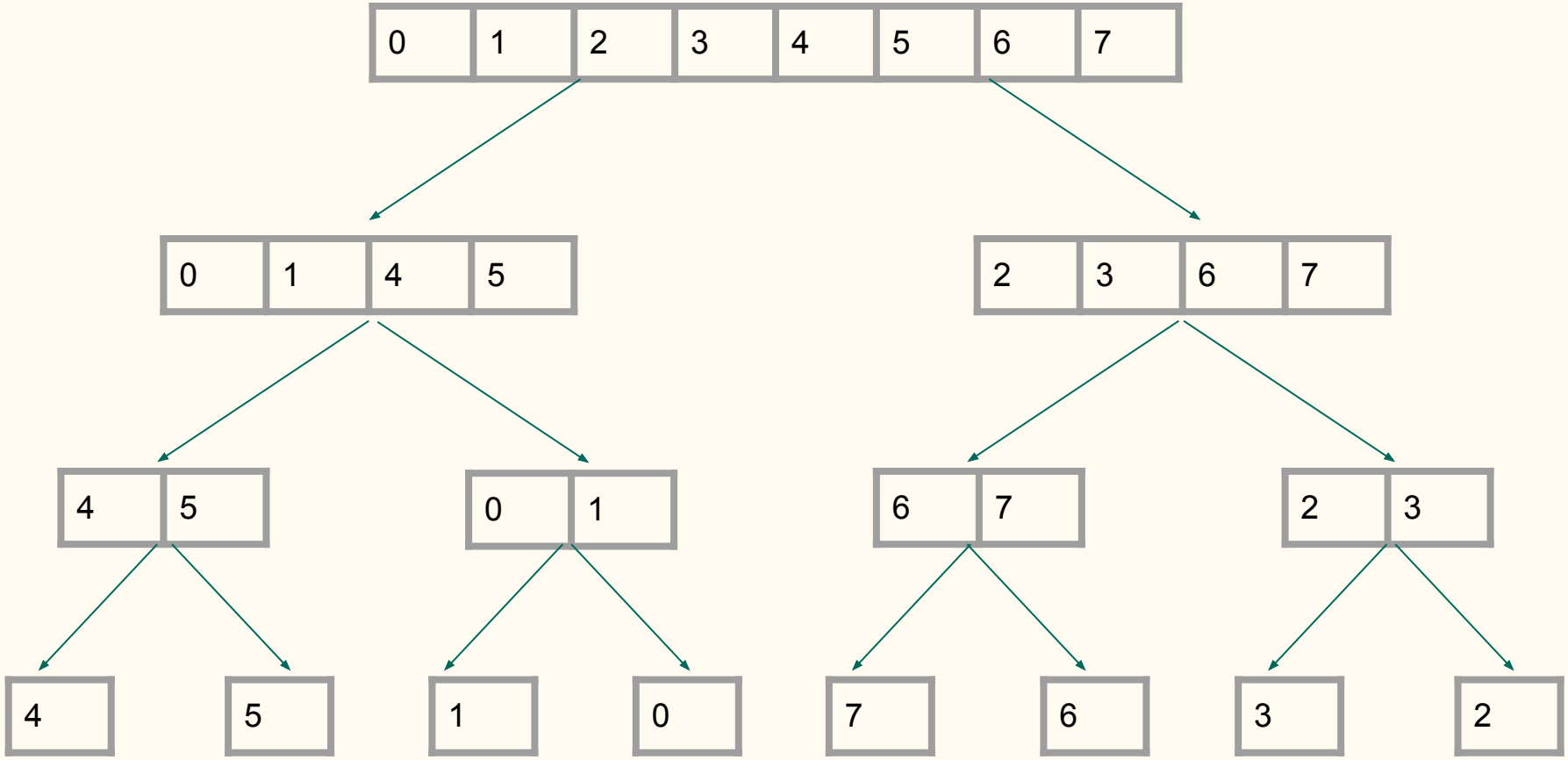
Merge Sort - Passo a passo



Merge Sort - Passo a passo



Merge Sort - Passo a passo



Merge Sort - Passo a passo do intercalar

- Objetivo: Combinar 2 vetores ordenados em 1 único vetor ordenado
- Ideia: comparar menor valor do v1 com v2, copiar menor valor

- v1:

0	1	4	5

v2:

2	3	6	7

- vfinal:

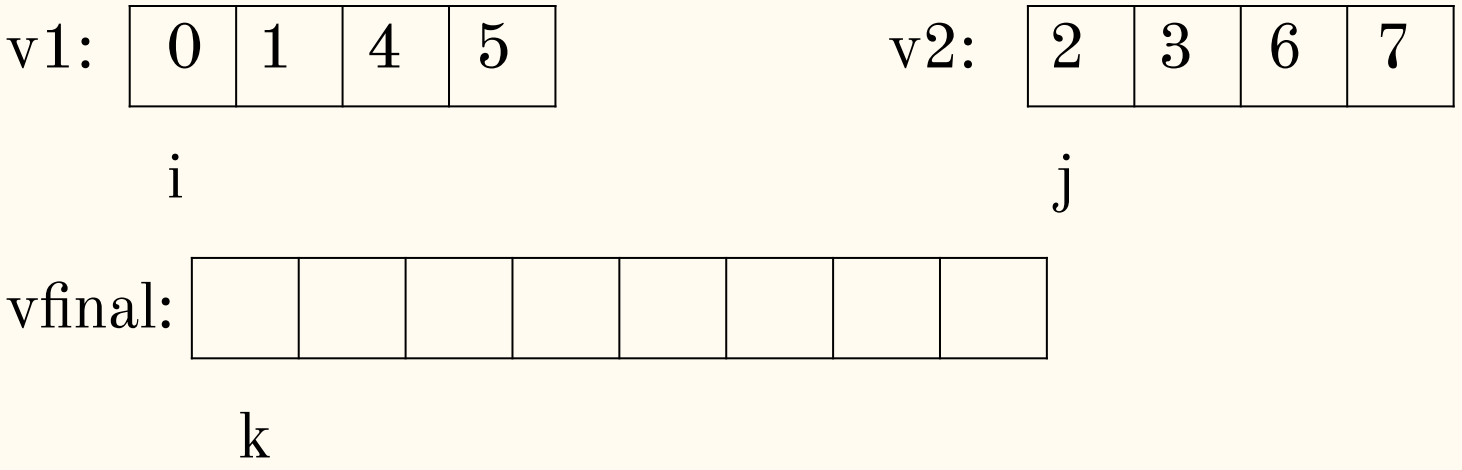
0	1	2	3	4	5	6	7

i percorre v1;

j percorre v2;

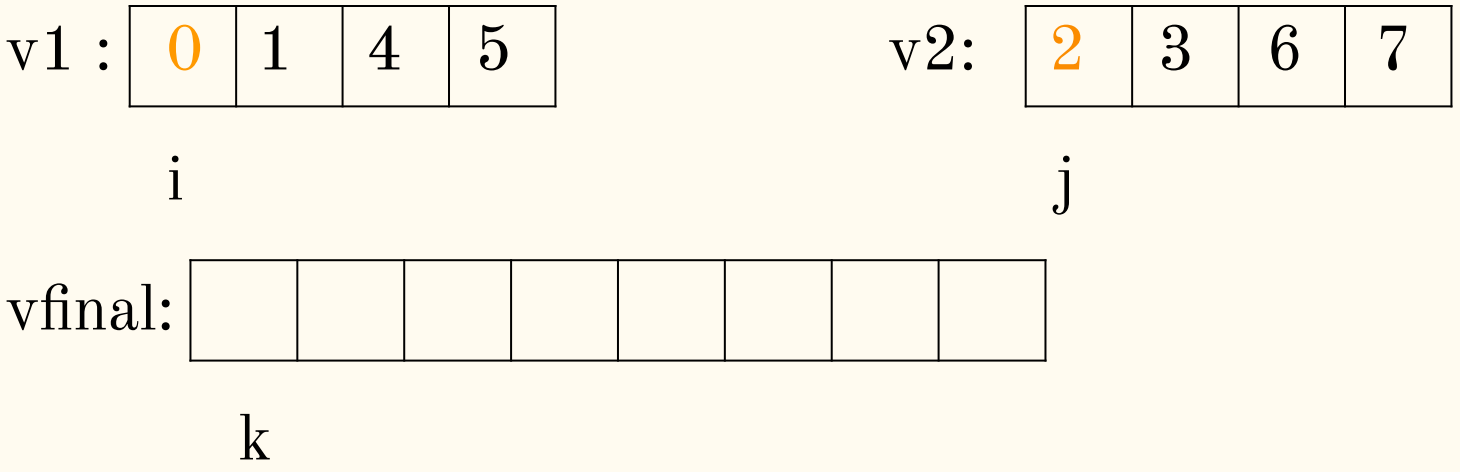
k percorre vfinal, inicialmente vazio.

Merge Sort - Passo a passo do intercalar



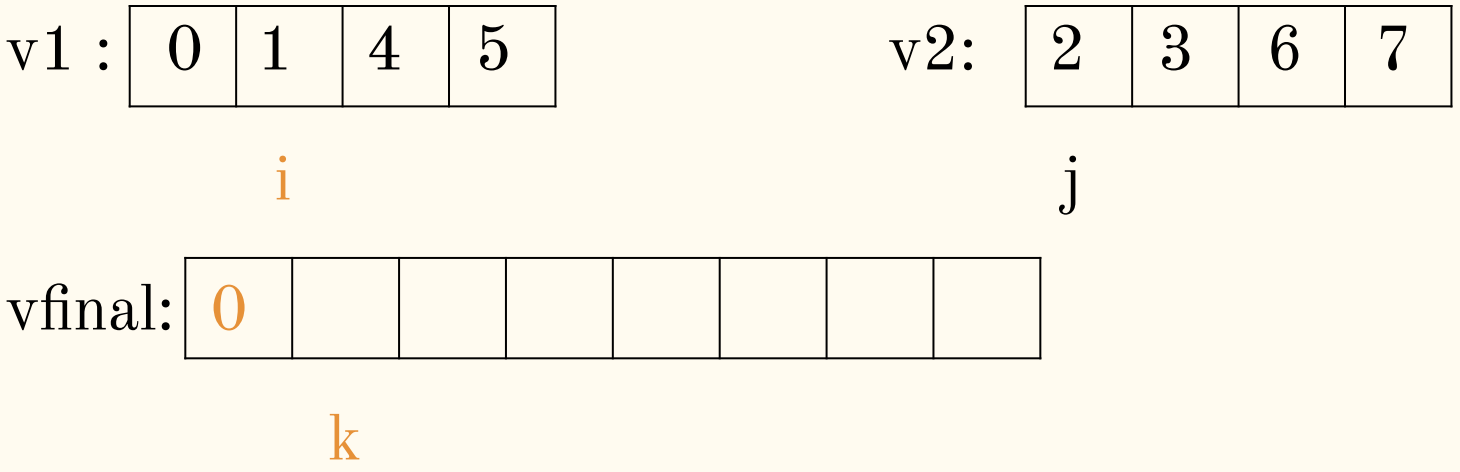
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
    - iterar a variável k
    - Se copiar v[i], iterar a variável i (mesmo para v[j])



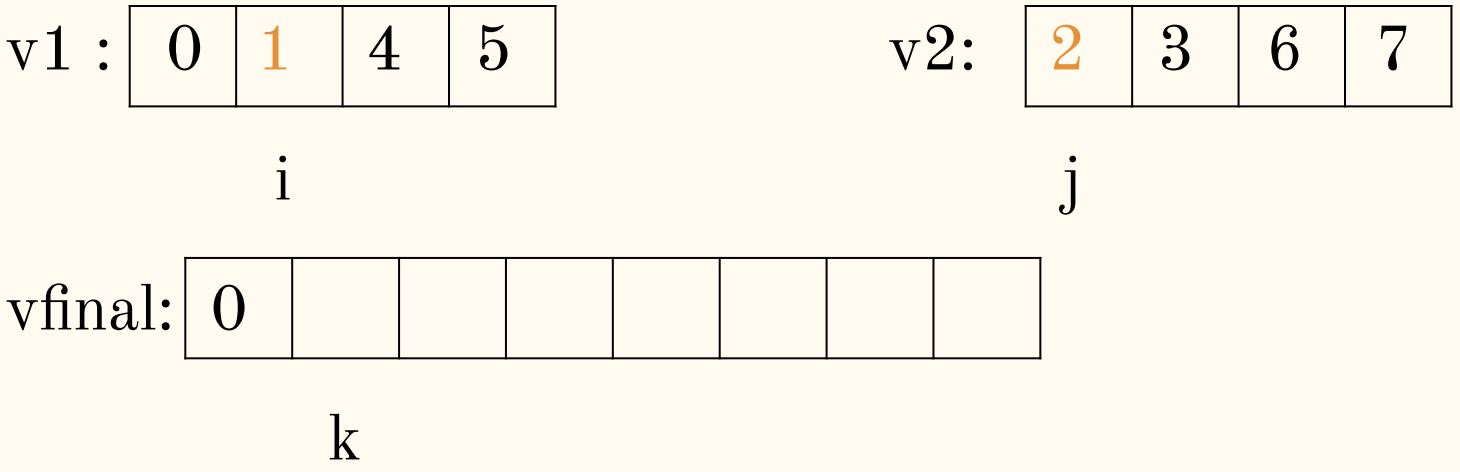
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



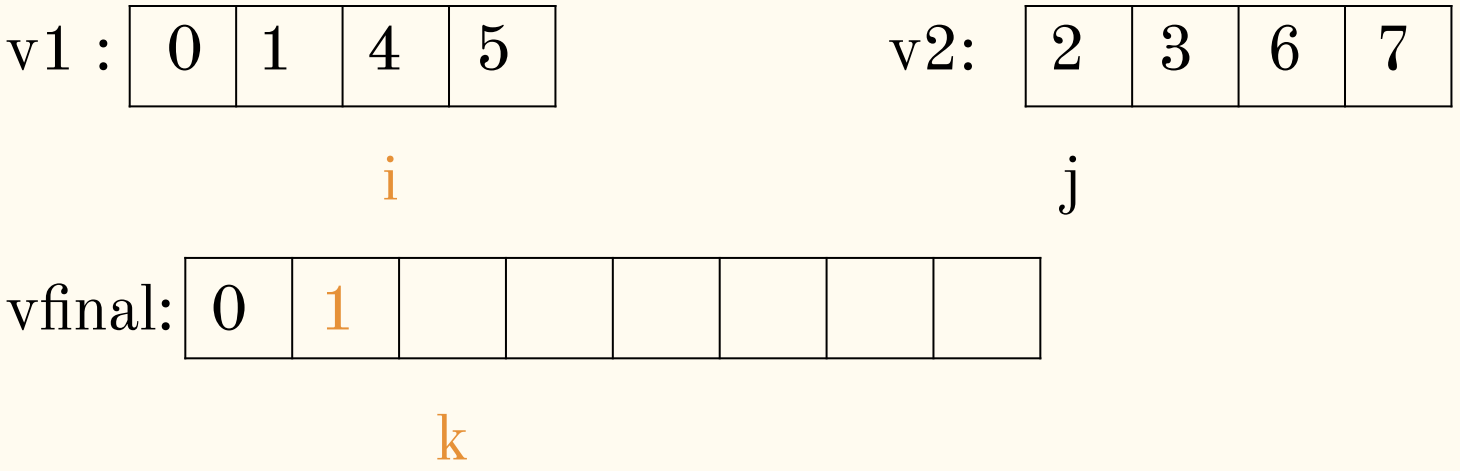
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



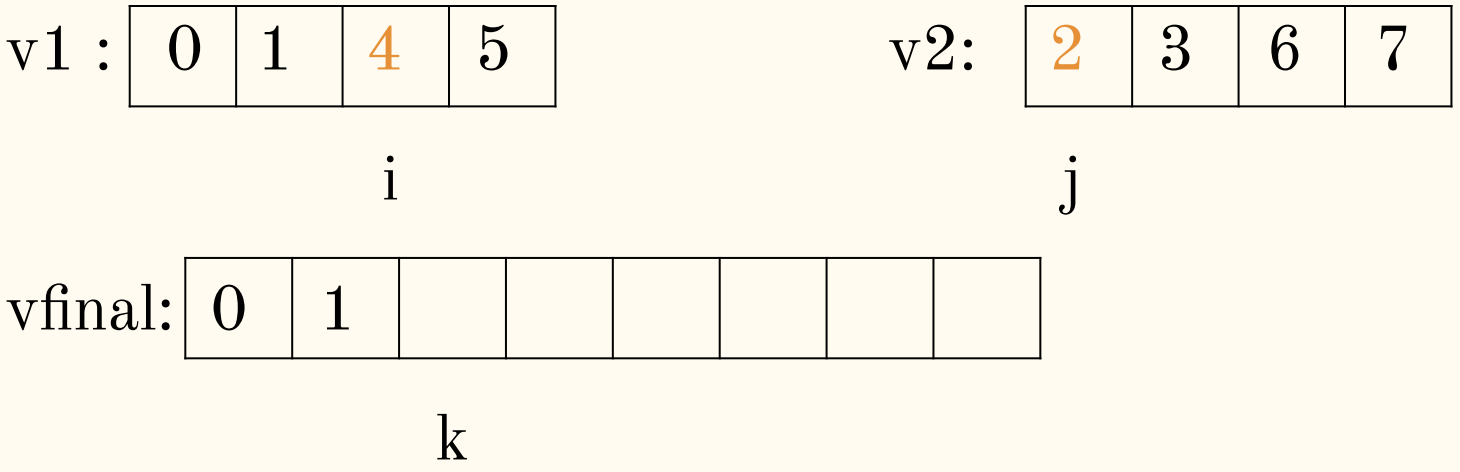
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



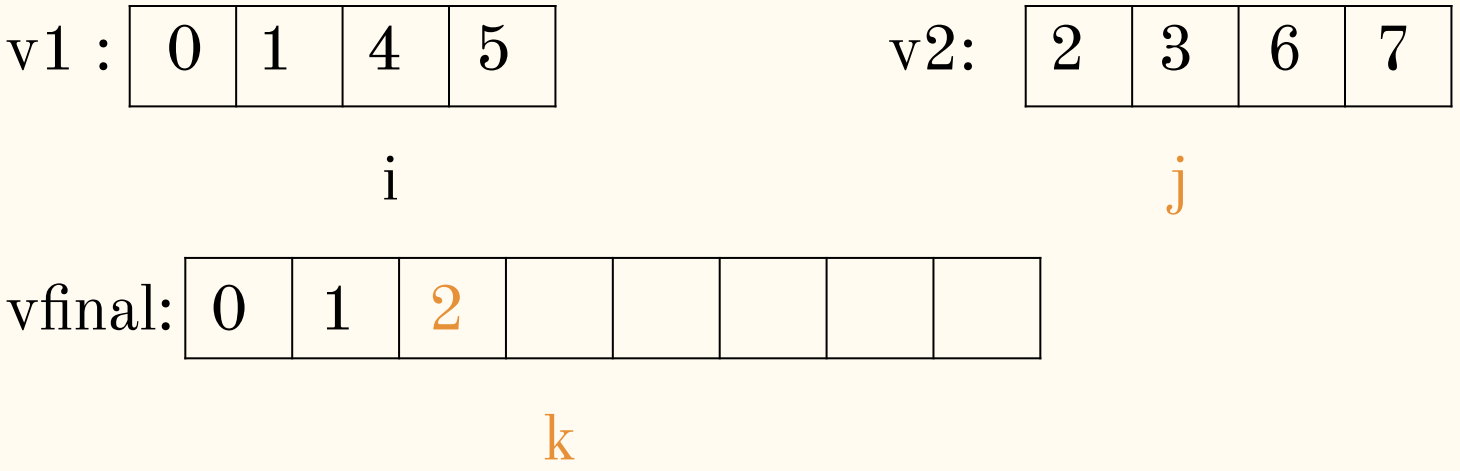
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



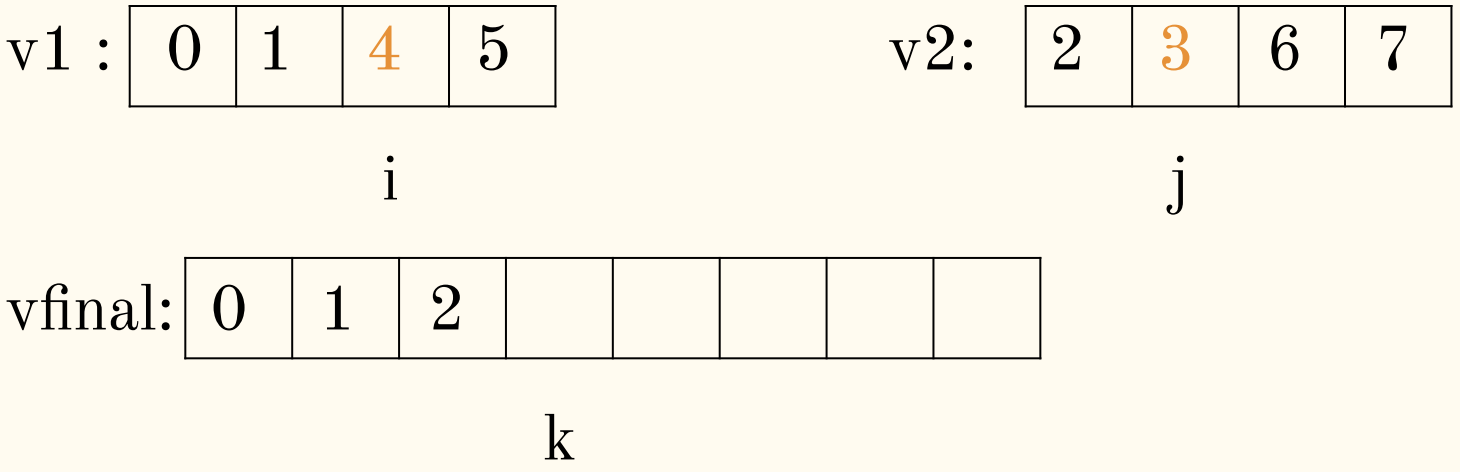
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



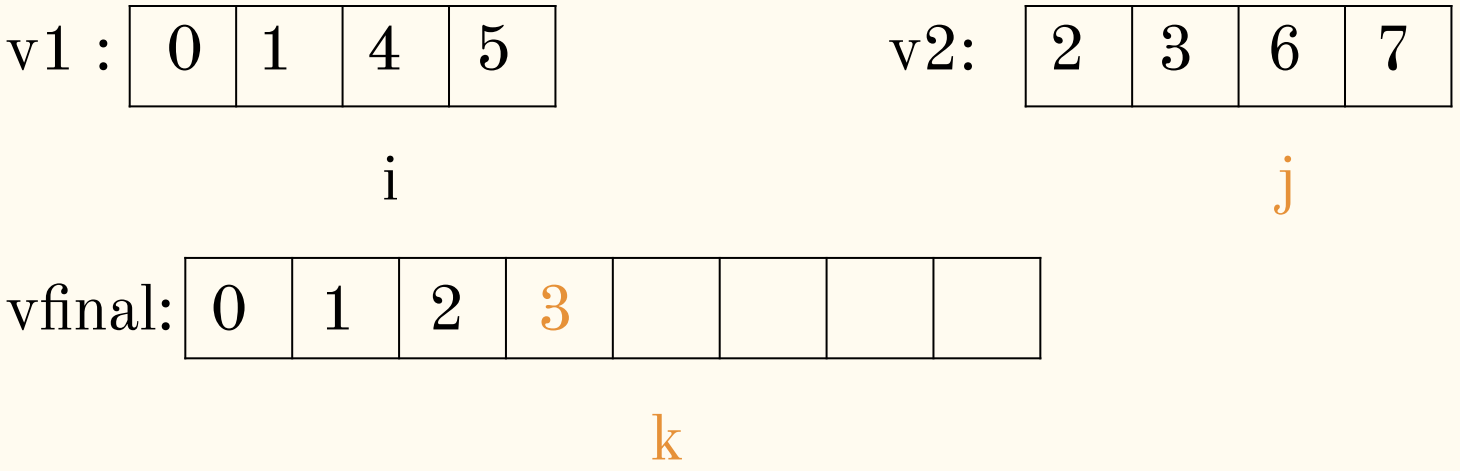
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



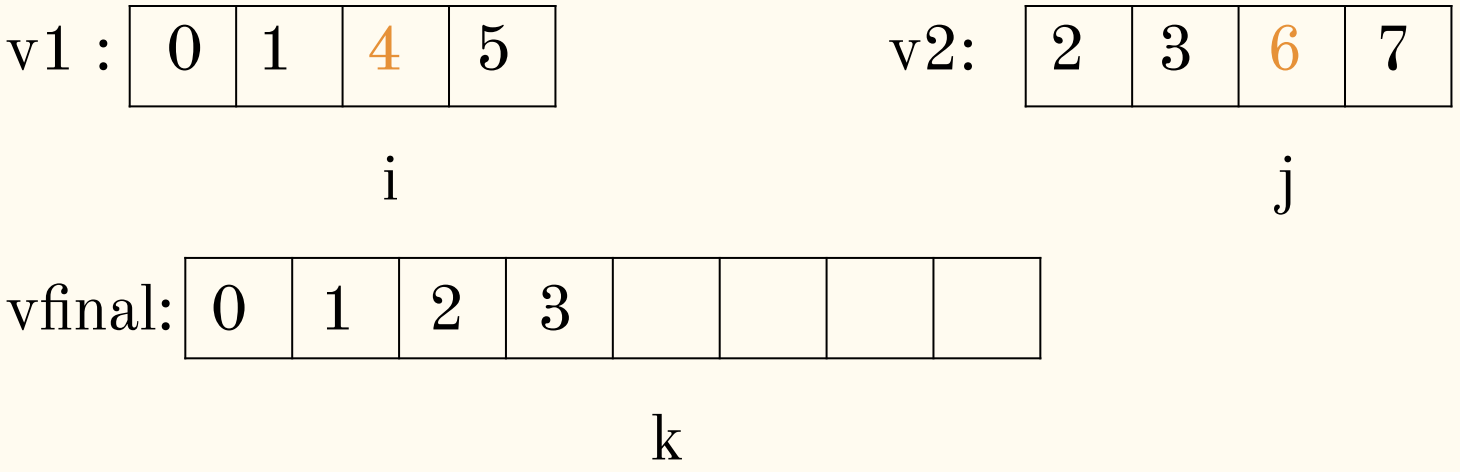
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



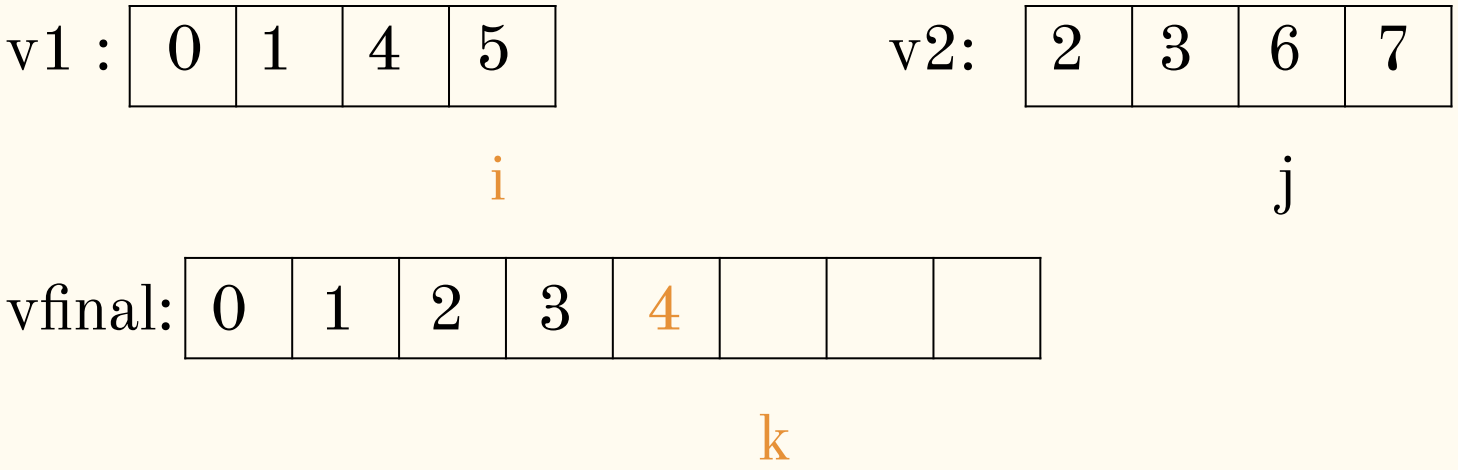
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



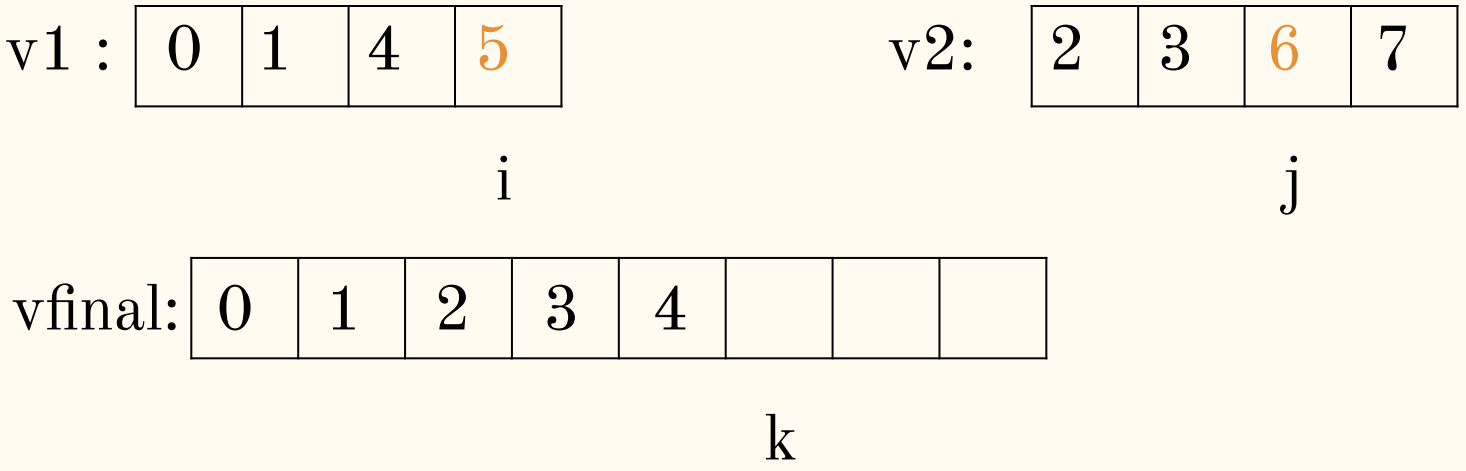
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



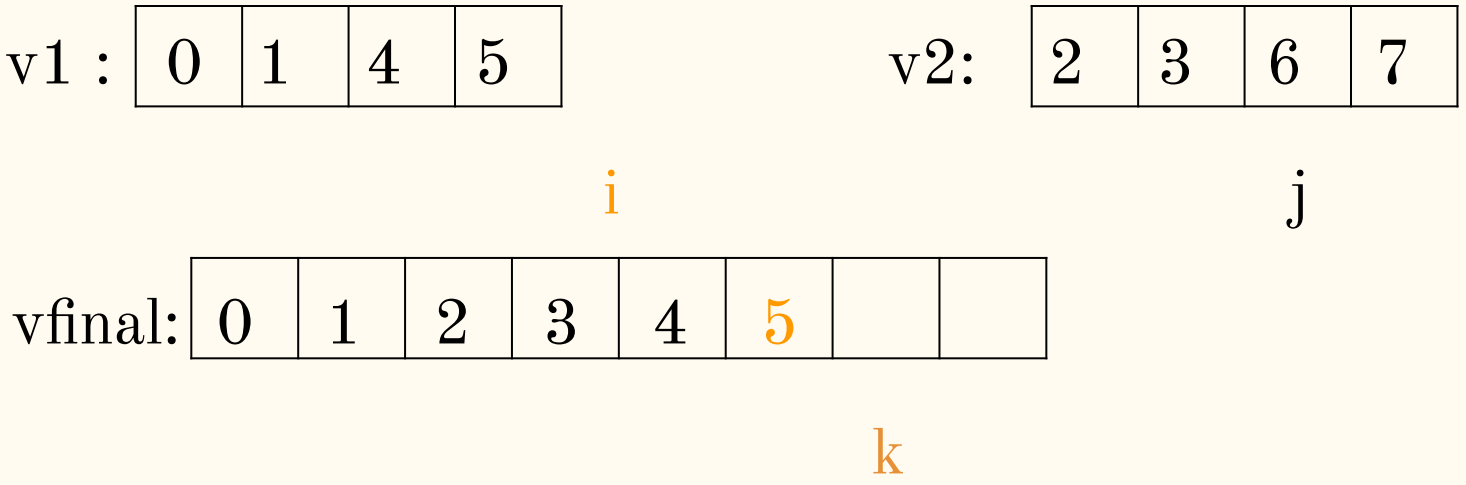
Merge Sort - Passo a passo do intercalar

- Comparar v1[i] com v2[j], o menor será copiado para vfinal
- Ações ao copiar:
  - iterar a variável k
  - Se copiar v[i], iterar a variável i (mesmo para v[j])



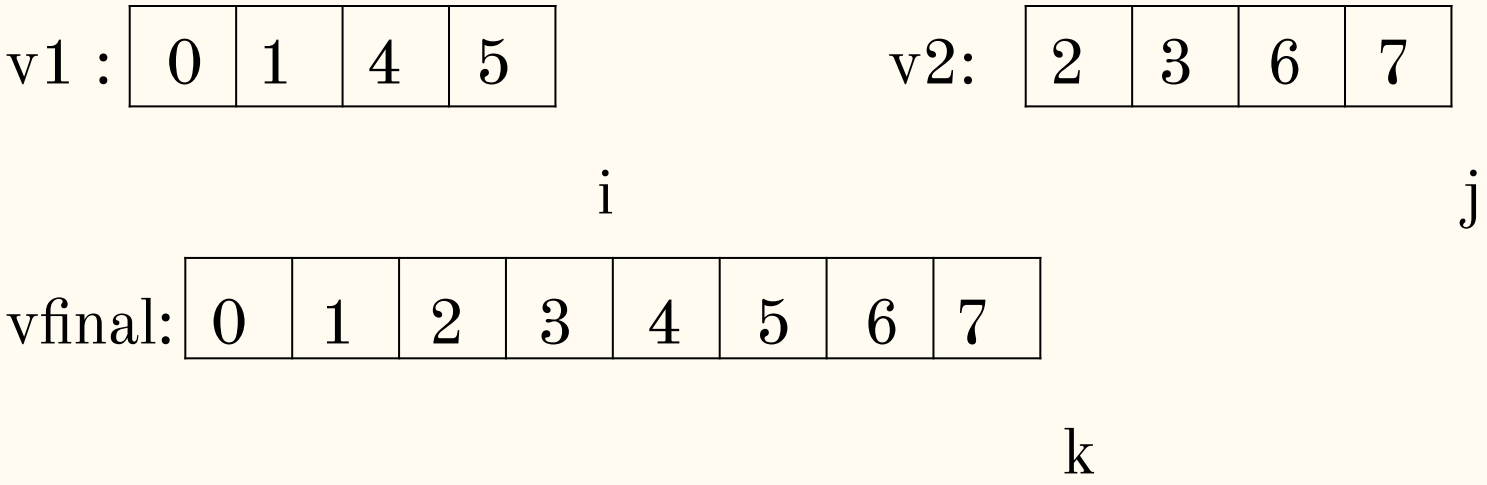
Merge Sort - Passo a passo do intercalar

- Chegando ao final de algum vetor, copia o restante do outro



Merge Sort - Passo a passo do intercalar

- Chegando ao final de algum vetor, copia o restante do outro



Merge Sort - Codificando...

```
int intercalar(int vetor[], int ini, int meio, int fim) {
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }

    // copia resto de cada subvetor
    while (i <= meio) auxiliar[k++] = vetor[i++];
    while (j <= fim)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

Merge Sort - Cod

corresponde ao vfinal

```
int intercalar(int vetor[..., int ini, int meio, int fim) {
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }

    // copia resto de cada subvetor
    while (i <= meio) auxiliar[k++] = vetor[i++];
    while (j <= fim)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

Merge Sort - Codificando...

```
int intercalar(int vetor[], int ini, int mei
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }

    // copia resto de cada subvetor
    while (i <= meio) auxiliar[k++] = vetor[i++];
    while (j <= fim)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

índices para percorrer todos os vetores

Merge Sort - Codificando...

```
int intercalar(int vetor[], int ini, int meio
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }

    // copia resto de cada subvetor
    while (i <= meio) auxiliar[k++] = vetor[i++];
    while (j <= fim)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

v1 e v2 são partes do mesmo vetor v

Merge Sort - Codificando...

```
int intercalar(int vetor[], int ini, int meio
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }

    de cada subvetor
    io) auxiliar[k++] = vetor[i++];
    m)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

v1 e v2 são partes do mesmo vetor v

v1 é vetor[i] com i variando de 0 até meio

Merge Sort - Codificando...

```
int intercalar(int vetor[], int ini, int meio
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }

    de cada subvetor
    io) auxiliar[k++] = vetor[i++];
    m)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

v1 e v2 são partes do mesmo vetor v

v2 é vetor[j] com j variando de meio+1 até fim

v1 é vetor[i] com i variando de 0 até meio

Merge Sort - Codificando...

```
int intercalar(int vetor[], int ini, int meio, int fim) {
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }

    // copia resto de cada subvetor
    while (i <= meio) auxiliar[k++] = vetor[i++];
    while (j <= fim)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

Merge Sort - Complexidade

```
int intercalar(int vetor[], int ini, int meio, int fim) {
    int auxiliar[MAX];
    int i = ini, j = meio + 1, k = 0;

    // intercala
    while(i <= meio && j <= fim) {
        if (vetor[i] <= vetor[j])
            auxiliar[k++] = vetor[i++];
        else
            auxiliar[k++] = vetor[j++];
    }

    // copia resto de cada subvetor
    while (i <= meio) auxiliar[k++] = vetor[i++];
    while (j <= fim)  auxiliar[k++] = vetor[j++];

    // copia de auxiliar para vetor
    for (i = ini, k=0; i <= fim; i++,k++)
        vetor[i] = auxiliar[k];
}
```

com i percorre primeira metade do vetor (n/2)

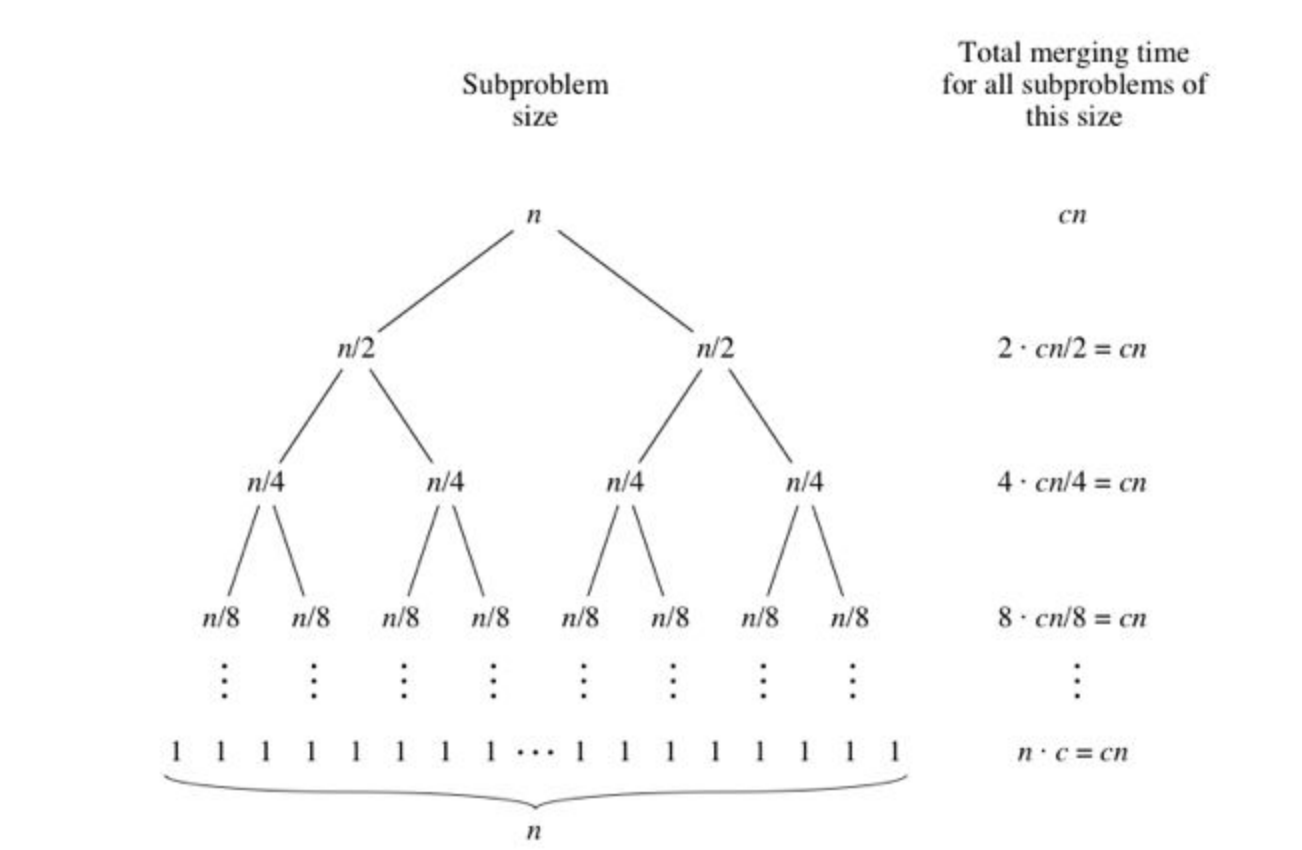
com j percorre segunda metade (n/2)

se nao chegou ao fim em i ou j,

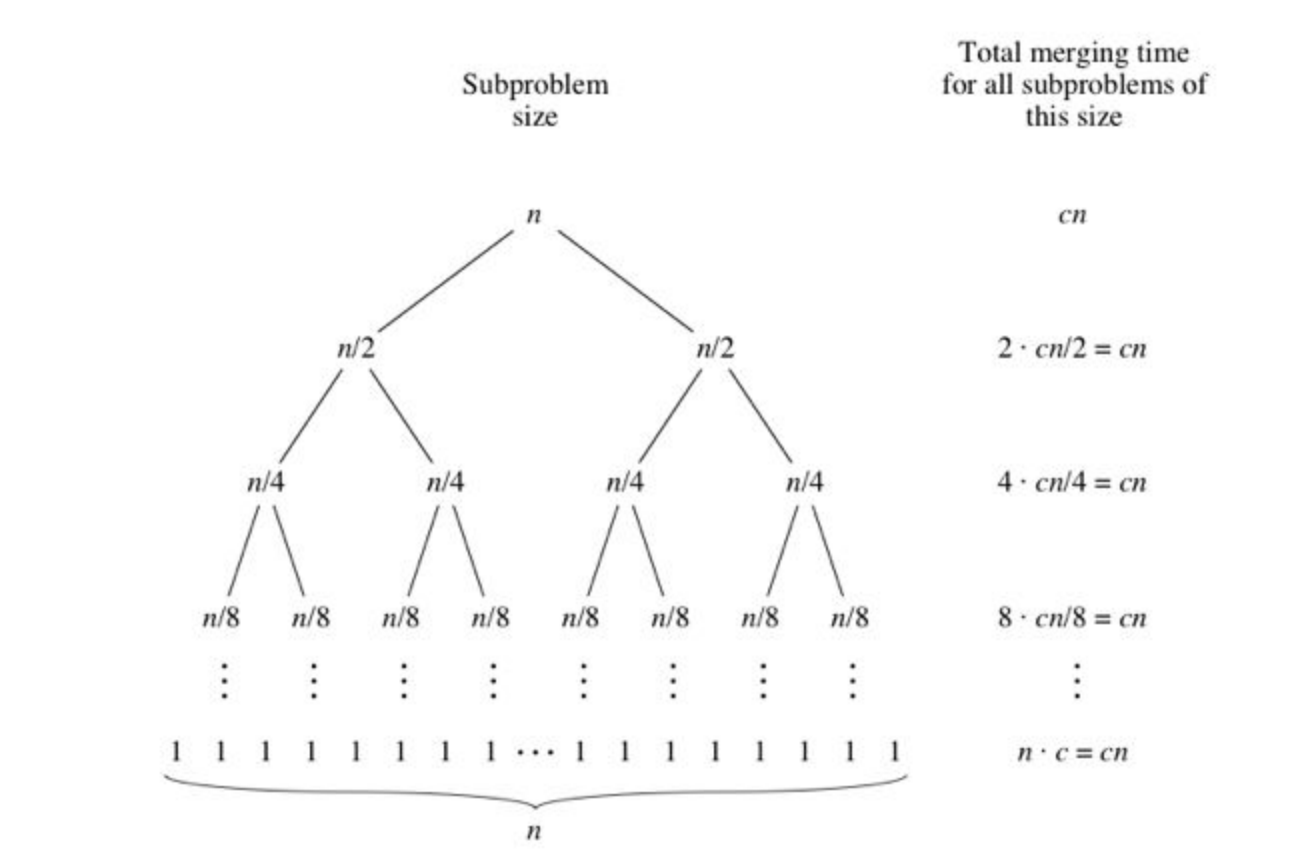
percorre o restante do vetor

custo n para copiar

Merge Sort - Complexidade

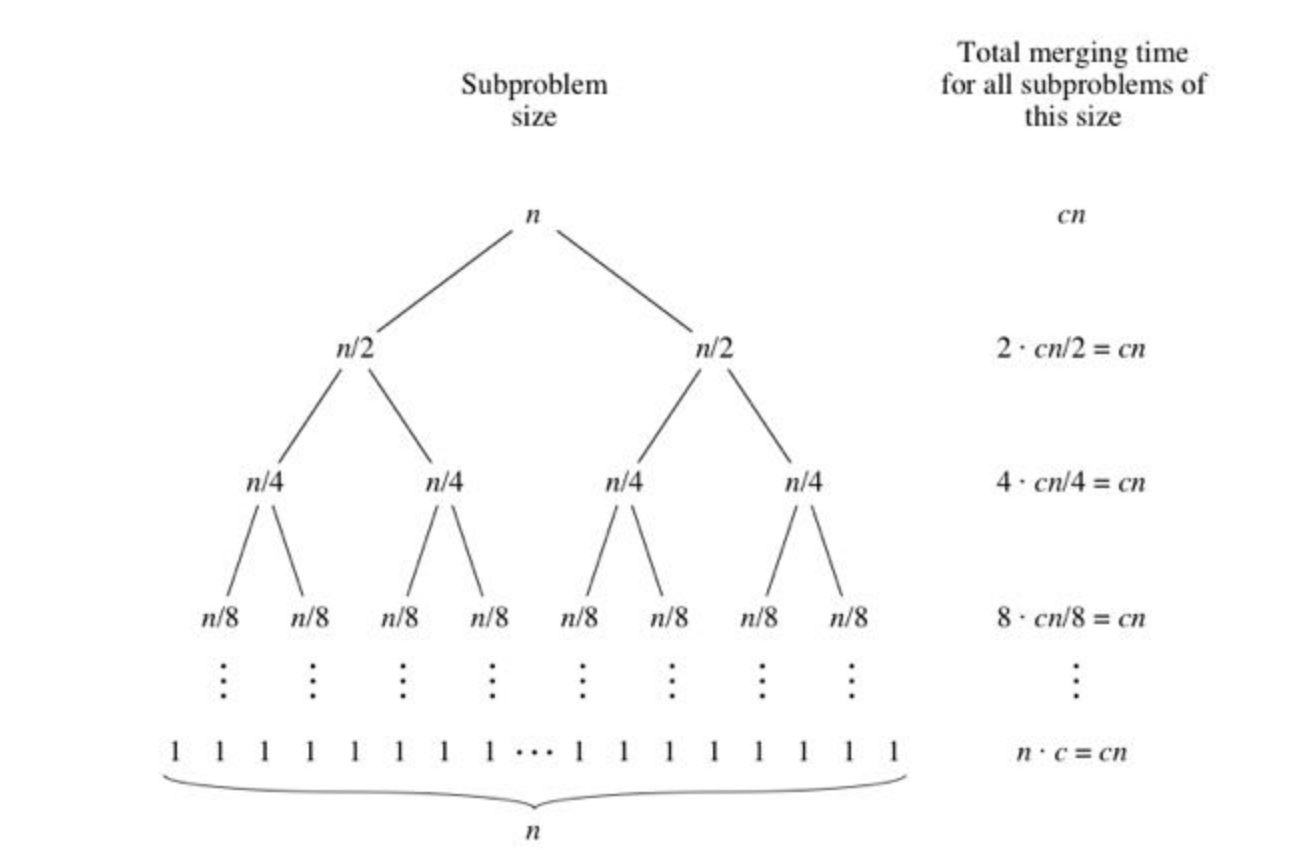


Merge Sort - Complexidade



Cada iteração custa O(n), porém quantas iterações, no máximo faz um merge-sort?

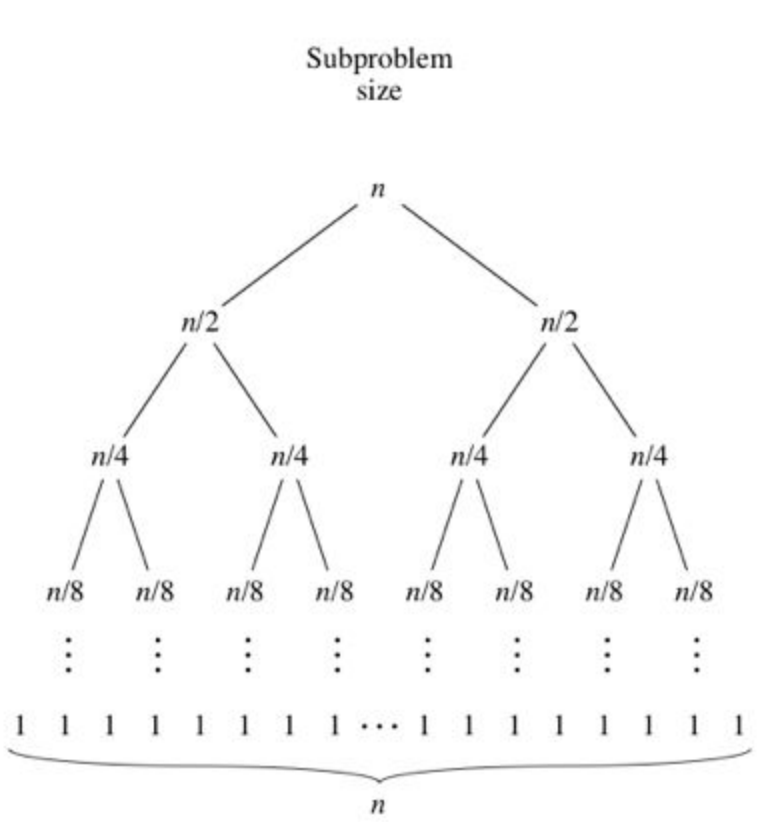
Merge Sort - Complexidade



Cada iteração custa O(n), porém quantas iterações, no máximo faz um merge-sort?

Qual a altura máxima de uma árvore cheia?

Merge Sort - Complexidade

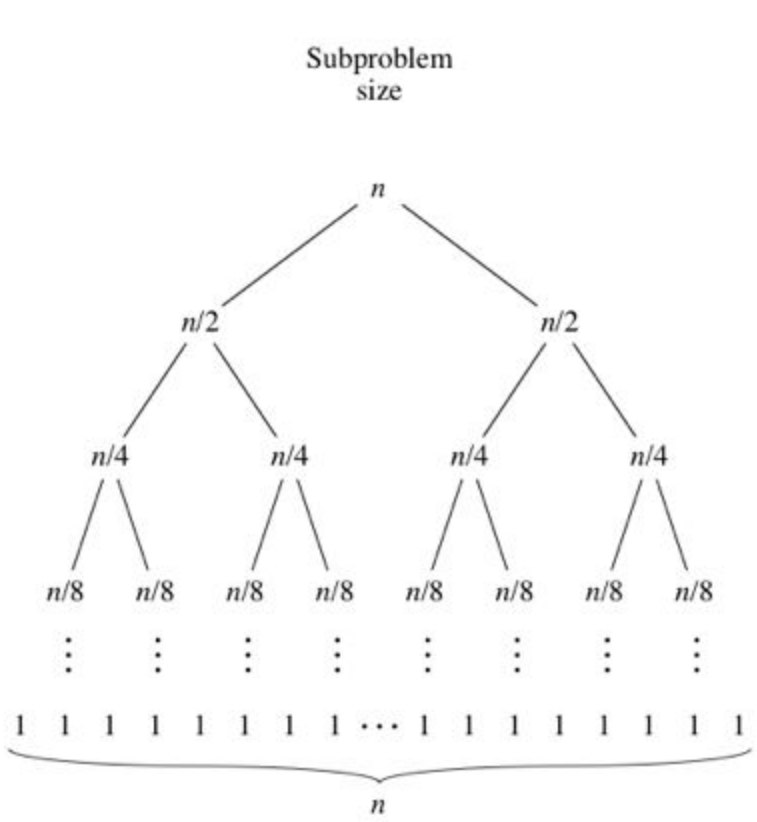


nº de nós	iteração	Regra nº de nós
1	0	
2	1	
4	2	
8	3	
n	?	

Cada iteração custa O(n), porém quantas iterações, no máximo faz um merge-sort?

Qual a altura máxima de uma árvore cheia?

Merge Sort - Complexidade

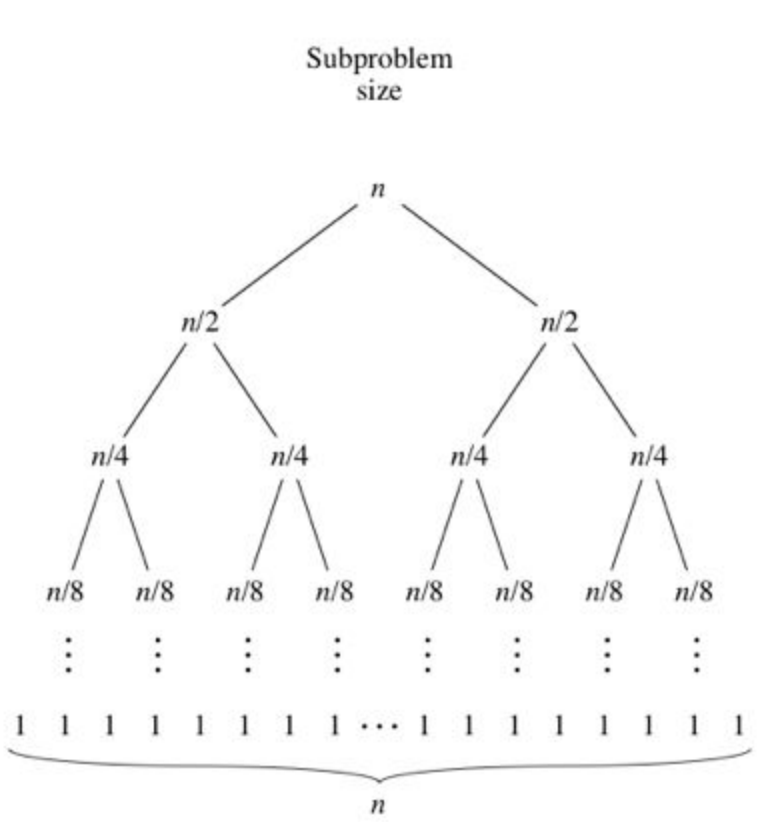


nº de nós	iteração	Regra nº de nós
1	0	2^0
2	1	2^1
4	2	2^2
8	3	2^3
n	?	2^i

Cada iteração custa O(n), porém quantas iterações, no máximo faz um merge-sort?

Qual a altura máxima de uma árvore cheia?

Merge Sort - Complexidade



nº de nós	iteração	Regra nº de nós
1	0	2^0
2	1	2^1
4	2	2^2
8	3	2^3
n	?	2^i

Cada iteração custa O(n), porém quantas iterações, no máximo faz um merge-sort?

Qual a altura máxima de uma árvore cheia? i=log(n) **Merge sort=O(nlogn)**

Roteiro

Introdução

Métodos iterativos
- Ordenação por trocas
- Método das bolhas
- Ordenação por inserção

Métodos recursivos - Divisão e Conquista
- Merge Sort
- **Quick Sort**

Complexidade algorítmica dos algoritmos de classificação

Quick Sort - Ideia

- Recebemos um vetor de tamanho n

- Dividimos em dois subvetores
  - Escolhemos 1 elemento para ser o pivô (o último)
  - vetor_pequenos: elementos menores ou iguais a pivô
  - vetor_grandes: elementos maiores a pivô
  - função `particionar`

- Ordenamos cada subvetor

Quick Sort - Condificando ideia...

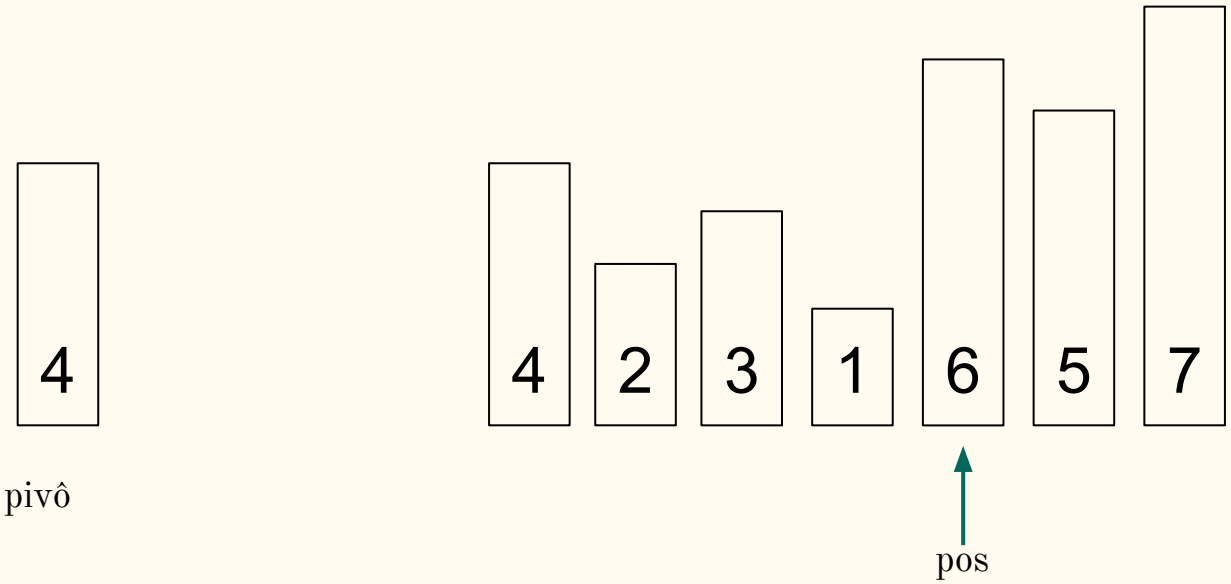
```
void quick_sort(int vetor[], int ini, int fim) {
        int pos;

        if (ini < fim){
            pos = particionar(vetor, ini, fim);

            quick_sort(vetor, ini, pos - 1);
            quick_sort(vetor, pos, fim);
        }
}
```

Quick sort - Ideia

Como funciona o particionamento:

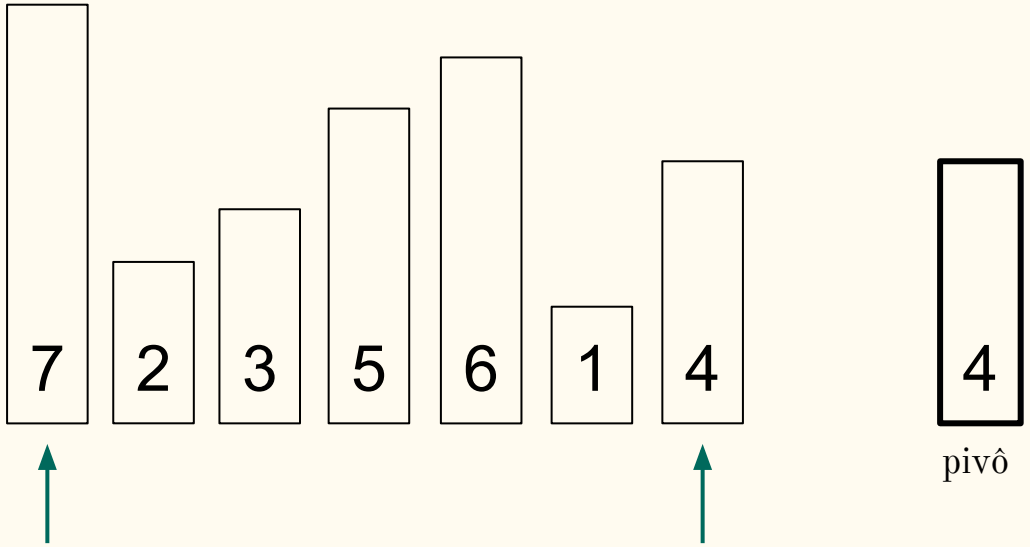
```
int particionar(int vetor[], int ini, int fim)
```

a primeira parte do vetor contém elementos “pequenos”
a segunda parte do vetor contém elementos “grandes”
retorna um inteiro (pos) que indica a 1ª posição do vetor “grandes”



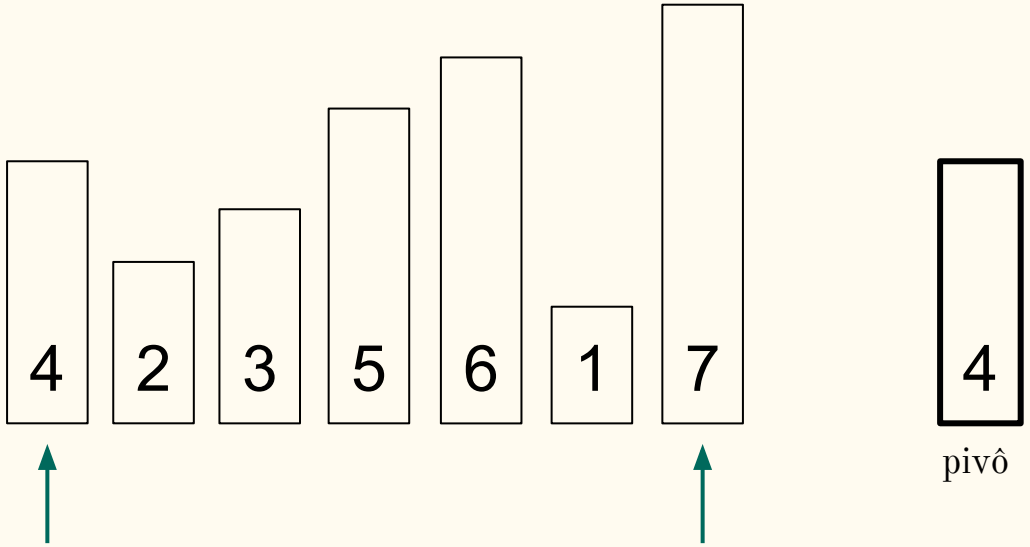
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca



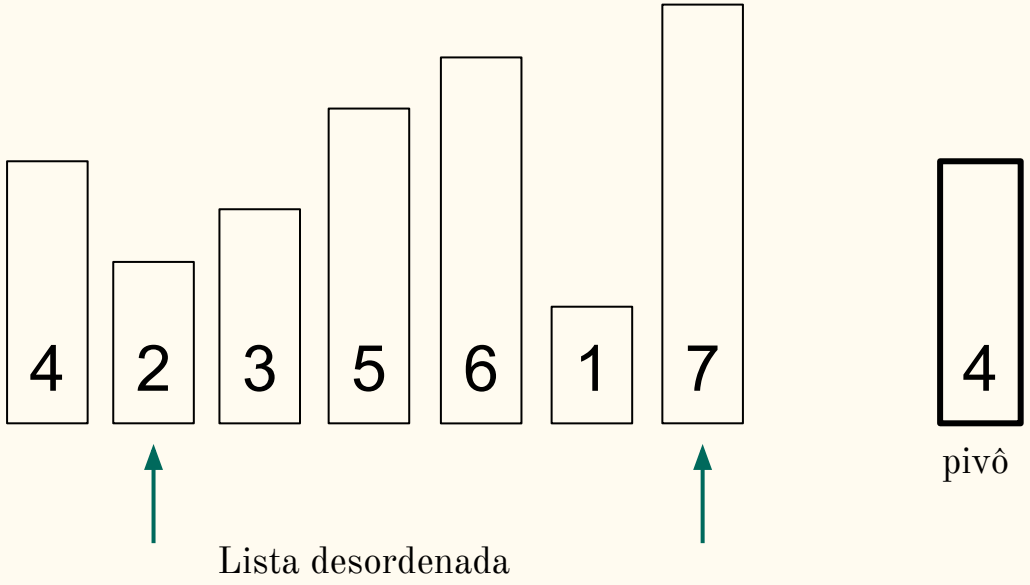
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca



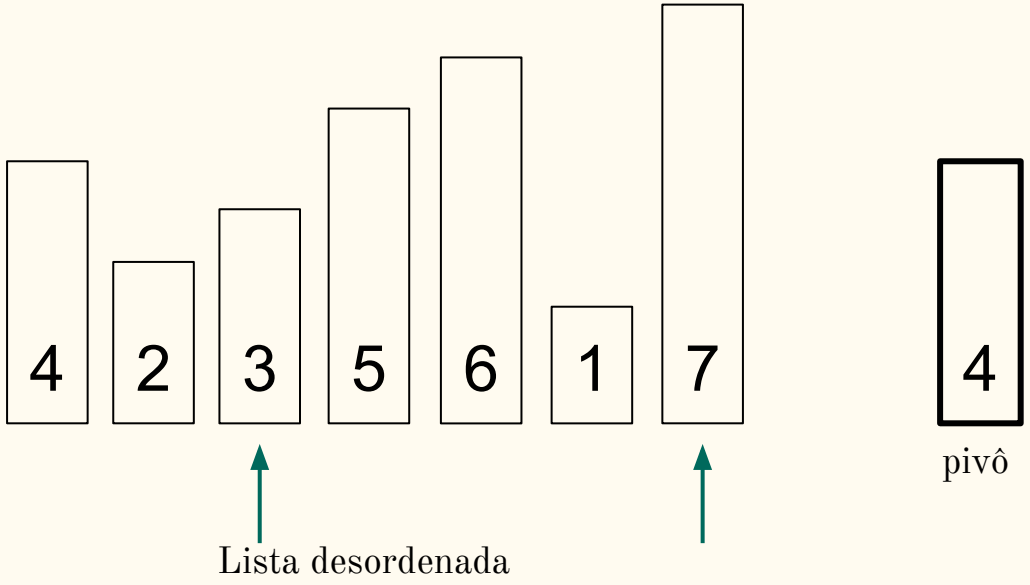
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca



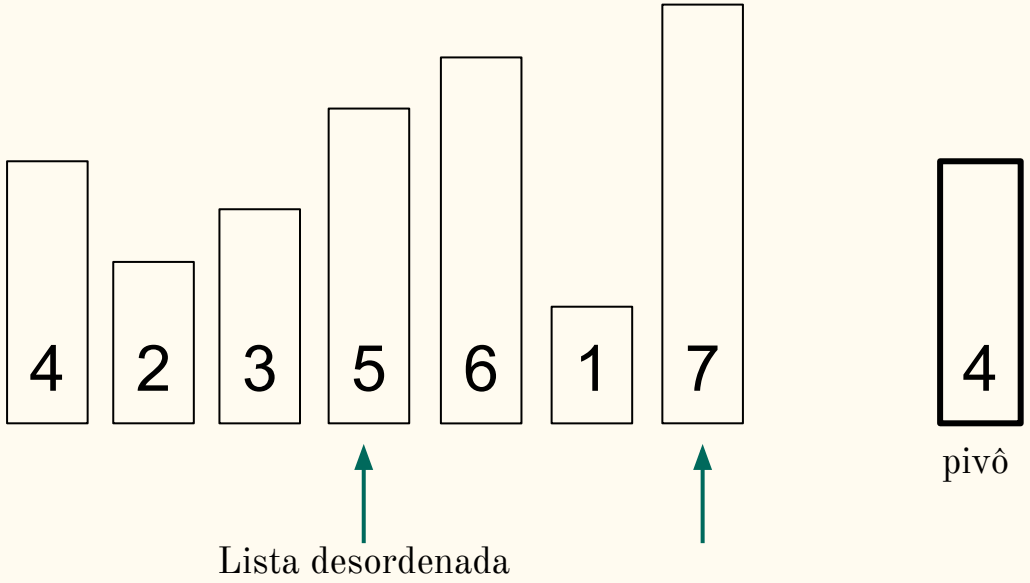
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca



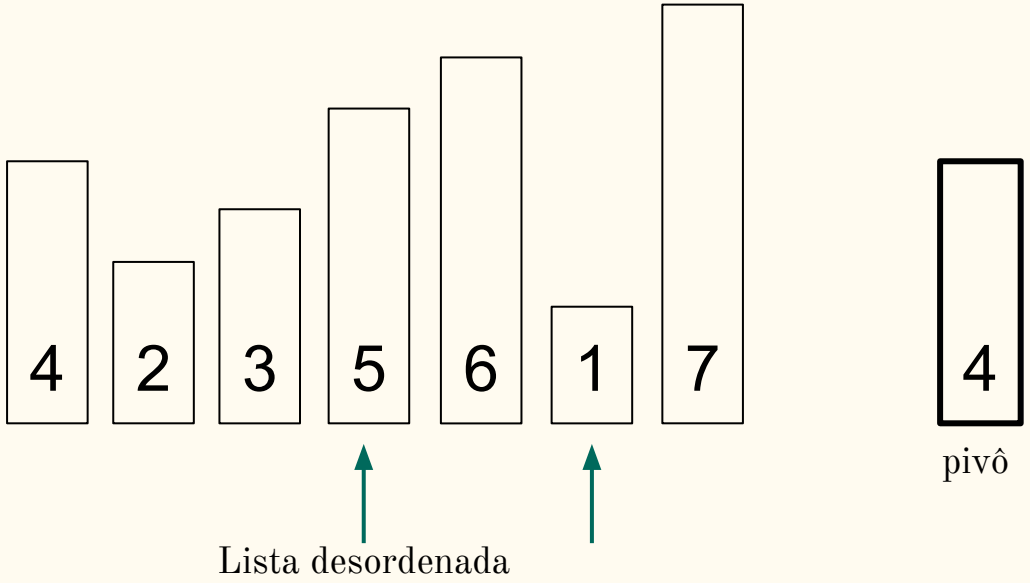
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca



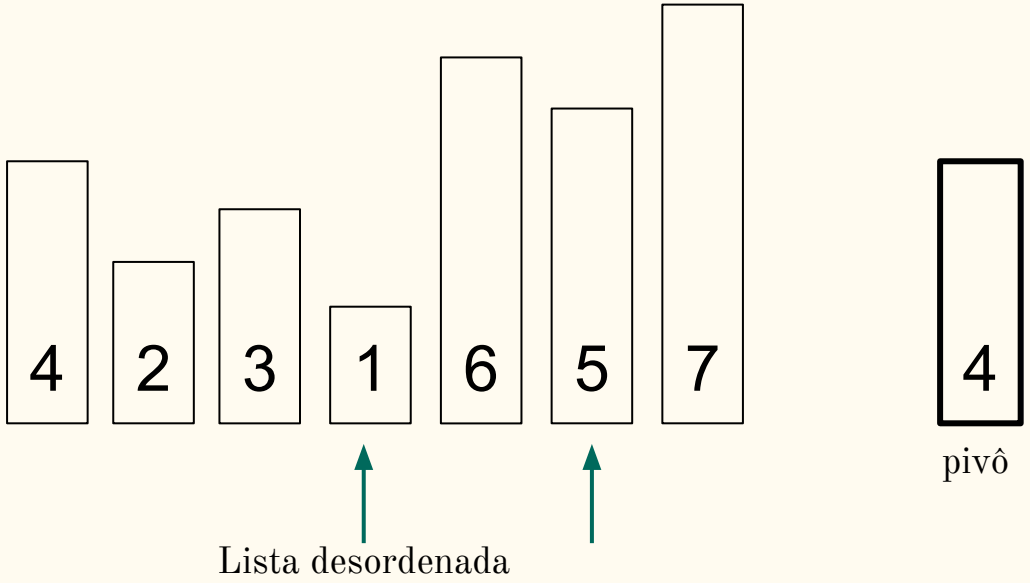
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca



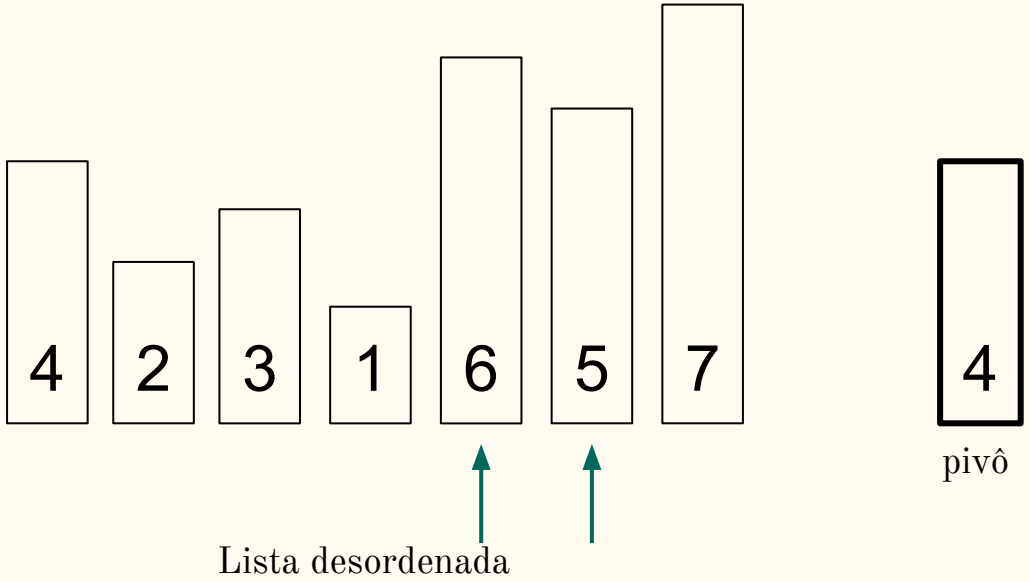
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca



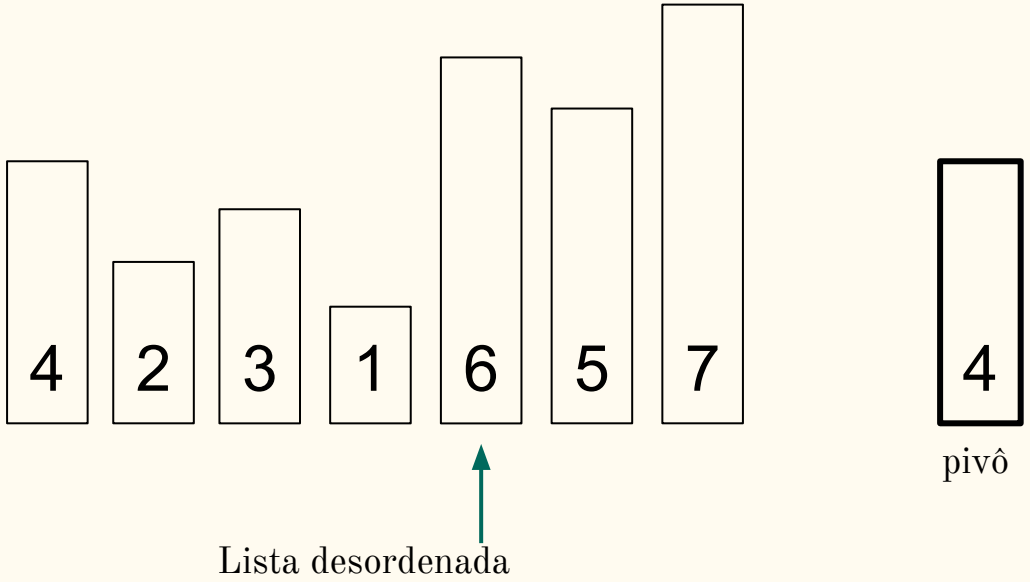
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca



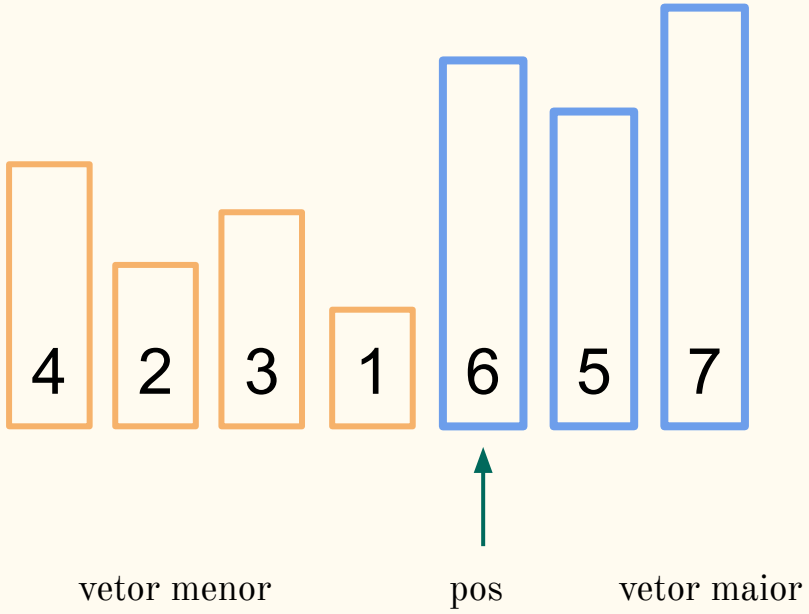
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração

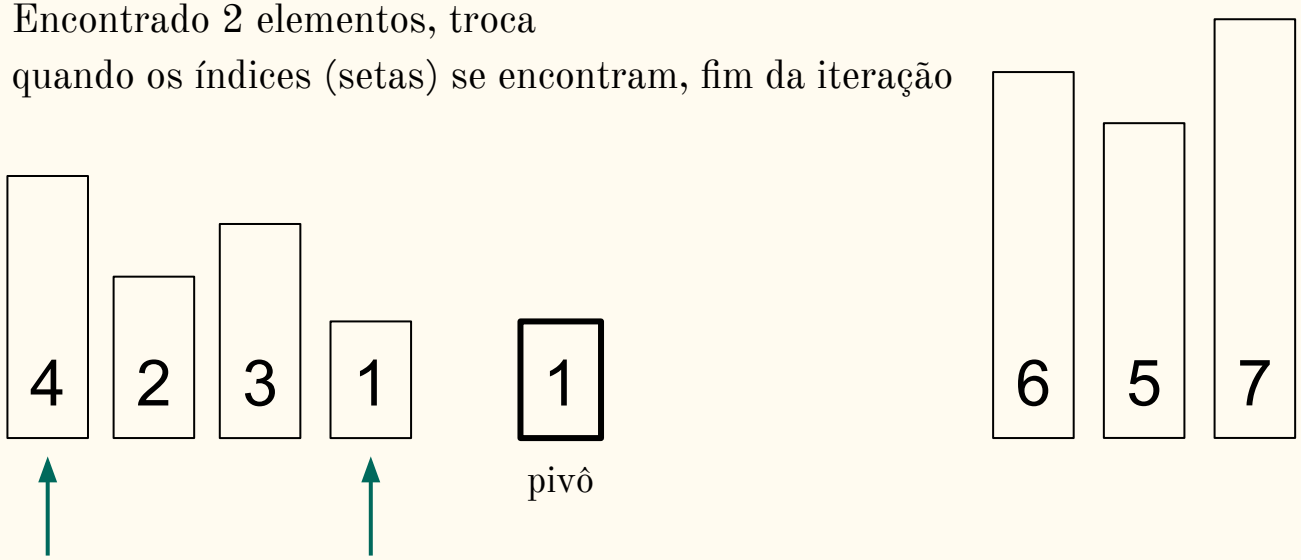


Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração

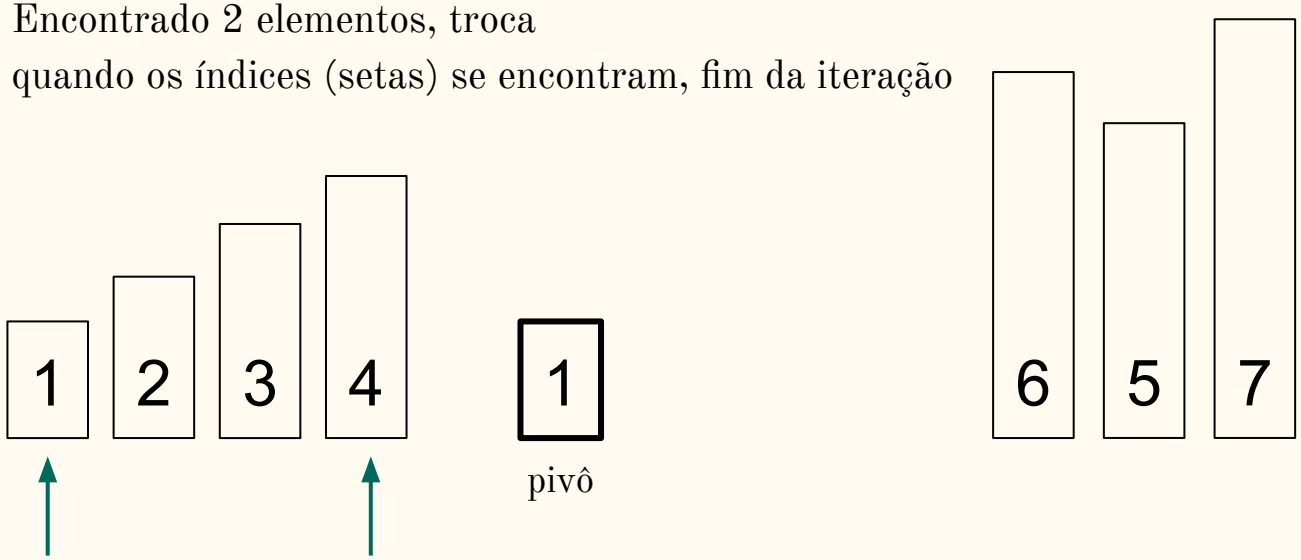
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



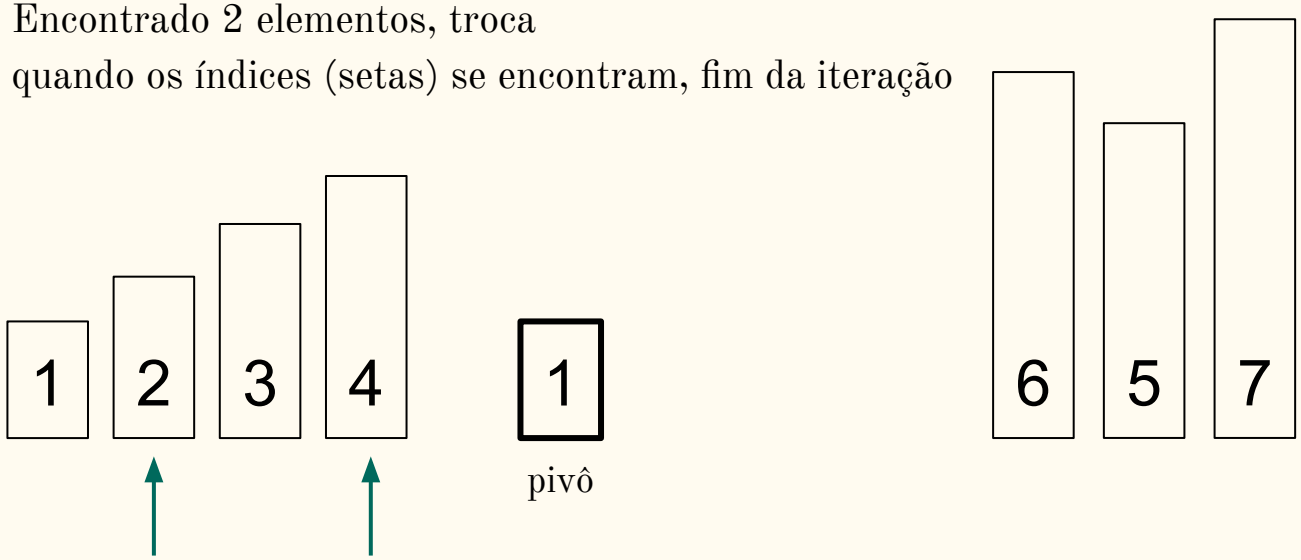
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



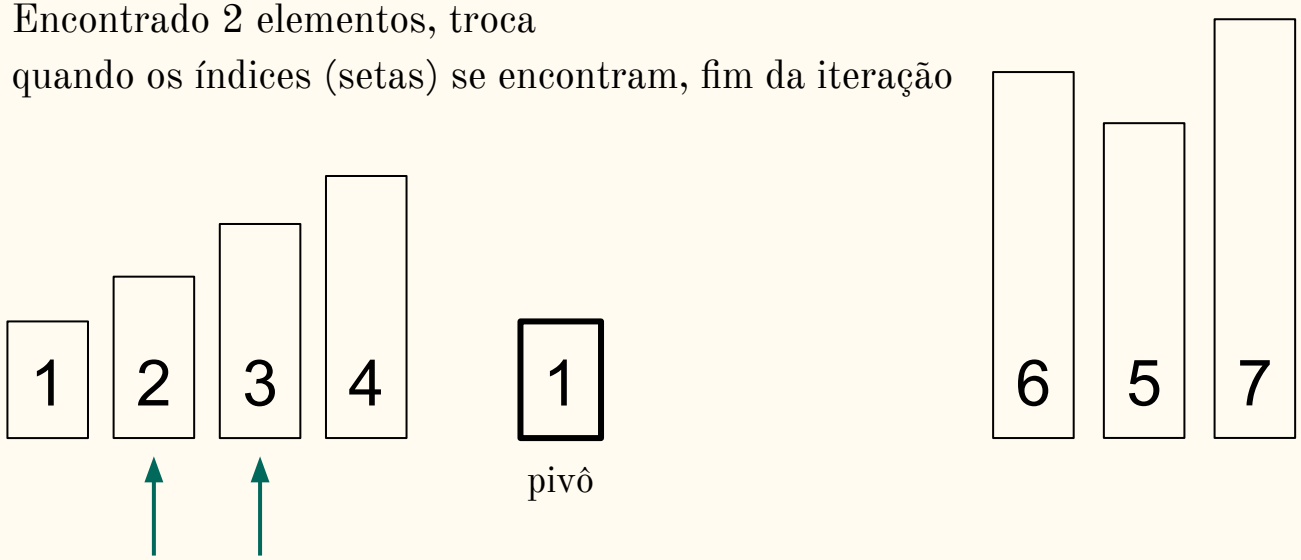
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



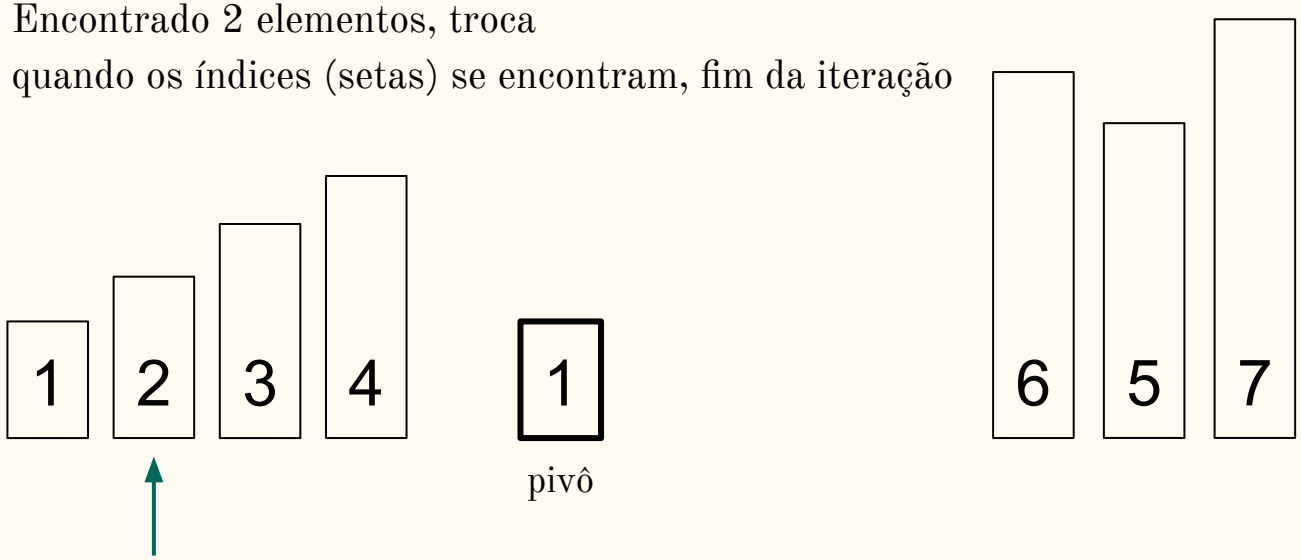
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



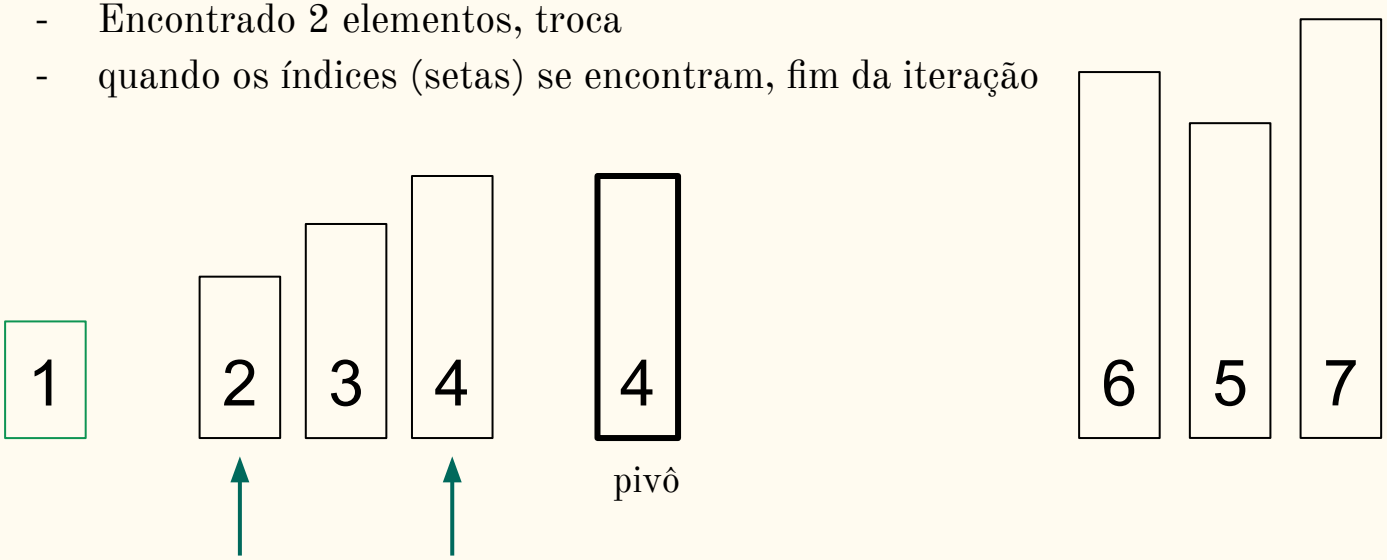
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



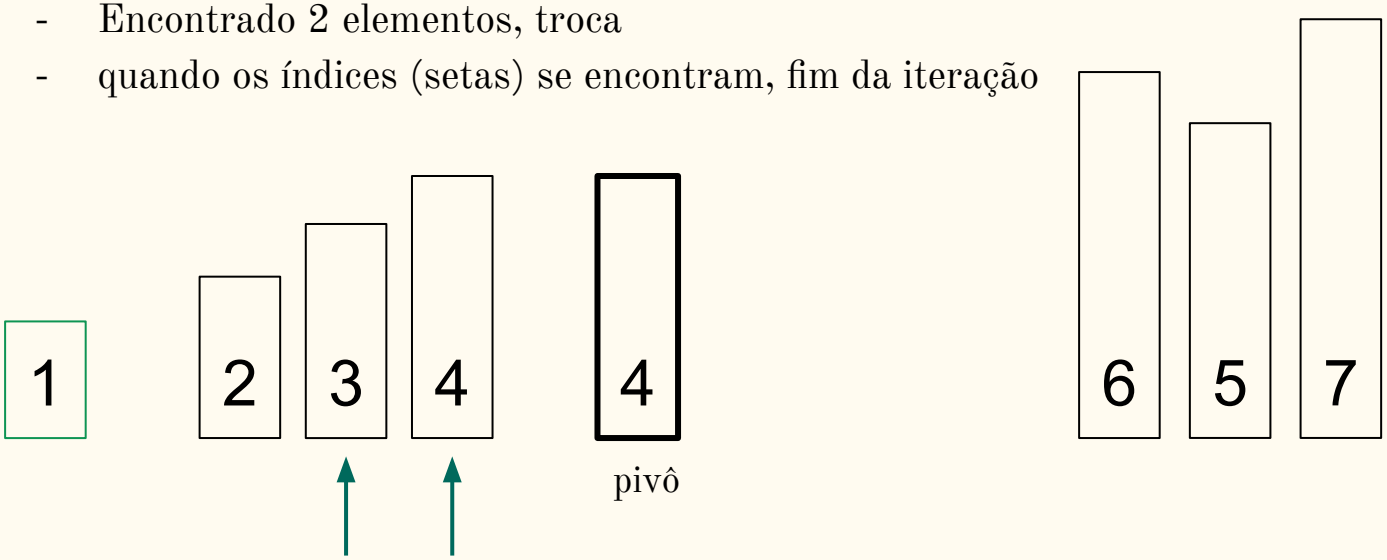
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



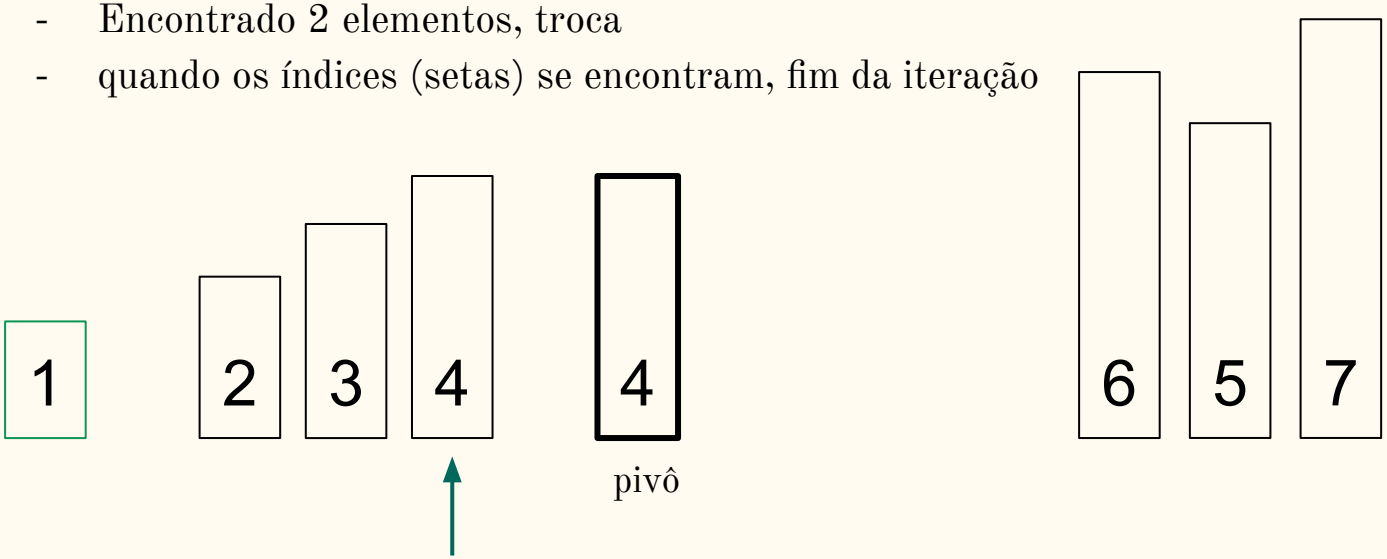
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



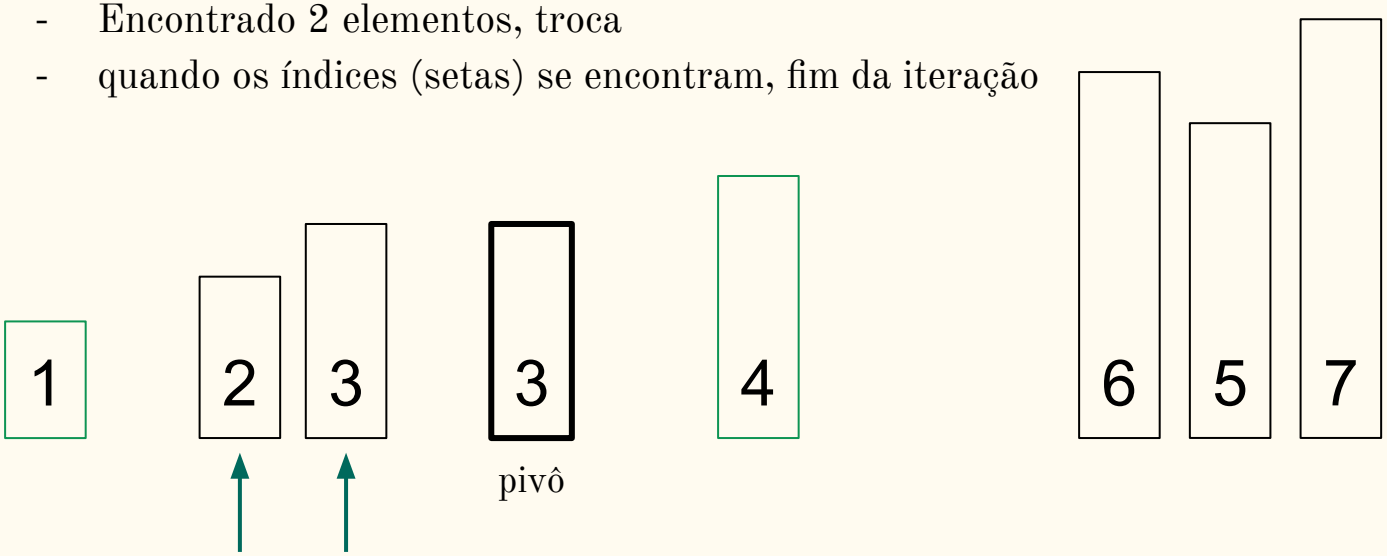
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



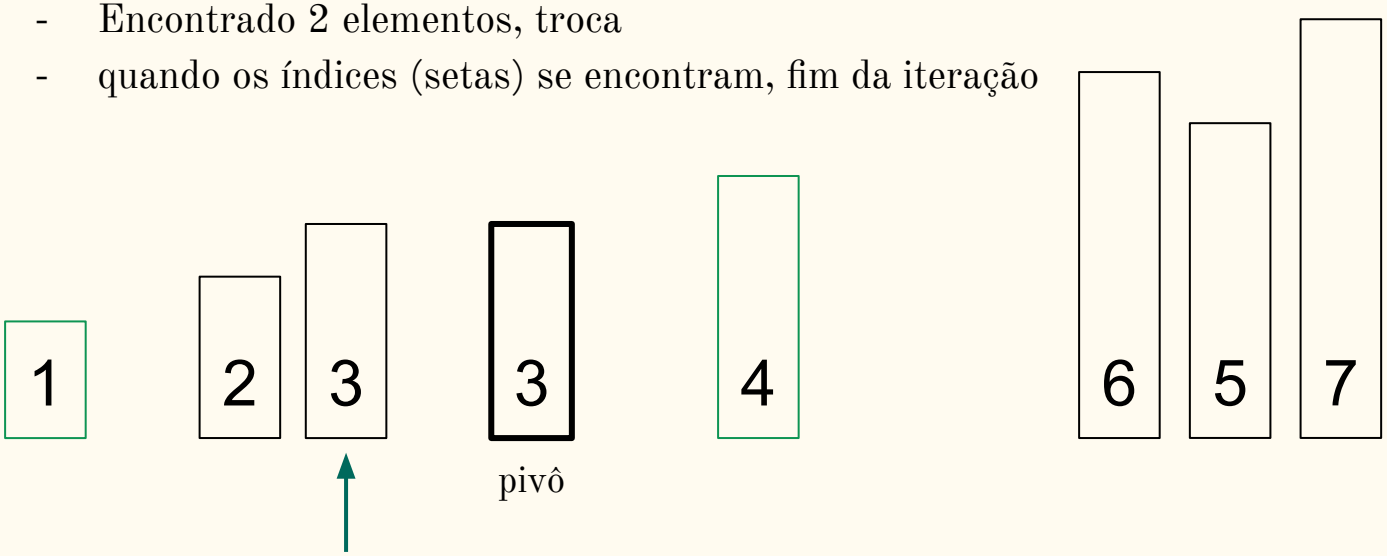
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



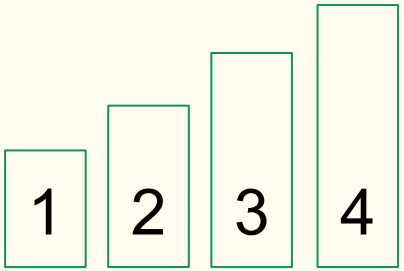
Quick sort - Passo a passo

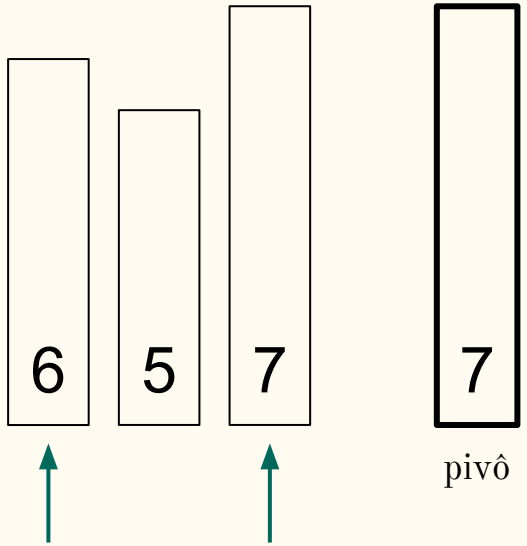
- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



Quick sort - Passo a passo

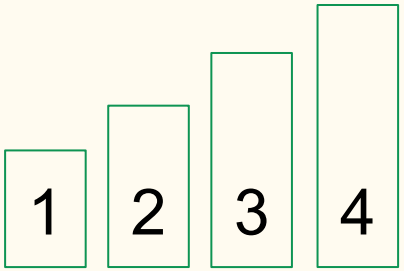
- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração

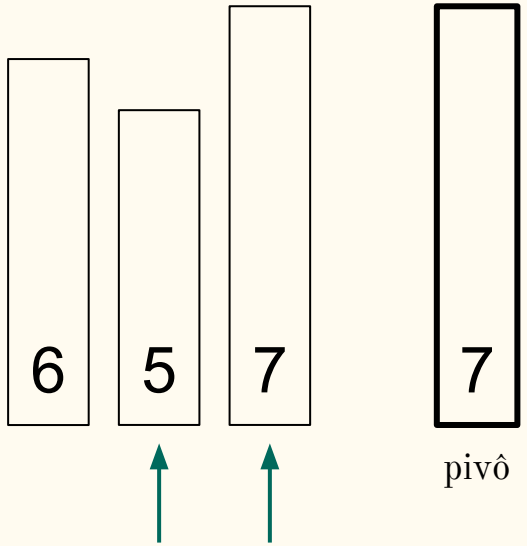




Quick sort - Passo a passo

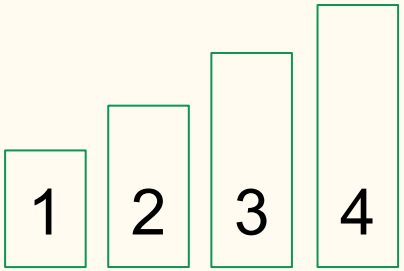
- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração

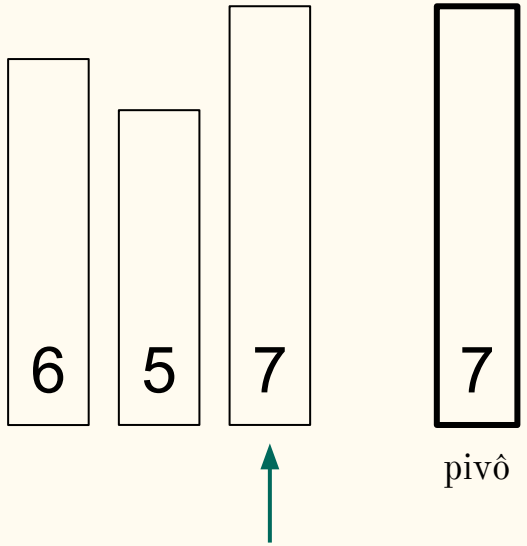




Quick sort - Passo a passo

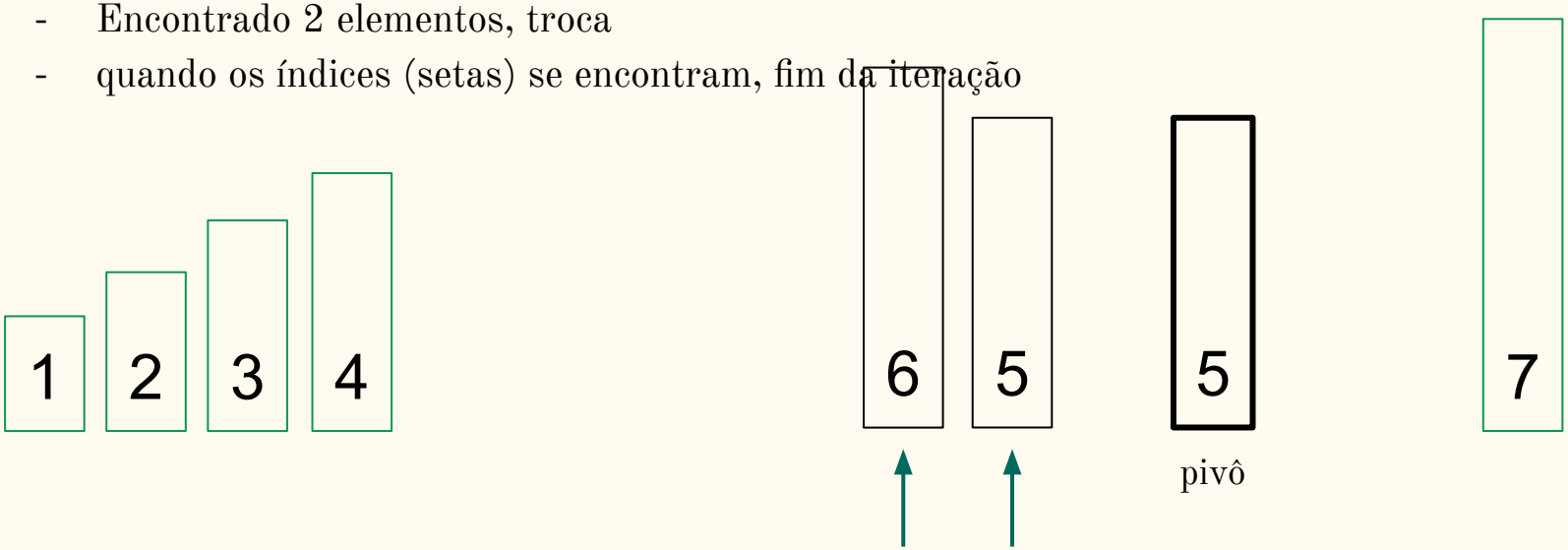
- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração





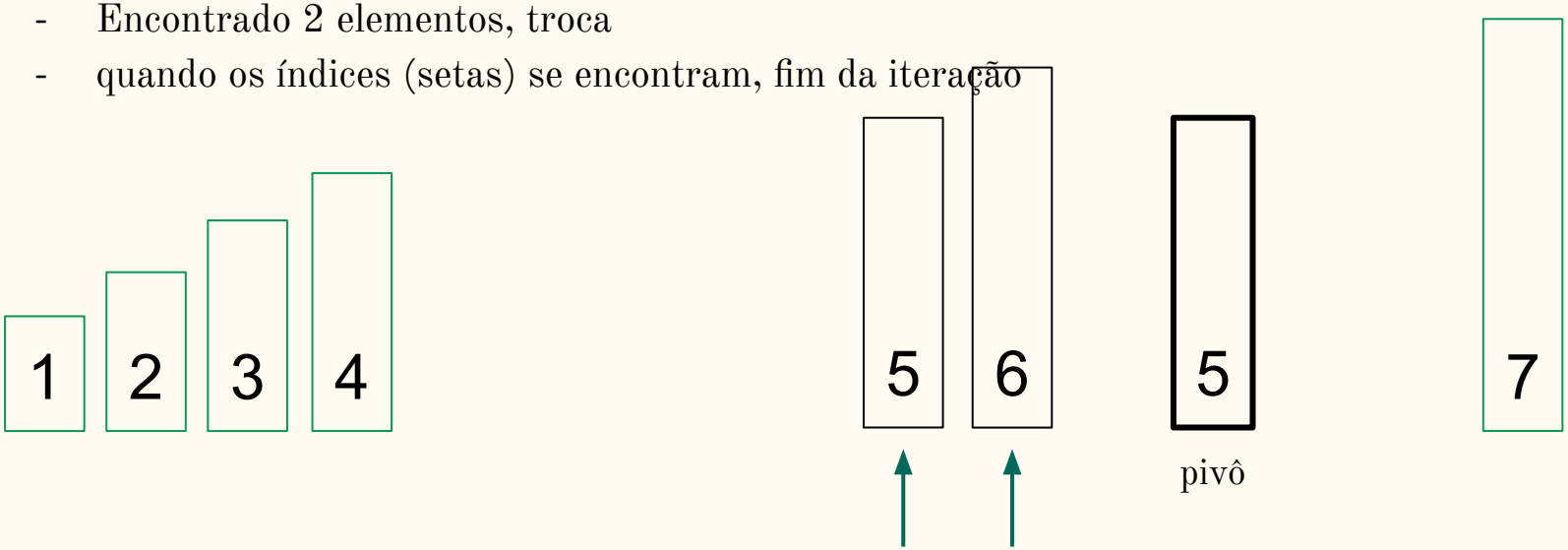
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



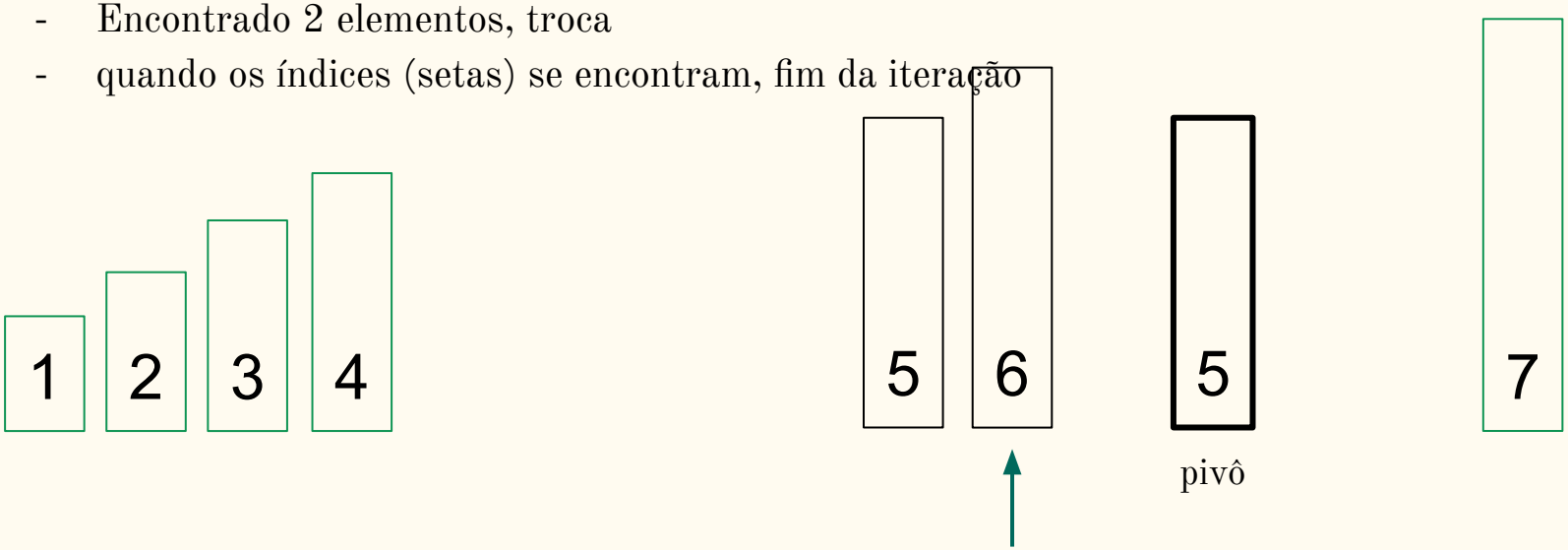
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



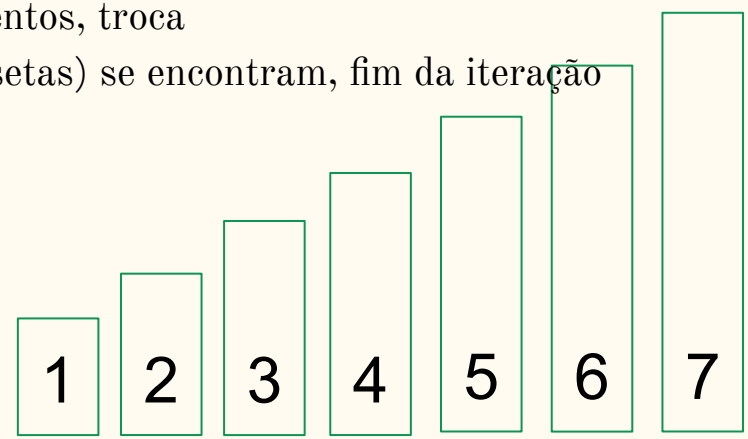
Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



Quick sort - Passo a passo

- Percorrendo da esquerda para direita (vetor menor)
  - se encontra elemento fora do lugar, pára
- Percorrendo da direita para esquerda (vetor maior)
  - se encontra elemento fora do lugar, pára
- Encontrado 2 elementos, troca
- quando os índices (setas) se encontram, fim da iteração



Quick sort - Codificando...

```
int particionar(int vetor[], int ini, int fim) {
    int pivo;

    pivo = vetor[fim];

    while (ini < fim) {
        while (ini < fim && vetor[ini] <= pivo)
            ini++;

        while (ini < fim && vetor[fim] > pivo)
            fim--;

        troca(&vetor[ini], &vetor[fim]);
    }

    return ini; // ini é a posição do primeiro elemento grande
}
```

Quick sort - Codificando...

Neste algoritmo não existe partição pela metade “certinha” como no merge.

Geralmente, nós não sabemos quantos números são $<=$ pivô e quantos são $>$pivô.

Existe um cálculo estatístico para verificar esse valor na média

Quick sort - Codificando...

Neste algoritmo não existe partição pela metade “certinha” como no merge.

Geralmente, nós não sabemos quantos números são $<=$ pivô e quantos são $>$pivô.

Existe um cálculo estatístico para verificar esse valor na média

Quick sort - Complexidade

Generalizando, seja k o número de elemento de um subvetor:

T(n)=custoParticionamento + T(k) + t(n-k)

Quick sort - Complexidade

Generalizando, seja k o número de elemento de um subvetor:

$T(n) = custoParticionamento + T(k) + t(n-k)$

O algoritmo de particionamento custa $O(n)$

$T(n) = O(n) + T(k) + t(n-k)$

Quick sort - Complexidade

$T(n)=O(n) + T(k) + t(n-k)$

Para o melhor caso:

Os subconjuntos tem tamanhos iguais:

> $T(n)=O(n) + T(n/2) + T(n/2) = O(nlgn)$ -> Constrói mesma árvove que o merge-sort

Quick sort - Complexidade

$T(n)=O(n) + T(k) + t(n-k)$

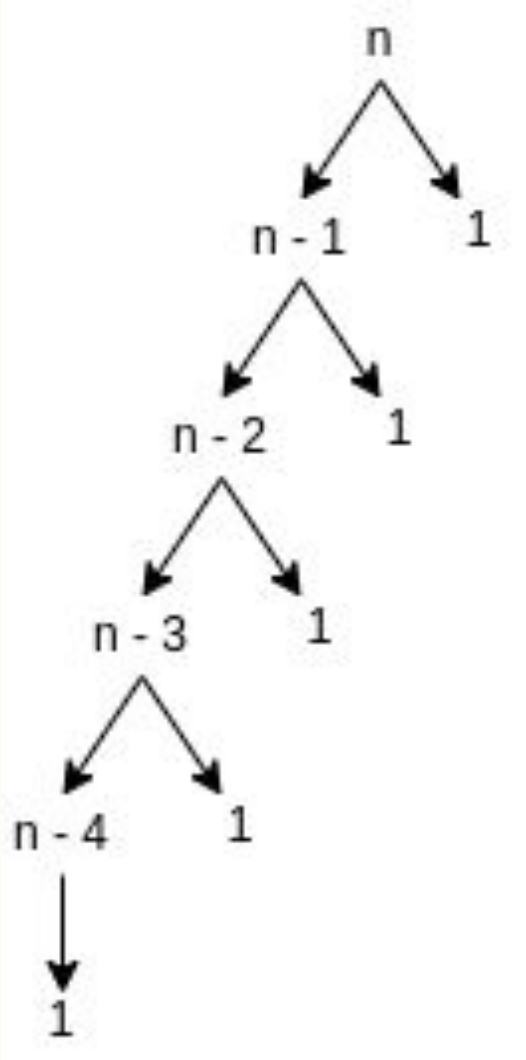
Para o pior caso:

Os subconjuntos têm tamanhos 1 e n-1:

$T(n)=O(n) + T(1) + T(n-1) = O(n^2)$ **por quê?**

Quick sort - Complexidade

Pior caso



Recapitulando...

	Melhor caso	Pior caso
Trocas	$O(n^2)$	$O(n^2)$
Bolhas	$O(n^2)$	$O(n^2)$
Inserção	$O(n^2)$	$O(n^2)$
Merge	$O(n \log n)$	$O(n \log n)$
Quick	$O(n \log n)$	$O(n^2)$

Recapitulando...

	Melhor caso	Pior caso
Trocas	$O(n^2)$	$O(n^2)$
Bolhas	$O(n^2)$	$O(n^2)$
Inserção	$O(n^2)$	$O(n^2)$
Merge	O(nlogn)	O(nlogn)
Quick	O(nlogn)	$O(n^2)$

No caso médio: O(nlogn)

Qual o pior caso?

Recapitulando...

	Melhor caso	Pior caso
Trocas	$O(n^2)$	$O(n^2)$
Bolhas	$O(n^2)$	$O(n^2)$
Inserção	$O(n^2)$	$O(n^2)$
Merge	O(nlogn)	O(nlogn)
Quick	O(nlogn)	$O(n^2)$

No caso médio: O(nlogn), constante é menor que a do merge

Ordenação no lugar, merge usa vetor auxiliar

Referências

AGUILAR, L. J.; ALONSO, M. C. Programação em C++: Algoritmos, Estruturas de Dados e Objetos. São Paulo: AMGH, 2007.

Khan Academy. **Divide and Conquer algorithms**. Disponível em: https://pt.khanacademy.org/computing/computer-science/algorithms/merge-sort/a/divide-and-conquer-algorithms. Acessado em 02/06/2022

Lehilton Pedrosa. Algoritmos de Ordenação. Slides. Disponível em: https://www.ic.unicamp.br/~lehilton/cursos/1s2016/mc102wy/ordenacao-slides.pdf. Acessado em: 02/08/2022