Quicksort

Profa. Franciny Medeiros
Disciplina de Estrutura de Dados 2
Bacharelado em Ciências da Computação - UFJ

Algoritmo de ordenação

- O problema de ordenação consiste em rearranjar um conjunto de dados em um determinada ordem.

- O Algoritmo de ordenação Quicksort resolve esse problema.
  - pertence a categoria de ordenação por troca
  - criado em 1962

Funcionamento

- O algoritmo **Quicksort** utiliza o paradigma de programação **Dividir para Conquistar** ou *divide and conquer:*
  - É uma abordagem recursiva em que a entrada do algoritmo é ramificada múltipla vezes a fim de dividir o problema em partes menores

Particionamento

- Dada uma sequência de entrada, o primeiro passo é escolher o **pivô**.

- O pivô é um elemento da sequência de dados que servirá como ponto para a partição
  - Escolher o primeiro elemento como pivô.
  - Escolher o último elemento como pivô.
  - Escolher um elemento aleatório como pivô.
  - Escolher a mediana como pivô.

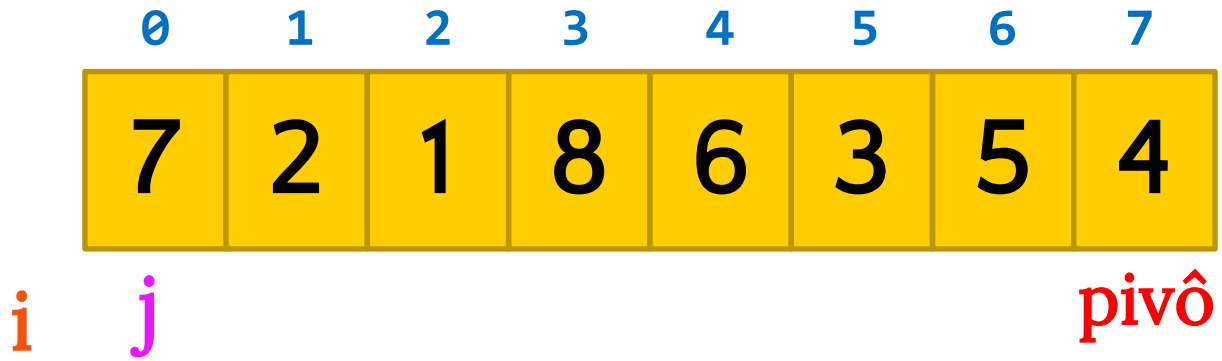
Particionamento

- Os elementos do vetor que forem maiores que o **pivô** serão considerados grandes e os demais serão considerados pequenos.

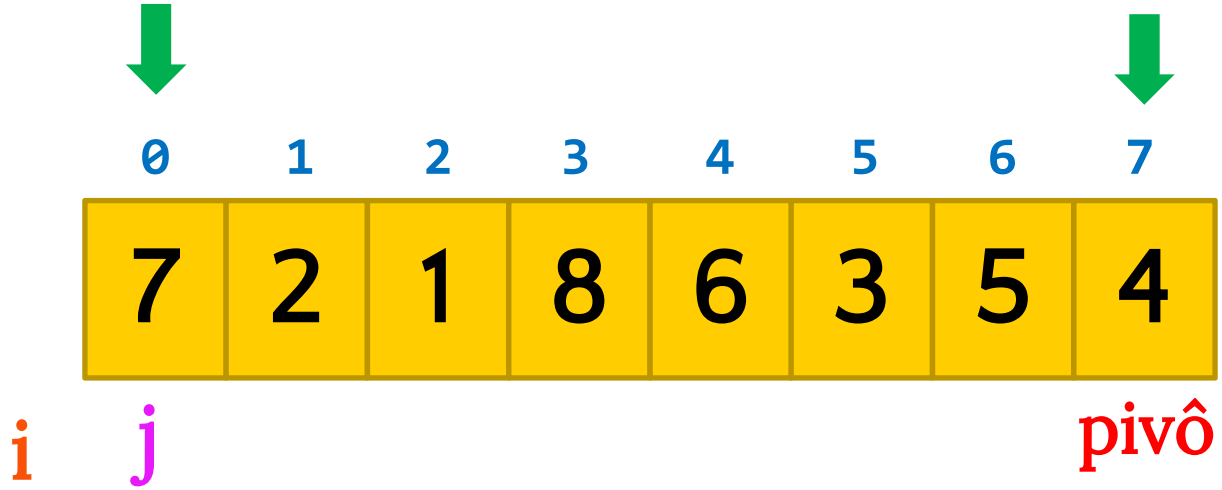
Executar recursivamente

- Traga o pivô para sua posição apropriada de forma que a esquerda do pivô seja menor e a direita maior;
- Quicksort na parte esquerda;
- Quicksort na parte direita;

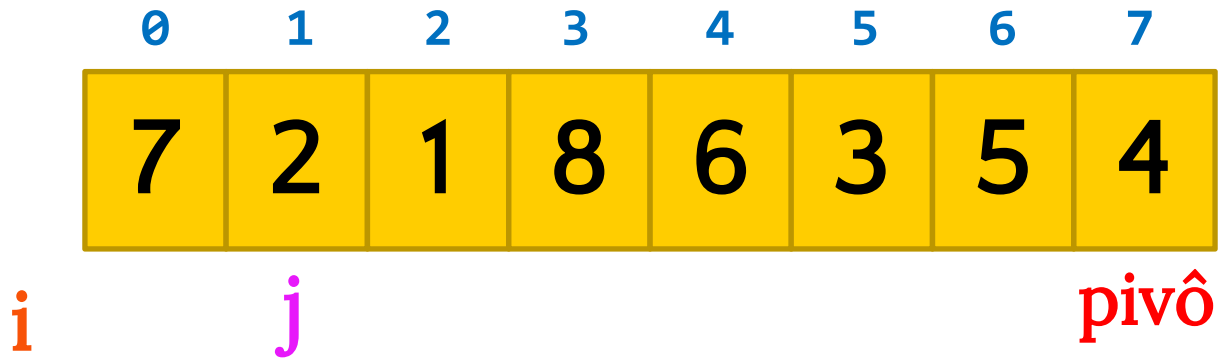
- Escolher o pivô
- i e j são variáveis de controle que ajudarão a organizar o vetor de modo que todos a esquerda do pivô sejam menores que o pivô, e a direita sejam maiores



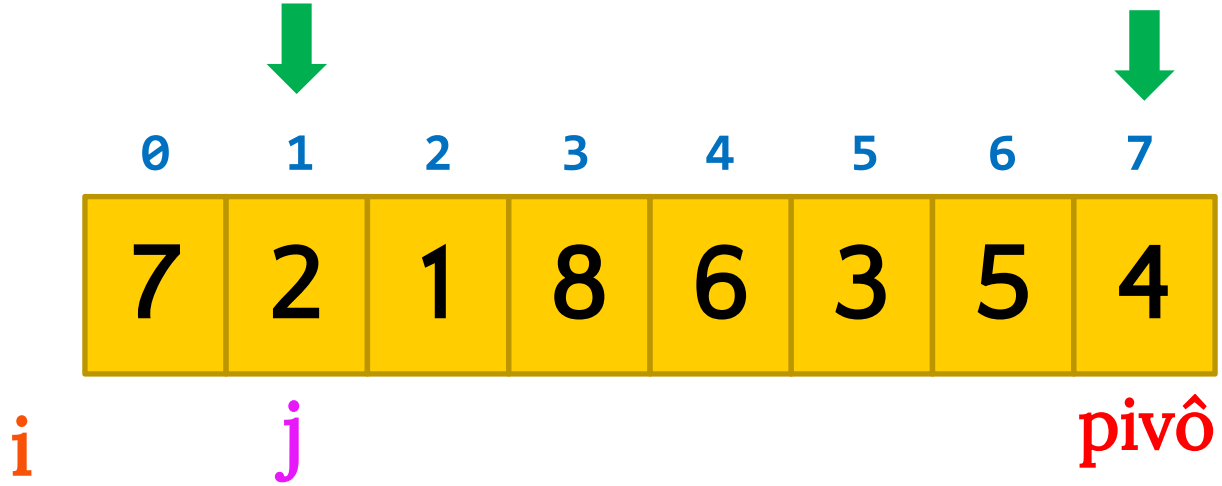

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



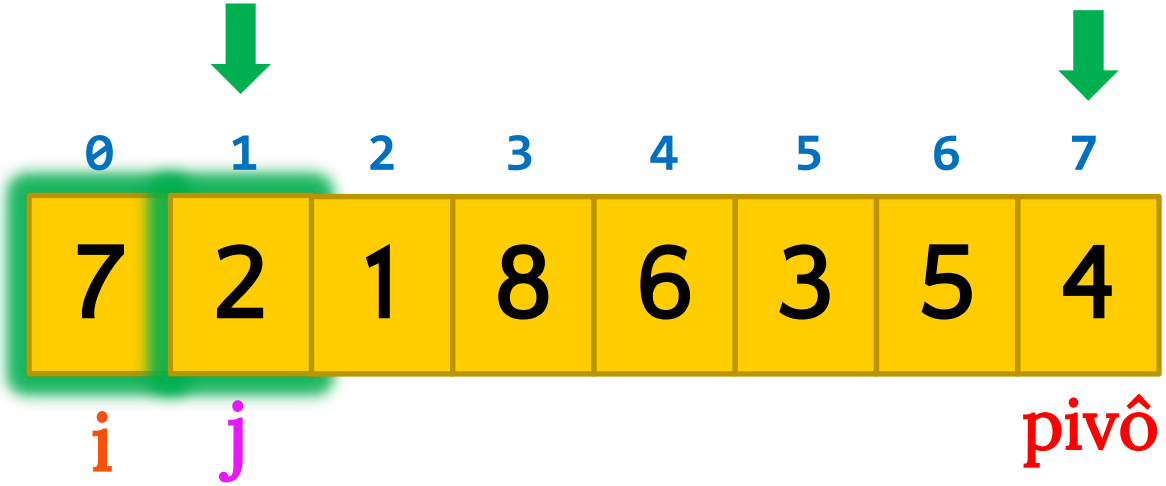

```
Se A[j] > A[pivo]
----> então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



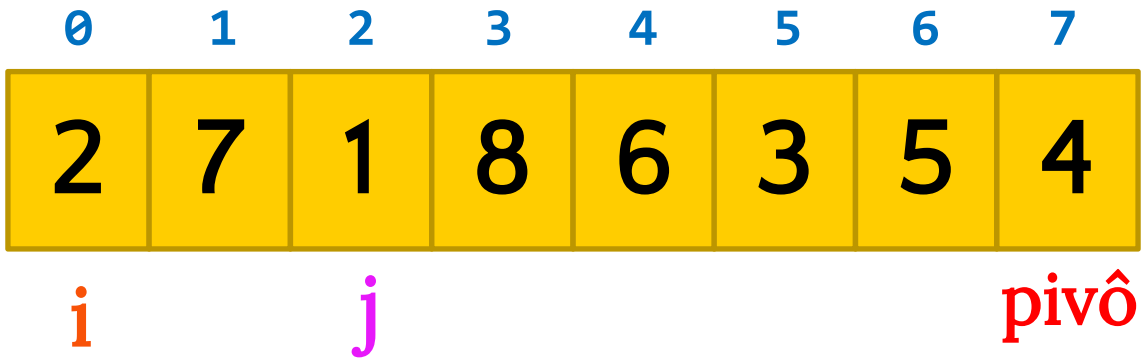

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



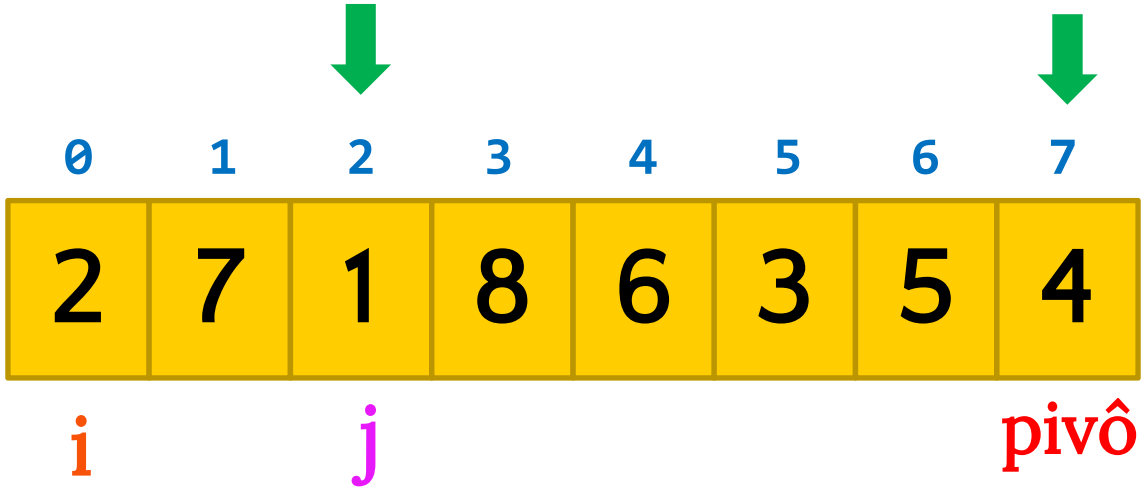

```
Se A[j] > A[pivo]
   então j++
Senão
----> i++
   troca A[j] <> A [i] e j++
```



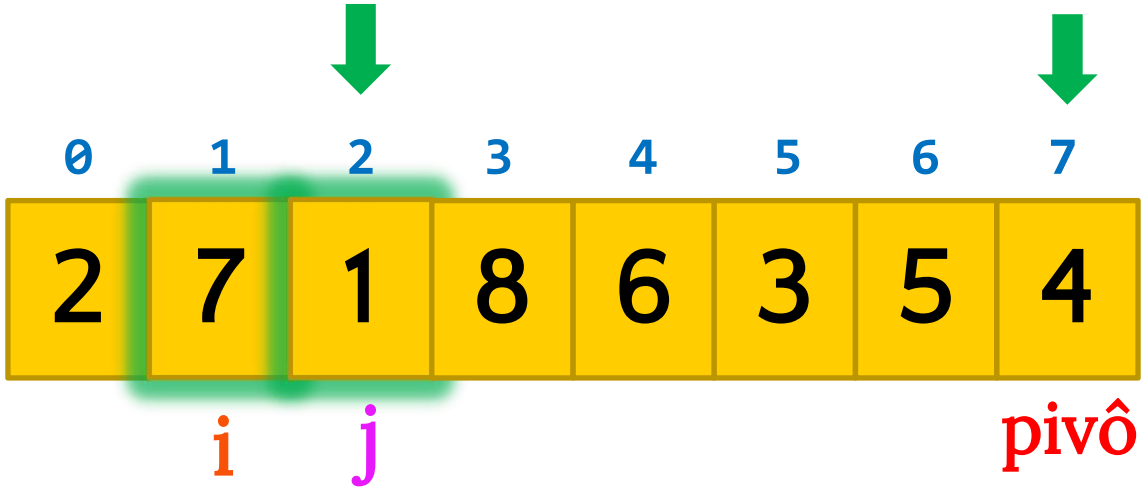

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
----> troca A[j] <> A [i] e j++
```



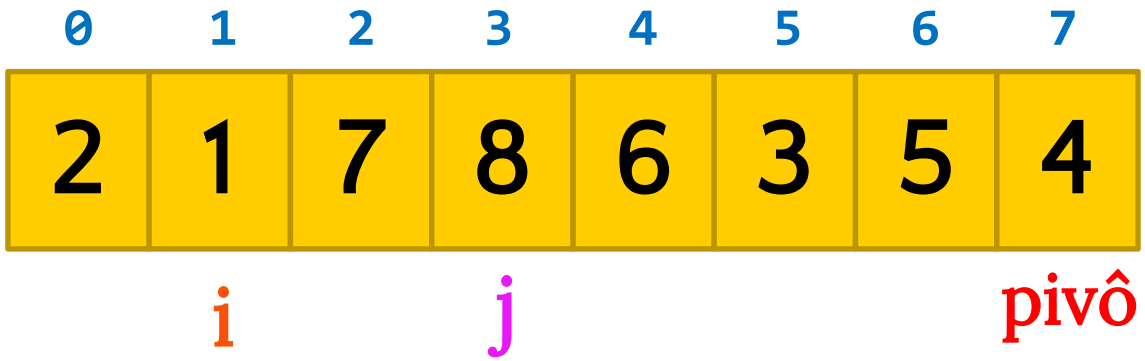

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



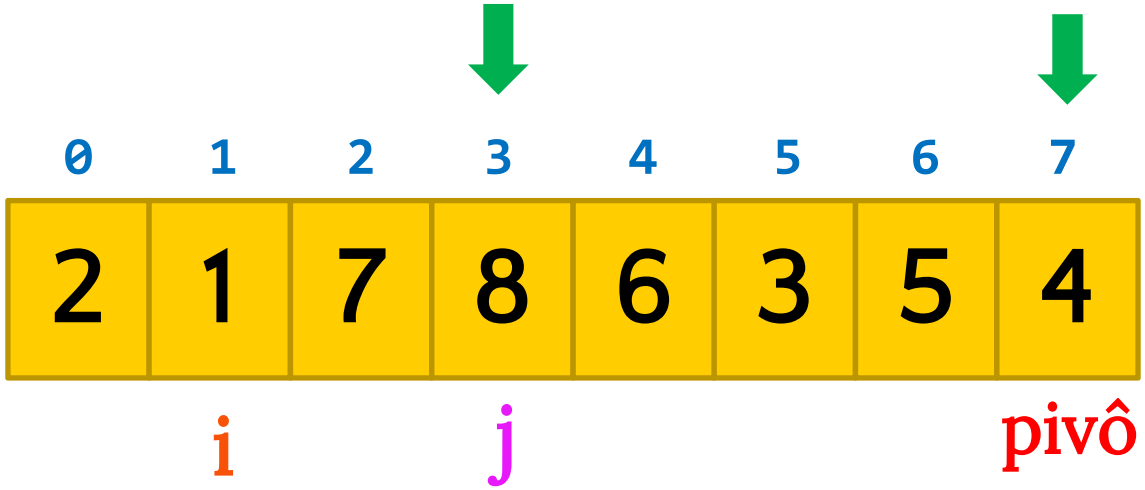

```
Se A[j] > A[pivo]
   então j++
Senão
----> i++
   troca A[j] <> A [i] e j++
```



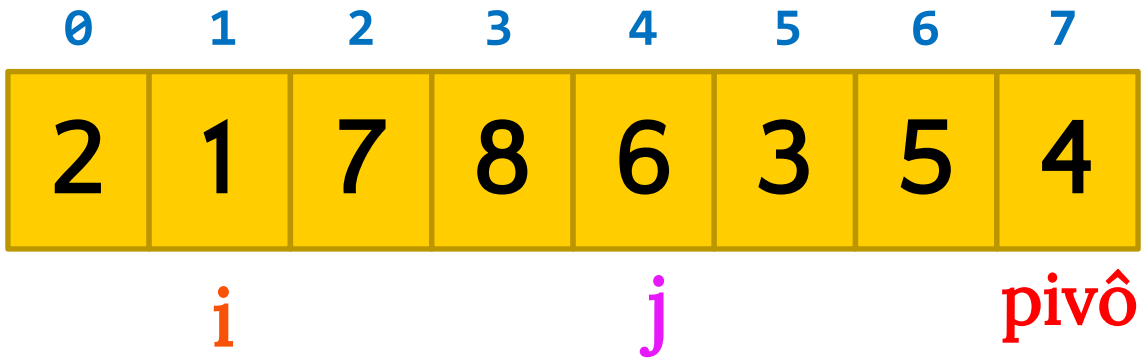

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
----> troca A[j] <> A [i] e j++
```



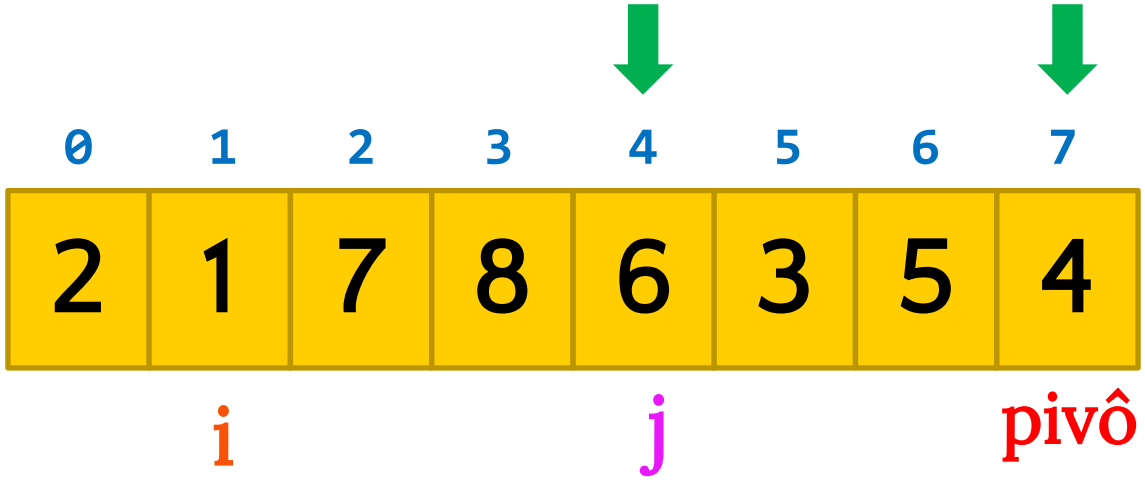

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



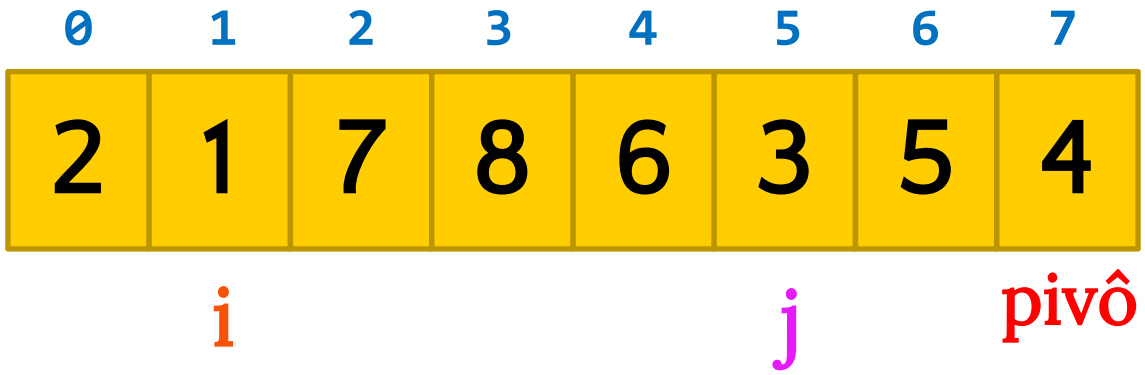

```
Se A[j] > A[pivo]
----> então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



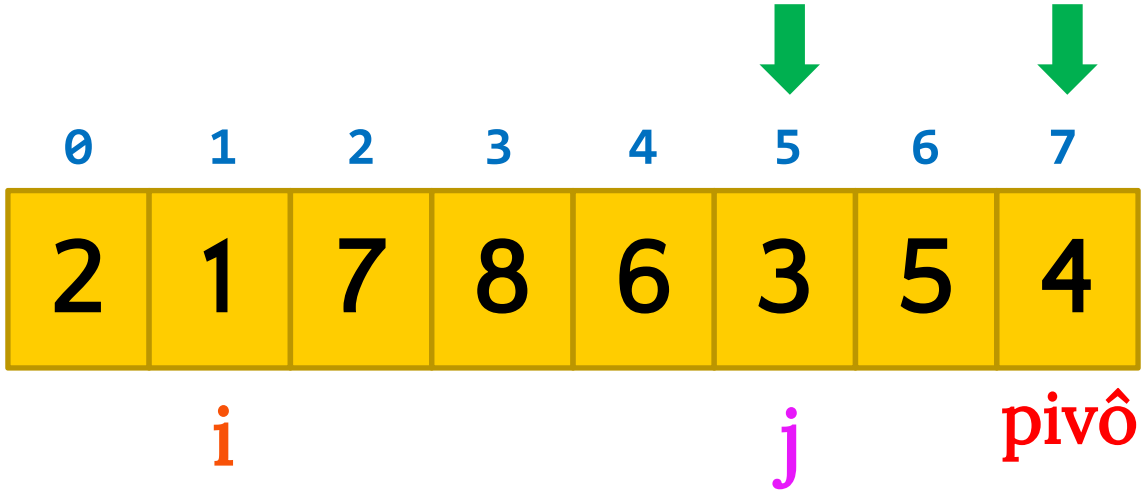

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



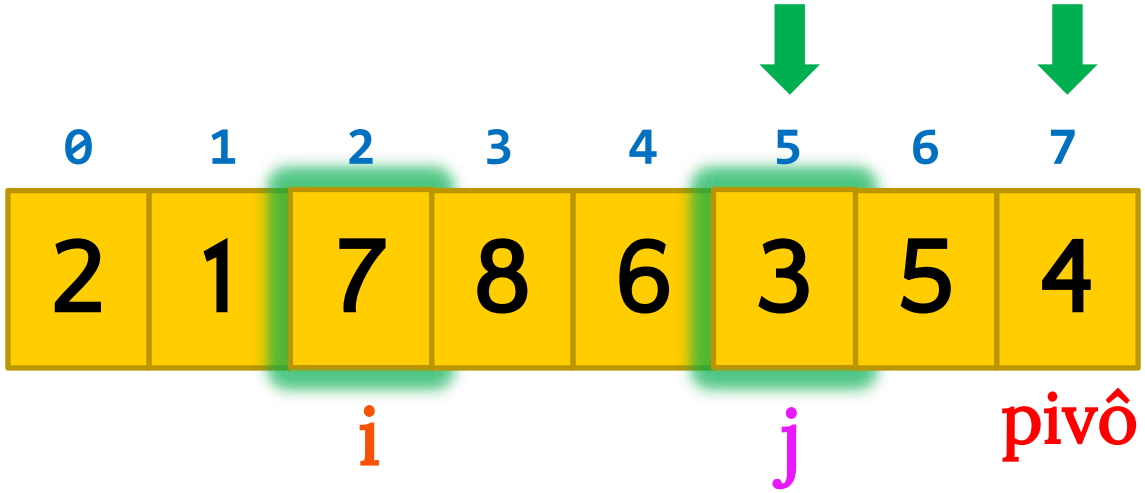

```
Se A[j] > A[pivo]
----> então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



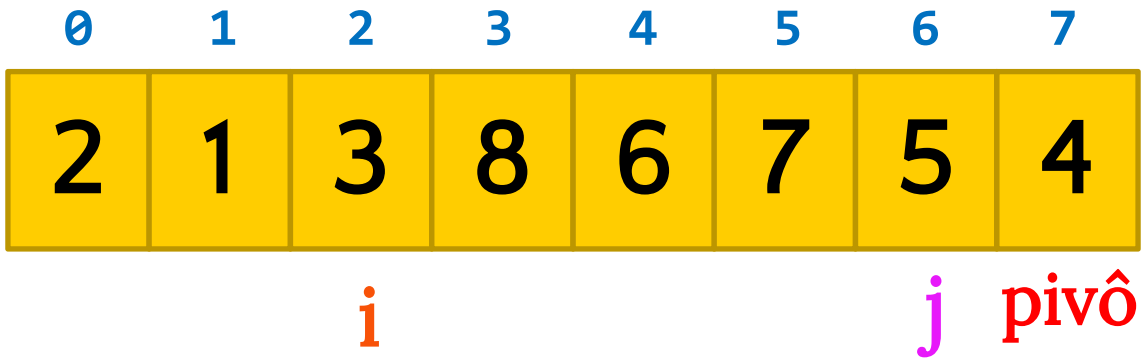

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```



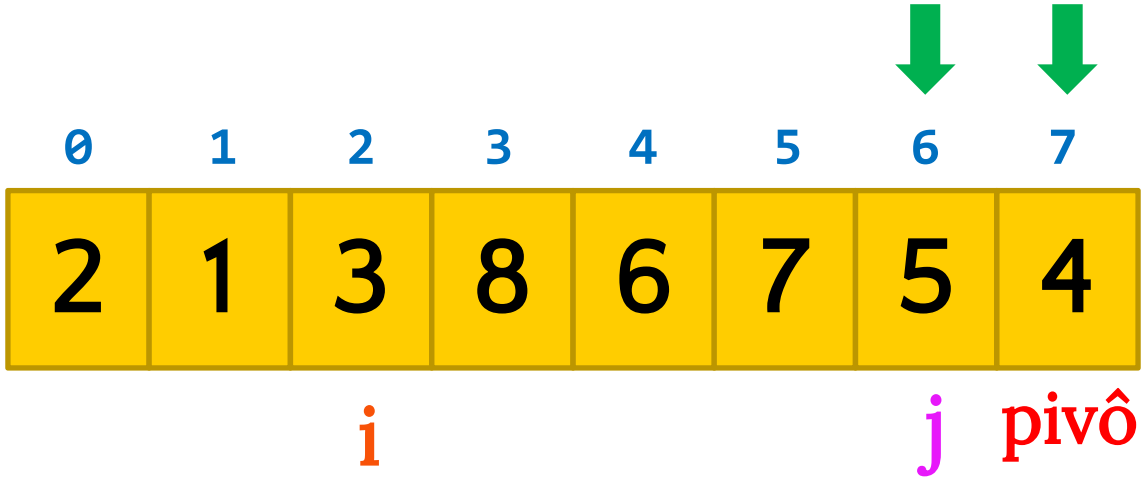

```
Se A[j] > A[pivo]
   então j++
Senão
----> i++
   troca A[j] <> A [i] e j++
```




```
Se A[j] > A[pivo]
   então j++
Senão
   i++
----> troca A[j] <> A [i] e j++
```




```
Se A[j] > A[pivo]
   então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```




```
Se A[j] > A[pivo]
----> então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```

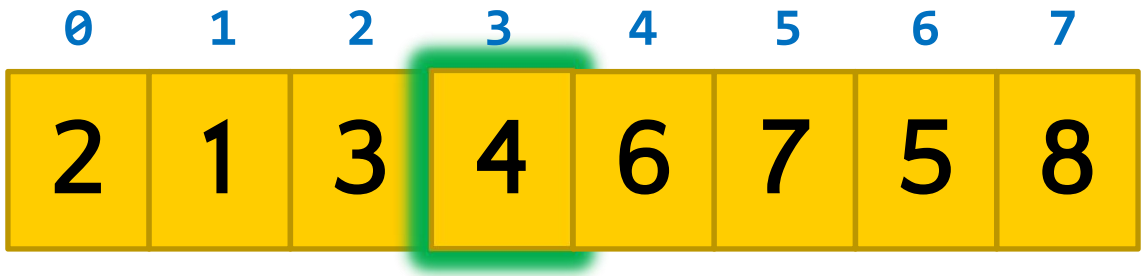

```
Se A[j] > A[pivo]
   então j++
Senão
   i++
   troca A[j] <> A [i] e j++
```

Quando j alcançar o final do vetor,
encerra-se o laço de repetição

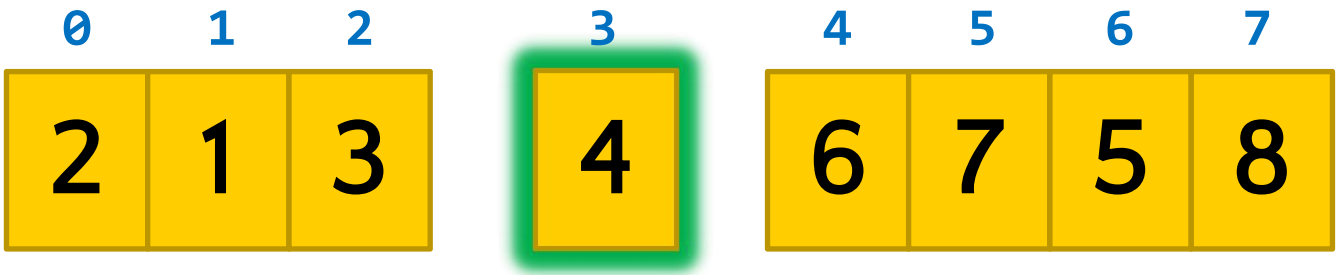
troca A[i+1] <> A[pivo]

Quando j alcançar o final do vetor,
encerra-se o laço de repetição

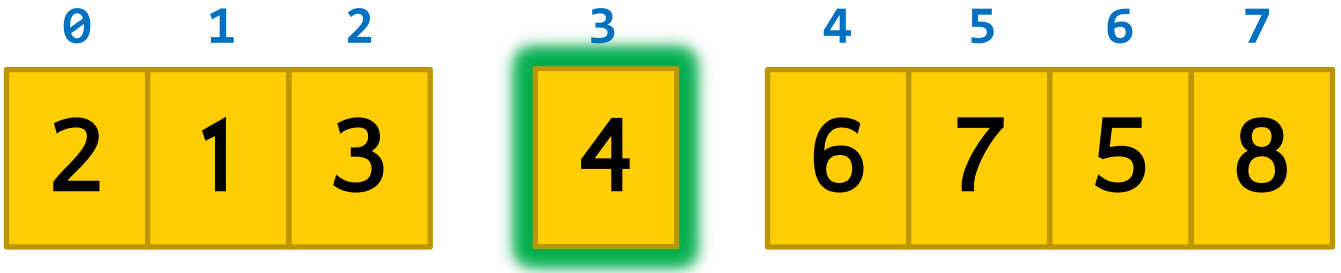
troca A[i+1] <> A[pivo]



Após a primeira execução, esse será o vetor resultante.
Note que a esquerda do pivô estão elementos menores,
e a direita estão elementos maiores.

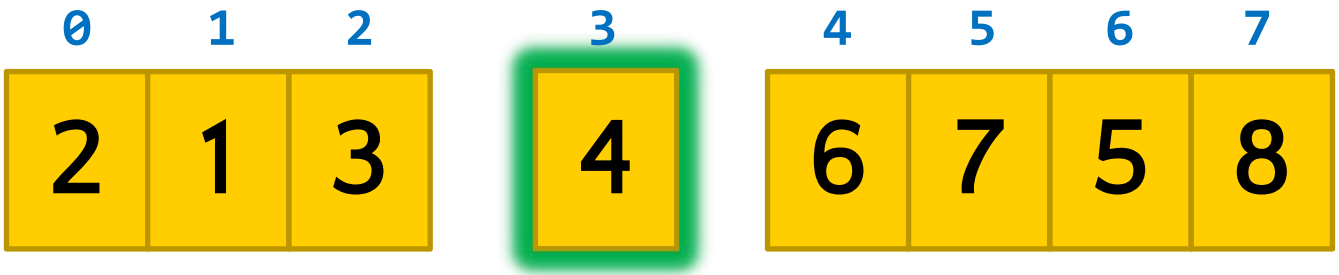


O próximo passo é executar novamente o quick sort para o lado esquerdo do pivô, e para o lado direito do pivô



`quickSort(A[], 0, pivo-1)`

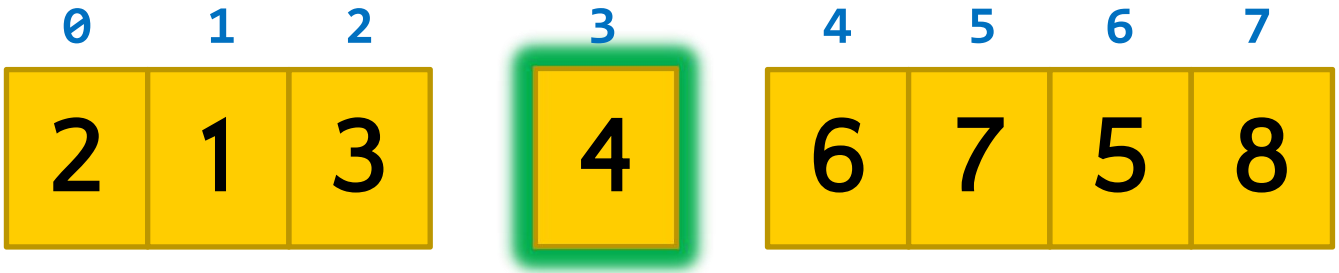
`quickSort(A[], pivo+1, n)`



`quickSort(A[], 0, pivo-1)`

`quickSort(A[], pivo+1, n)`

Até quando devemos chamar a recursão?

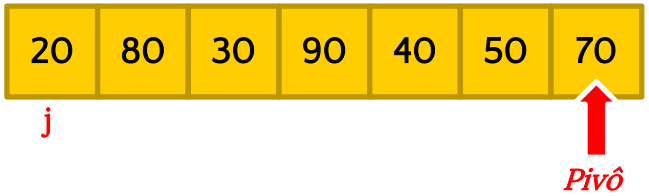


quickSort(A[], 0, pivo-1)

quickSort(A[], pivo+1, n)

Até quando devemos chamar a recursão?

Até o vetor ter tamanho 1



i = -1

Variáveis de controle

i: índice do menor elemento

j: variável do loop

Condição:

vetor[j] <= pivô ?

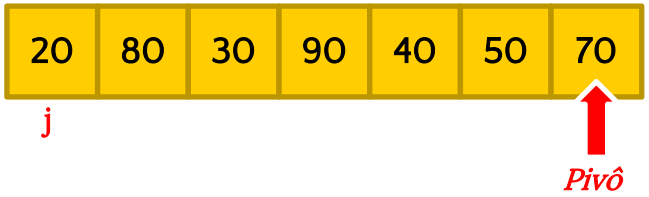
Se sim então

i++

troca (vetor[i], vetor[j])

PASSO 1

Pivo = último elemento



i = 0

j = 0

Variáveis de controle

i: índice do menor elemento

j: variável do loop

Condição:

vetor[j] <= pivô ?

Se sim então

i++

troca (vetor[i], vetor[j])

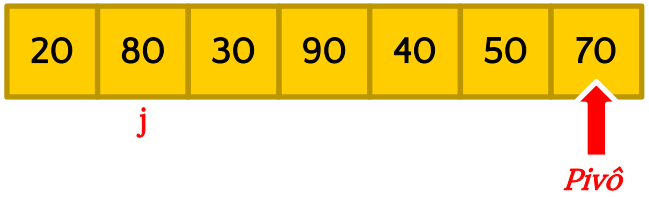
20 < 70 SIM

i = 0

j = 0

troca vetor[0], vetor[0]

Pivo = último elemento



i = 0

j = 1

Variáveis de controle

i: índice do menor elemento

j: variável do loop

Condição:

vetor[j] <= pivô ?

Se sim então

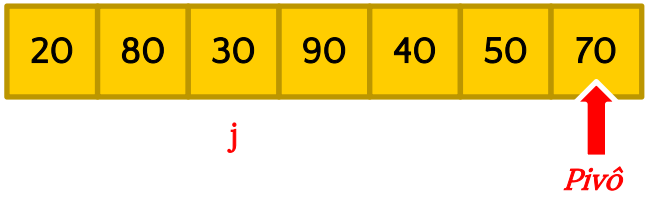
i++

troca (vetor[i], vetor[j])

80 < 70 NÃO

faz nada!

Pivo = último elemento



i = 1

j = 2

Variáveis de controle

i: índice do menor elemento

j: variável do loop

Condição:

vetor[j] <= pivô ?

Se sim então

i++

troca (vetor[i], vetor[j])

30 < 70 SIM

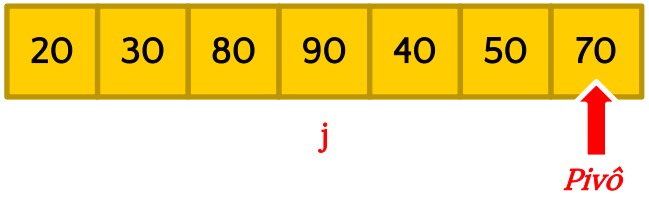
i = 1

j = 2

troca vetor[1], vetor[2]

Vetor resultante

Pivo = último elemento



i = 1

j = 3

Variáveis de controle

i: índice do menor elemento

j: variável do loop

Condição:

vetor[j] <= pivô ?

Se sim então

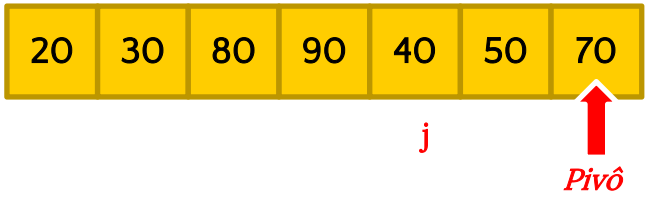
i++

troca (vetor[i], vetor[j])

90 < 70 NÃO

faz nada!

Pivo = último elemento



i = 1

j = 4

Variáveis de controle

i: índice do menor elemento

j: variável do loop

Condição:

vetor[j] <= pivô ?

Se sim então

i++

troca (vetor[i], vetor[j])

40 < 70 SIM

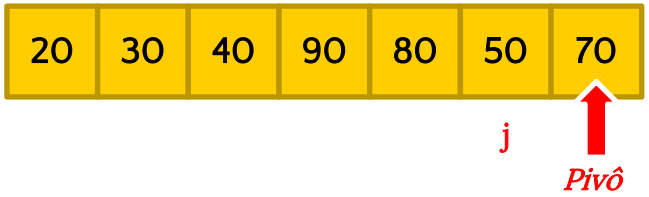
i = 2

j = 4

troca vetor[2], vetor[4]

Vetor resultante

Pivo = último elemento



i = 2

j = 5

Variáveis de controle

i: índice do menor elemento

j: variável do loop

Condição:

vetor[j] <= pivô ?

Se sim então

i++

troca (vetor[i], vetor[j])

50 < 70 SIM

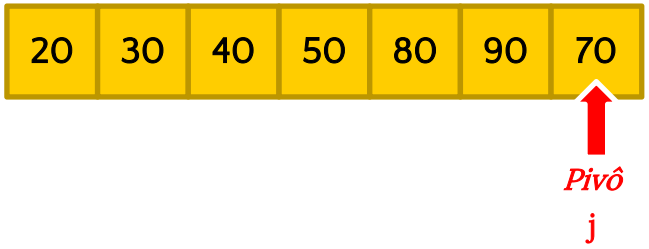
i = 3

j = 5

troca vetor[3], vetor[5]

Vetor resultante

Pivo = último elemento



i = 3

j = 5

Variáveis de controle

i: índice do menor elemento

j: variável do loop

Condição:

vetor[j] <= pivô ?

Se sim então

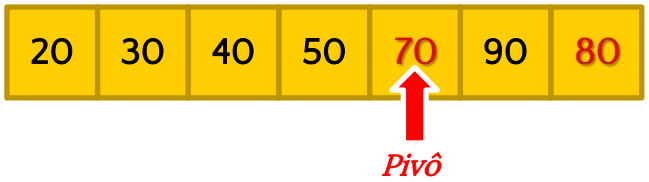
i++

troca (vetor[i], vetor[j])

Agora j = 6 (tem que sair do loop)

trocar o vetor[i+1] com o pivô

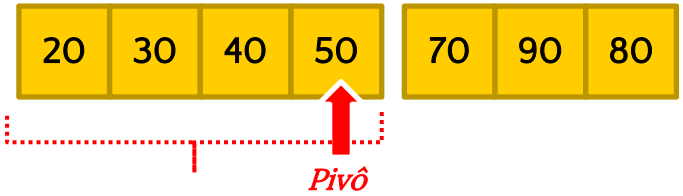
Vetor resultante



Agora o pivô está na posição correta

Então precisa executar o quicksort para a parte esquerda e depois para a direita

Pivo = último elemento



Variáveis de controle

i: índice do menor elemento

j: variável do loop

Por ser recursivo, chama-se para o vetor a esquerda do pivô

Condição:

vetor[j] <= pivô ?

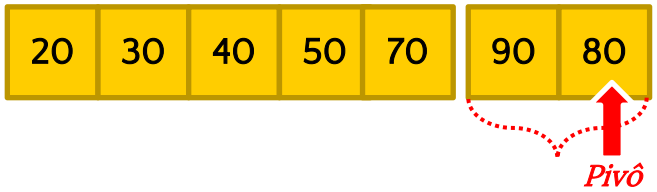
Se sim então

i++

troca (vetor[i], vetor[j])

Escolhe um novo pivô e repete todo o processo...

Depois faz o mesmo para o lado direito



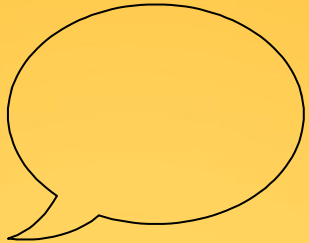
Pivo = último elemento

```
int partition (int arr[], int low, int high)
{
    int pivot = arr[high];
    int i = (low - 1);

    for (int j = low; j <= high- 1; j++)
    {
        if (arr[j] <= pivot)
        {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);

        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```



Dúvidas?

franciny@ufj.edu.br