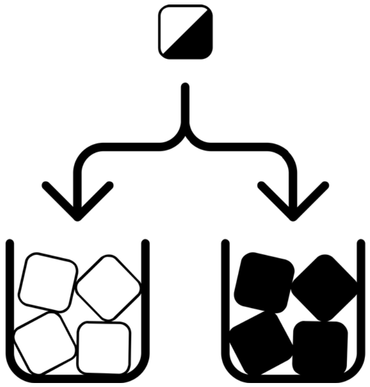
Pesquisa e Ordenação de Dados

Unidade 2.5:

Quick Sort



Quick Sort

- Algoritmo do tipo **divisão e conquista**
  - *Divide and conquer*: técnica que consiste em decompor o problema a ser resolvido em instâncias cada vez menores do mesmo tipo de problema, resolver estas instâncias (em geral, recursivamente) e, por fim, combinar as soluções parciais para obter uma solução do problema original.
- Método proposto por Tony Hoare em 1961

Sir Charles Antony Richard Hoare
1980 Turing Award

Quick Sort

- Ideia geral:
  - Escolher um elemento especial, chamado de **Pivô**;
  - Posicionar todos os elementos **menores ou iguais ao Pivô à sua esquerda** e os **maiores à sua direita**;
  - Chamar recursivamente a função para a parte esquerda e para a parte direita.

<= pivô	pivô	> pivô

ordena com Quicksort

ordena com Quicksort

Quick Sort

- O funcionamento do algoritmo obedece aos seguintes passos:
  - Selecionar o Pivô
    - Esta é uma fase importante, pois a escolha de um bom pivô torna o método mais eficiente.
    - Existem várias técnicas para selecionar um pivô:
      - Pegar o primeiro elemento
      - Pegar o último elemento
      - Pegar o elemento do meio
      - Pegar 3 elementos e escolher o mediano
      - Pegar um elemento aleatório

Vamos exemplificar com esse!

Quick Sort

- O funcionamento do algoritmo obedece aos seguintes passos:
  - Selecionar o Pivô
  - Fase de particionamento
    - Posicionar todos os elementos menores ou iguais ao pivô à sua esquerda
      - Considerando o elemento **p** como a posição do pivô, e **n** o tamanho máximo do vetor, temos que:

$$\forall A[j] \in A,\ \ j < p\ \wedge\ j \geq 0\ \Rightarrow\ (A[j] \leq A[p])$$

    - Posicionar todos os elementos maiores que o pivô à sua direita
      - Considerando o elemento **p** como a posição do pivô, e **n** o tamanho máximo do vetor, temos que:

$$\forall A[i] \in A,\ \ i > p\ \wedge\ i < n\ \Rightarrow\ (A[i] > A[p])$$

Quick Sort

- O funcionamento do algoritmo obedece aos seguintes passos:
  - Selecionar o Pivô
  - Fase de particionamento
  - Chamadas recursivas para a partição da esquerda e para a partição da direita
    - Para a partição da esquerda é feita uma chamada recursiva considerando os elementos de 0 até a posição anterior ao pivô

      $$quickSort(vet, 0, p-1)$$

    - Para a partição da esquerda é feita uma chamada recursiva considerando como primeiro elemento o elemento posterior do pivô até o último elemento

      $$quickSort(vet, p+1, n)$$

Quick Sort

- O funcionamento do algoritmo obedece aos seguintes passos:
  - Selecionar o Pivô
  - Fase de particionamento
  - Chamadas recursivas para a partição da esquerda e para a partição da direita
    - Para a partição da esquerda é feita uma chamada recursiva considerando os elementos de 0 at

      $quickSort(vet, 0, p-1)$

    - Para a partição da esquerda é feita uma chamada recursiva considerando como primeiro elemento o elemento posterior do pivô até o último elemento

      $quickSort(vet, p+1, n)$

os limites inicial/final irão variar a cada etapa

Quick Sort

Como funciona - Pivô

30	21	43	3	9	82	$\boxed{15}$

pivô

última posição

Quick Sort
Como funciona - Particionamento

30	21	43	3	9	82	15

pivô

Particionamento: dividir este conjunto de elementos em 2 regiões:
- menores ou iguais ao pivô
- maiores do que o pivô

Ao final da etapa, o pivô será colocado entre essas duas regiões.

Quick Sort

Como funciona - Particionamento

i						
30	21	43	3	9	82	15
k						pivô

i = percorre o vetor do início até a posição anterior ao pivô. Representa o elemento **atual** que está sendo comparado com o pivô.

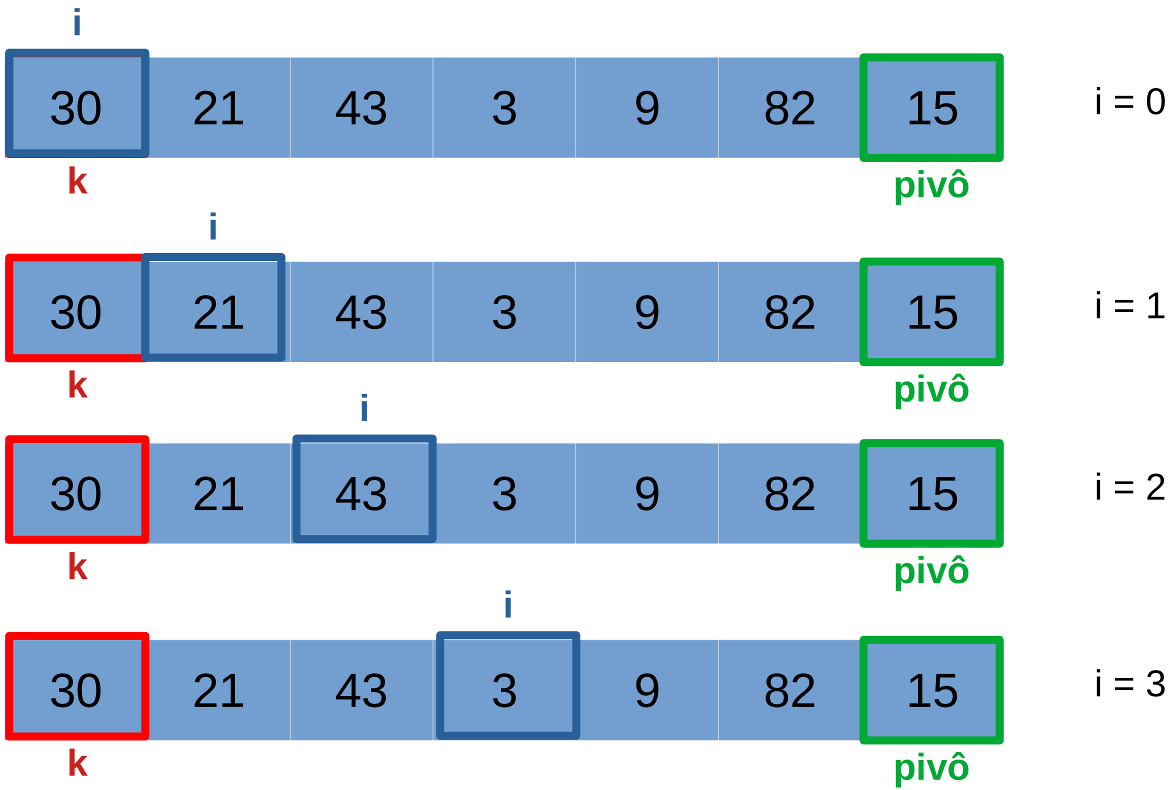
k = representa o **limite** entre os elementos menores ou iguais ao pivô (que já foram posicionados) e a região onde estão os elementos maiores que o pivô. Troca com **i** sempre que um elemento **<= pivô** for encontrado.

O objetivo é que todos os elementos menores ou iguais ao pivô sejam colocados antes de **k**.
Ao final da iteração, o pivô troca com **k**, indo portanto para sua posição correta.

Quick Sort

Como funciona - Particionamento

Se A[i] <= A[pivô]
troca(A[i], A[k])



Quick Sort

Como funciona - Particionamento

Se A[i] <= A[pivô]
troca(A[i], A[k])

			i				
30	21	43	3	9	82	15	i = 3
k						pivô	

			i				
3	21	43	30	9	82	15	i = 3
	k					pivô	

				i			
3	21	43	30	9	82	15	i = 4
	k					pivô	

					i		
3	9	43	30	21	82	15	i = 5
		k				pivô	

Quick Sort

Como funciona - Particionamento

Se A[i] <= A[pivô]
    troca(A[i], A[k])

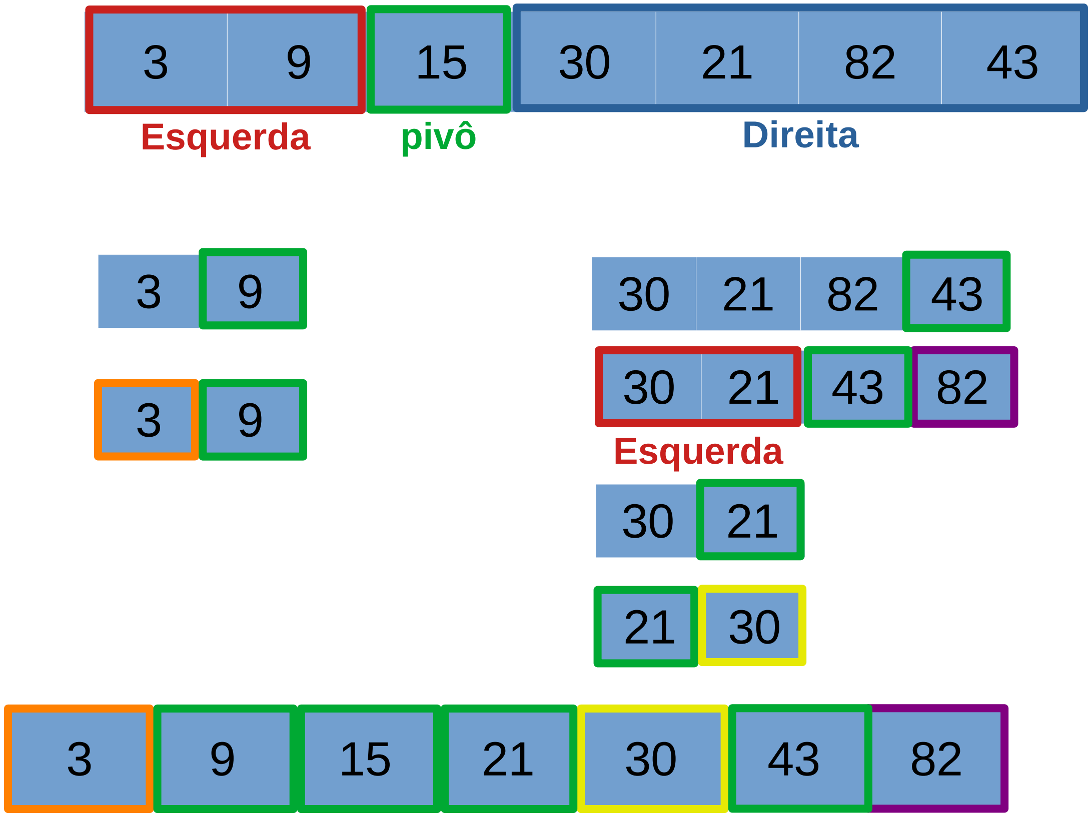
					i	
3	9	43	30	21	82	15
		k				pivô

i = 5

					i	
3	9	15	30	21	82	43
		pivô				

FINAL

Quick Sort
Como funciona – Chamada recursiva



Faz o particionamento sempre que o **limite inferior** for **menor** que o **limite superior**

Quick Sort
Implementação

- O algoritmo será implementado em duas partes:
  - **quickSort**
    - Função principal que será chamada pelos demais programas
    - Responsável por chamar a função de particionamento e a função recursiva
    - Recebe como parâmetro o vetor, seu índice inicial e seu índice final
  - **particiona**
    - Função auxiliar responsável pela seleção do pivô (último elemento) e organizar os dados conforme a lógica do método
      - Menores à esquerda do pivô e maiores à direita
    - Recebe como parâmetro o vetor e os limites de cada segmento (índices inicial e final)
    - Reorganiza o vetor original
    - Retorna a posição do pivô

Quick Sort
Pseudocódigo

```
função quickSort(A[], inicio, fim)
inicio
    se inicio < fim então
        posPivo ← particiona(A[], inicio, fim)    /* vai retornar a posicao do pivô */
        quickSort(A, inicio, posPivo-1)
        quickSort(A, posPivo+1, fim)
    fimSe
fim
```

Segmento tem pelo menos 2 elementos

Chamada recursiva p/ elementos à esquerda do pivô

Chamada recursiva p/ elementos à direita do pivô

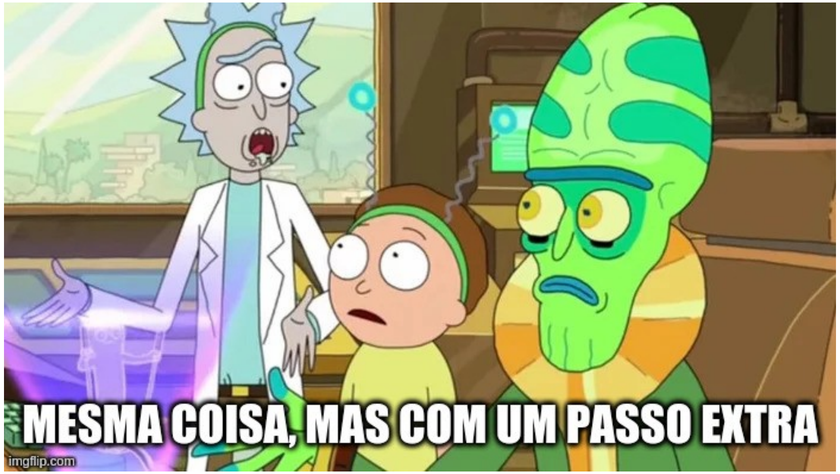
Quick Sort
Pseudocódigo

```
função particiona(A[], inicio, fim)
inicio
    posPivo ← fim /* usando o ultimo; poderia ser outra estratégia */
    k ← inicio    /* k: posição de swap para o pivo */
    para i = inicio, i < fim  faça
        se A[i] <= A[posPivo] entao
            troca(A[i], A[k])
            k++
       fimSe
    fimPara
   se A[k] > A[posPivo] entao
       troca(A[k], A[posPivo])
   fimSe
   retorna posPivo
```

Quick Sort - Particionamento

Exemplo passo a passo (1)

k								
3	5	8	9	10	2	4	6	7
i								pivo



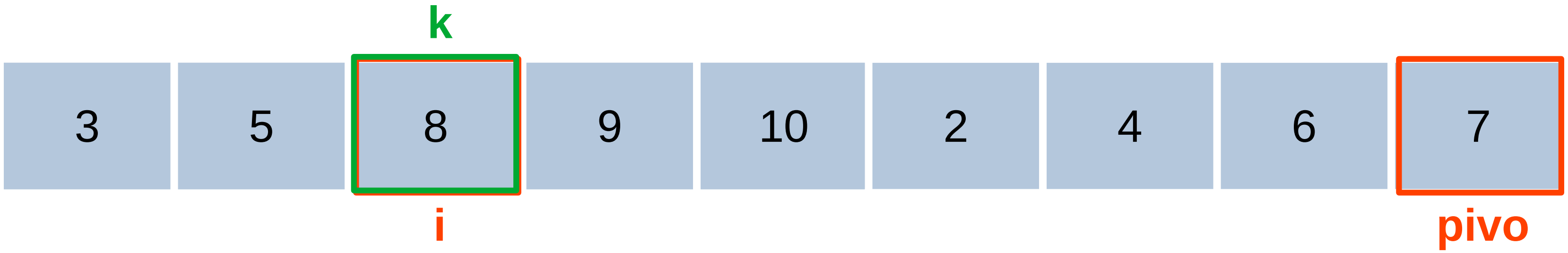
Quick Sort - Particionamento

Exemplo passo a passo (2)

	k							
3	5	8	9	10	2	4	6	7
	i							pivo

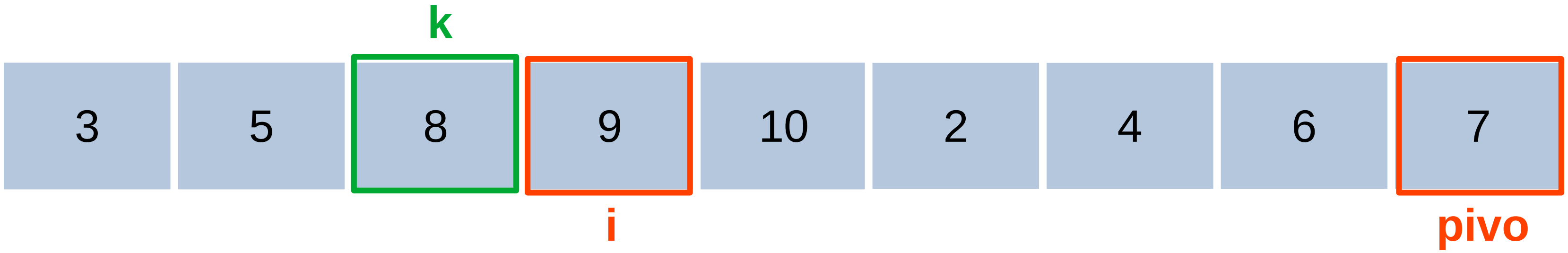
Quick Sort - Particionamento

Exemplo passo a passo (3)



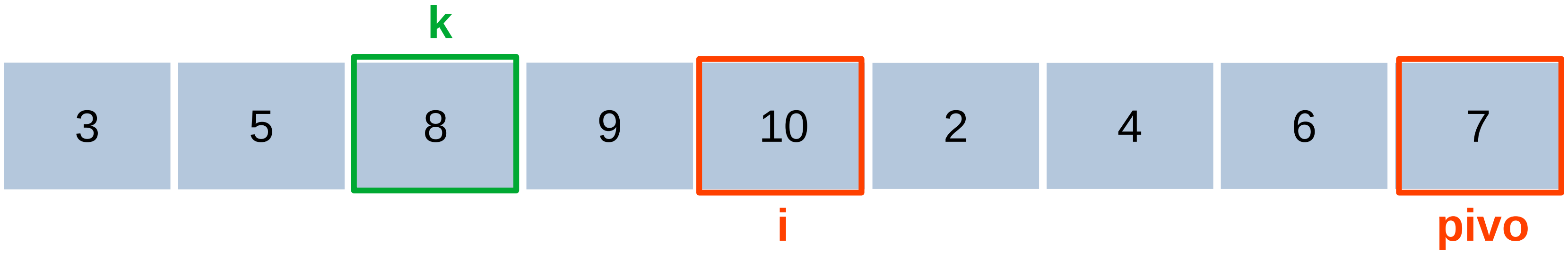
Quick Sort - Particionamento

Exemplo passo a passo (4)



Quick Sort - Particionamento

Exemplo passo a passo (5)



Quick Sort - Particionamento

Exemplo passo a passo (6)

		k						
3	5	8	9	10	2	4	6	7
					i			pivo

Quick Sort - Particionamento

Exemplo passo a passo (7)

			k					
3	5	2	9	10	8	4	6	7
					i			pivo

Quick Sort - Particionamento

Exemplo passo a passo (8)

			k					
3	5	2	9	10	8	4	6	7
						i		pivo

Quick Sort - Particionamento

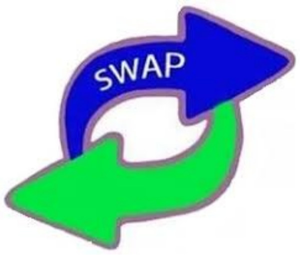
Exemplo passo a passo (9)

				k				
3	5	2	4	10	8	9	6	7
						i		pivo

Quick Sort - Particionamento

Exemplo passo a passo (10)

				k				
3	5	2	4	10	8	9	6	7
							i	pivo



Quick Sort - Particionamento

Exemplo passo a passo (11)

					k			
3	5	2	4	6	8	9	10	7
							i	pivo

Laço terminou

Troca o pivô pelo elemento K se este for maior

Quick Sort - Particionamento

Exemplo passo a passo (12)

k

3	5	2	4	6	7	9	10	8

ex-pivo

Laço terminou

Troca o pivô pelo elemento K se este for maior

Retorna posição atual do pivô

Quick Sort - Particionamento

Exemplo passo a passo (13)

3	5	2	4	6	7	9	10	8

inicio..pivô-1

pivô+1..fim

Chamadas recursivas

e continua...

Quick Sort - Outro Exemplo

- Lista original:

23	17	8	15	9	12	19	7

	0	1	2	3	4	5	6	7
QuickSort(A, 0, 7)	23	17	8	15	9	12	19	7
particiona(A, 0, 7)	7	17	8	15	9	12	19	23
QuickSort(A, 0, -1), QuickSort(A, 1, 7)	7	17	8	15	9	12	19	23
QuickSort(A, 1, 7)	7	17	8	15	9	12	19	23
particiona(A, 1, 7)	7	17	8	15	9	12	19	23
QuickSort(A, 1, 6), QuickSort(A, 8, 7)	7	17	8	15	9	12	19	23
QuickSort(A, 1, 6)	7	17	8	15	9	12	19	23
particiona(A, 1, 6)	7	17	8	15	9	12	19	23
QuickSort(A, 1, 5), QuickSort(A, 7, 6)	7	17	8	15	9	12	19	23
QuickSort(A, 1, 5)	7	17	8	15	9	12	19	23
particiona(A, 1, 5)	7	8	9	12	17	15	19	23
QuickSort(A, 1, 2), QuickSort(A, 4, 5)	7	8	9	12	17	15	19	23
QuickSort(A, 1, 2)	7	8	9	12	17	15	19	23
QuickSort(A, 4, 5)	7	8	9	12	15	17	19	23
	7	8	9	12	15	17	19	23

Quick Sort
Análise

- $\theta(n \log n)$ no melhor caso
  - pivô que divide o vetor em duas partes exatamente iguais
- $O(n^2)$ no pior caso
  - Caso a escolha do pivô seja sempre o elemento localizado no fim do vetor, nossa implementação pode cair nesse problema se o vetor estiver em ordem crescente
  - Neste caso, o particionamento gera uma partição com zero elementos e outra com todos os demais
    - Para solucionar pode ser feita a implementação de pivô randômico, ou mesmo randomizar a entrada
    - Pivô randômico torna improvável atingir o pior caso
- Resumo:
  - Na teoria: seu pior caso é aproximadamente igual aos dos algoritmos mais simples de ordenação
  - Na prática: é um dos mais eficientes algoritmos de ordenação
- Não estável
- In place