Algoritmos e Estruturas de Dados
Aula 6 :: QuickSort

Filipe Cordeiro (filipe.rolim@ufrpe.br)

UNIVERSIDADE
FEDERAL RURAL
DE PERNAMBUCO

Crédito slides: Prof. Francisco Simões (francisco.simoes@ufrpe.br)

Quicksort

Quicksort
(Problema)

- Ordenar sequência de elementos

25	4	19	10	99	84	13	2	30	50

2	4	10	13	19	25	30	50	84	99

Quicksort
(Intuição)

- Utilizar abordagem de dividir para conquistar preocupando-se com a divisão (ao contrário do mergesort)
- Sem preocupação com reagrupamento

Dividir

Quicksort (Algoritmo)

- Divide o array em duas partes com relação a um pivô
  - A esquerda do pivô ficam os elementos menores que ele
  - A direita do pivô ficam os elementos maiores que ele
  - Ao fim das divisões o array já estará ordenado

QuickSort

- Como o Mergesort, também é baseado no paradigma “dividir para conquistar”
- Entretanto, divisões das partições são dinâmicas
- Passos básicos:

1. Escolhe-se um pivô

2. O pivô é posicionado de forma que todos os elementos anteriores a ele sejam menores e todos os posteriores, maiores

Elementos menores que o pivô	pivô	Elementos maiores que o pivô

3. Recursivamente, subvetores à esquerda e à direita são ordenados

QuickSort

- Com base no item 2, fica claro que o pivô encontra-se na sua posição correta
    - No entanto, os subvetores à esquerda e à direita não necessariamente estarão ordenados
- Ao executar as chamadas recursivas, ordenamos os subvetores

QuickSort

- Exemplo
- 25 57 48 37 12 92 86 33

QuickSort

- Exemplo
- (25 57 48 37 12 92 86 33}
- pivo

QuickSort

- Exemplo
- (12) 25 (57 48 37 92 86 33)
- Pivo colocado na posição correta
    - Por enquanto, abstraímos a forma como isso é feito...
    - 25 encontra-se na posição correta
    - (12) < 25 < (57 ... 33)
- Aplicamos o metodo recursivamente a cada um dos subvetores

QuickSort

- Exemplo
- (12) 25 (57 48 37 92 86 33)
    - Ordenação do subvetor à esquerda de 25 (antigo pivô)

QuickSort

- Exemplo
- (12) 25 (57 48 37 92 86 33)
- Ordenação do subvetor à direita de 25 (antigo pivô)

QuickSort

- Exemplo
- 12 25 (48 37 33) 57 (92 86)

QuickSort

- Exemplo
- 12 25 (48 37 33) 57 (92 86)

QuickSort

- Exemplo
- 12 25 (37 33) 48 57 (92 86)

QuickSort

- Exemplo
- 12 25 (37 33) 48 57 (92 86)

QuickSort

- Exemplo
- 12 25 (33) 37 48 57 (92 86)

QuickSort

- Exemplo
- 12 25 (33) 37 48 57 (92 86)

QuickSort

- Exemplo
- 12 25 33 37 48 57 (92 86)

QuickSort

- Exemplo
- 12 25 33 37 48 57 (92 86)

QuickSort

- Exemplo
- 12 25 33 37 48 57 (86 92)

QuickSort

- Exemplo
- 12 25 33 37 48 57 (86) 92

QuickSort

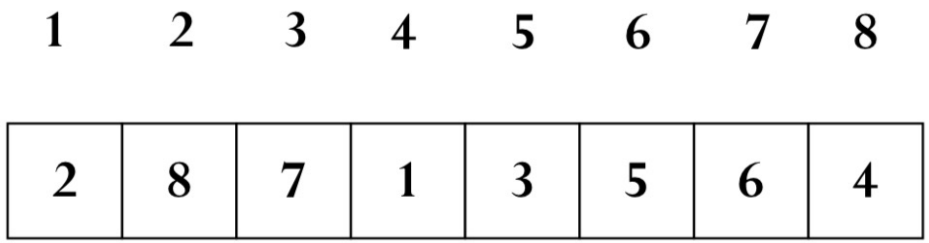
- Exemplo
- 12 25 33 37 48 57 (86) 92

QuickSort

- Exemplo
- 12 25 33 37 48 57 86 92

Quicksort

- Exemplo: ordenar o vetor A = {2, 8, 7, 1, 3, 5, 6, 4} usando o algoritmo quicksort com pivô igual ao último elemento de cada partição.



Quicksort

- O principal procedimento do quicksort é o de particionamento.
- O vetor A[p..r] é rearranjado de acordo com a escolha de um ponto arbitrário q, chamado de pivô.
- O vetor A é particionado em duas partes:
    - Parte esquerda: chaves ≤ q;
    - Parte direita: chaves ≥ q

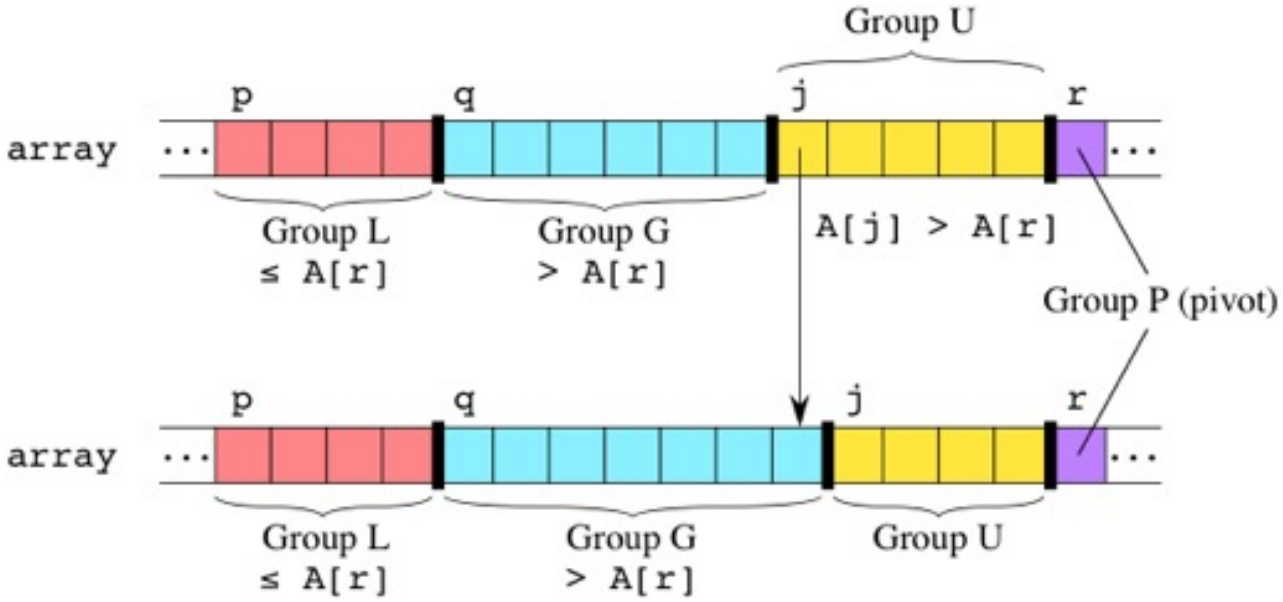
Quicksort

```
procedimento quicksort(A, p, r)
    se p < r então
        q ← particionar(A, p, r);
        quicksort(A, p, q-1);
        quicksort(A, q+1, r);
fim_quicksort
```

- A chamada inicial para o procedimento Quicksort acima é quicksort(A, 1, A.tamanho).

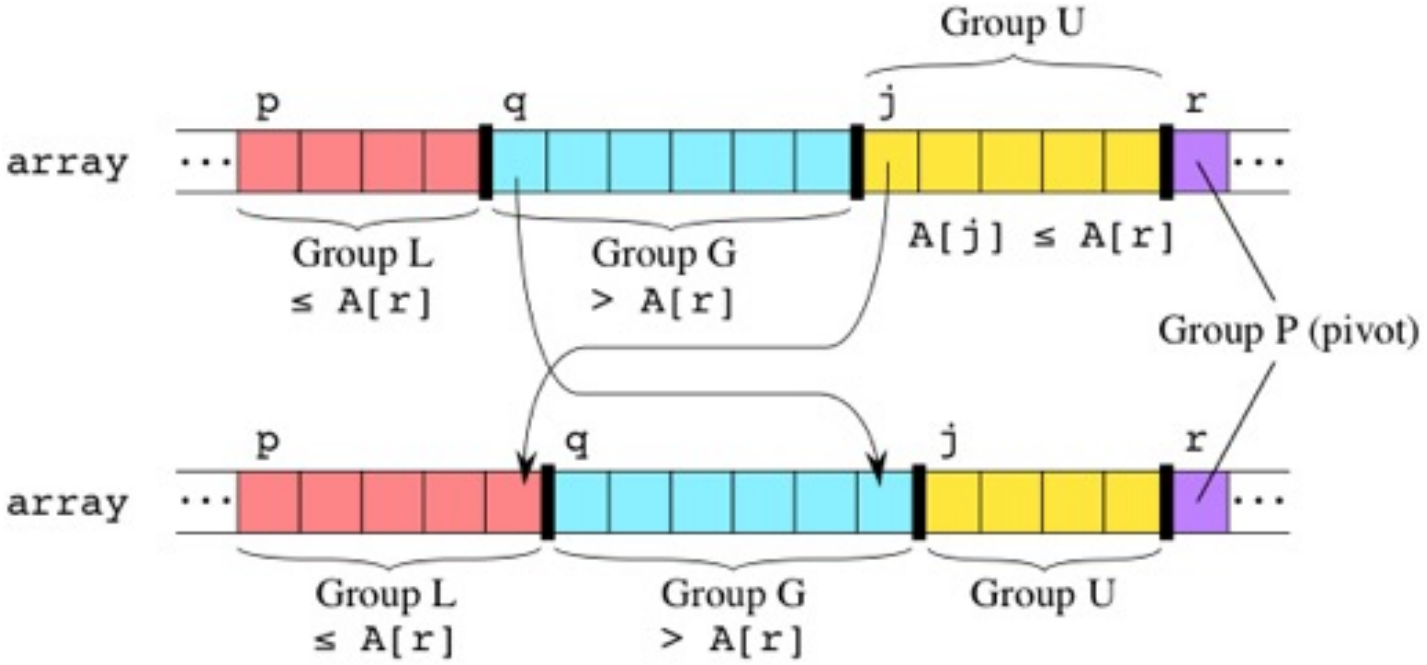
Quicksort

- Particionamento: Ideia Geral



Quicksort

- Particionamento: Ideia Geral

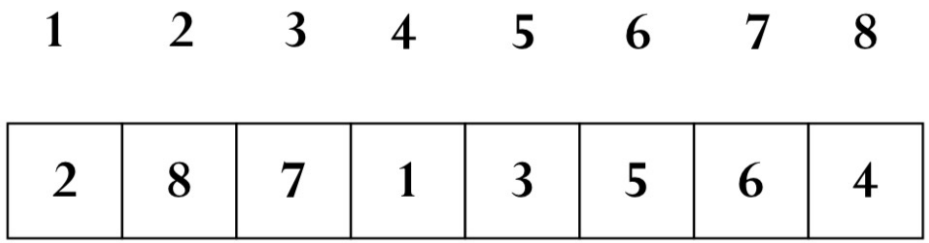


Quicksort

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```

Quicksort

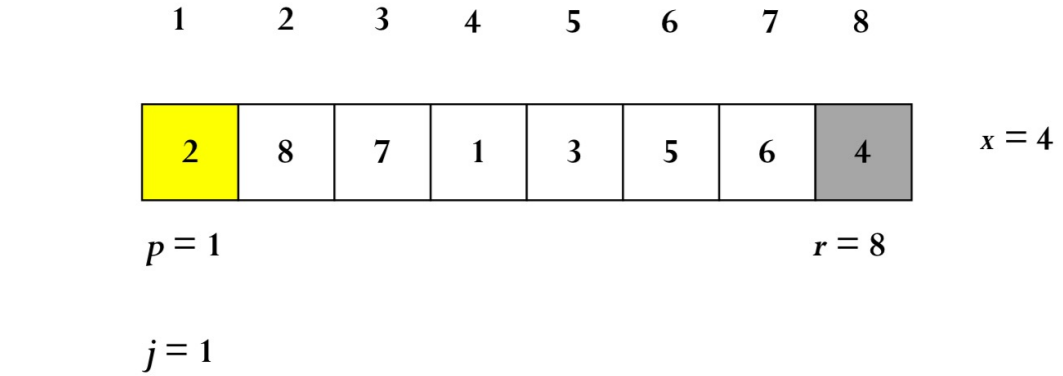
- Exemplo: ordenar o vetor A = {2, 8, 7, 1, 3, 5, 6, 4} usando o algoritmo quicksort com pivô igual ao último elemento de cada partição.



Quicksort

- Particionar(A, 1, 8):

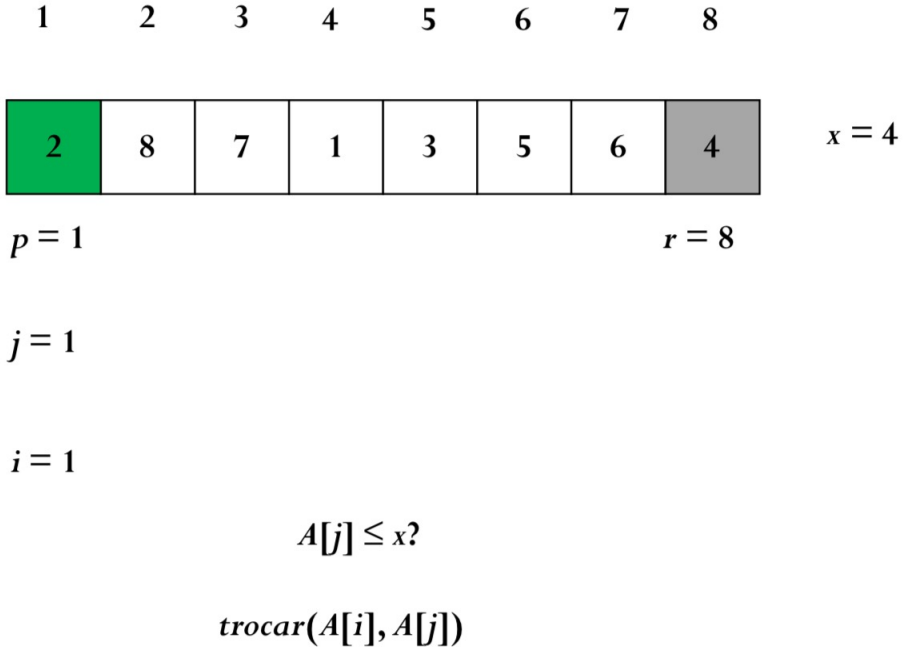
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

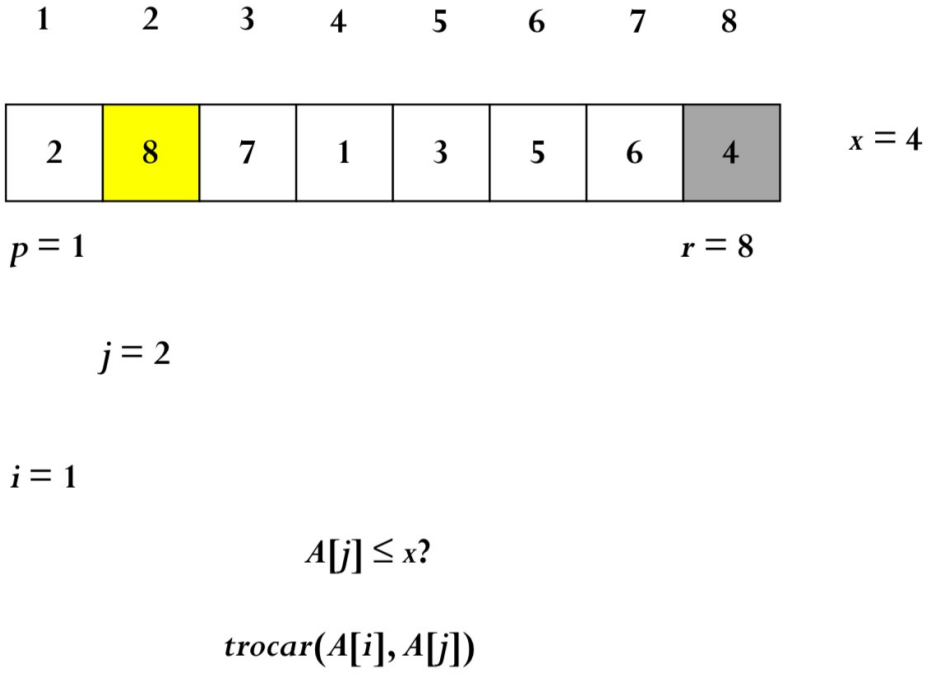
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

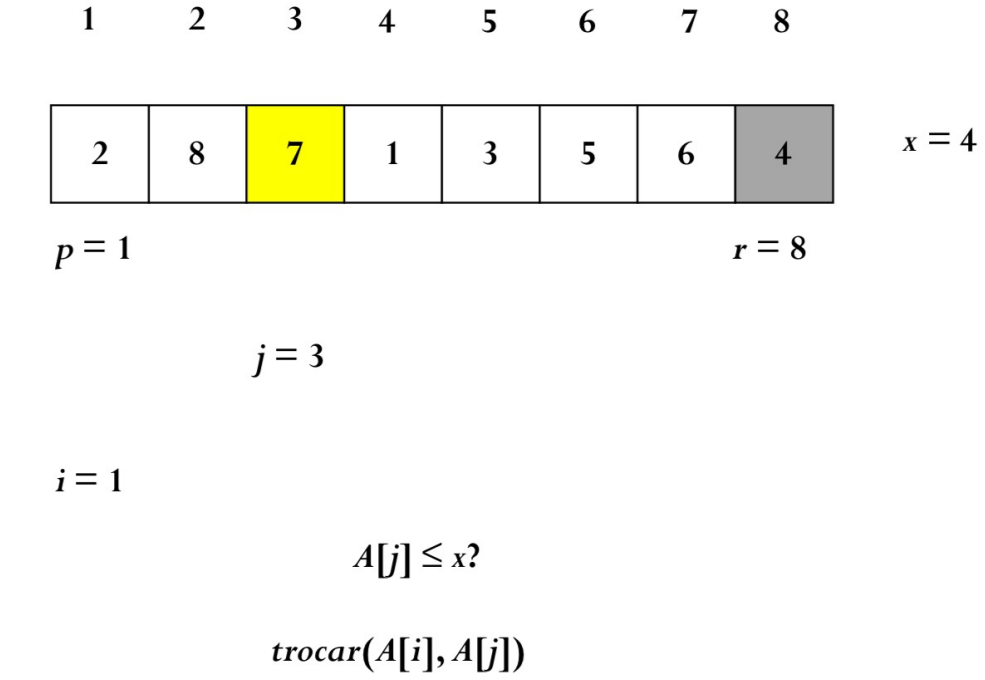
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

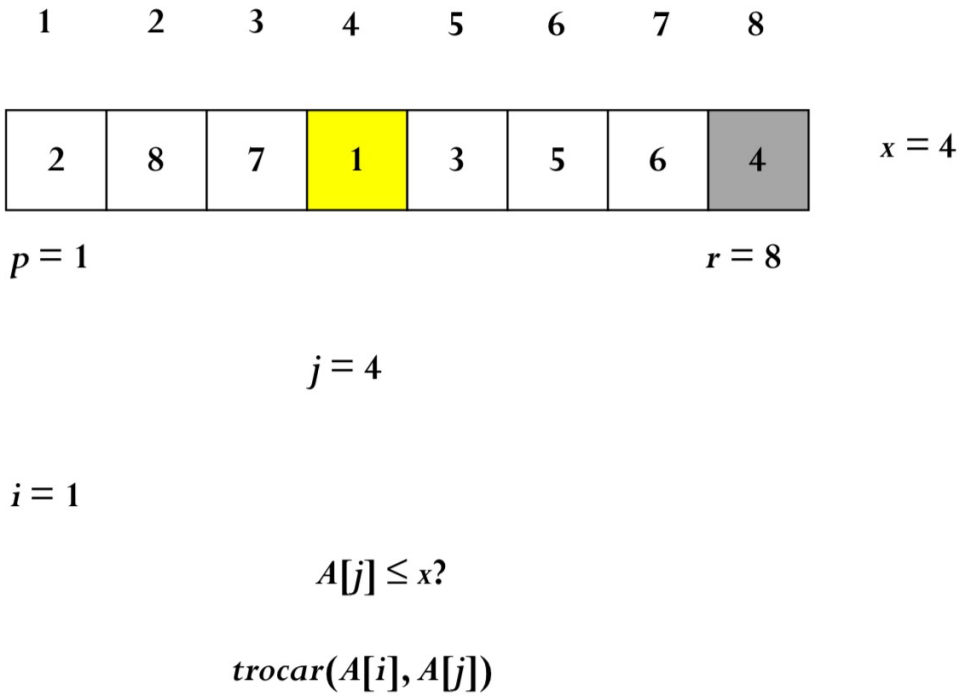
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

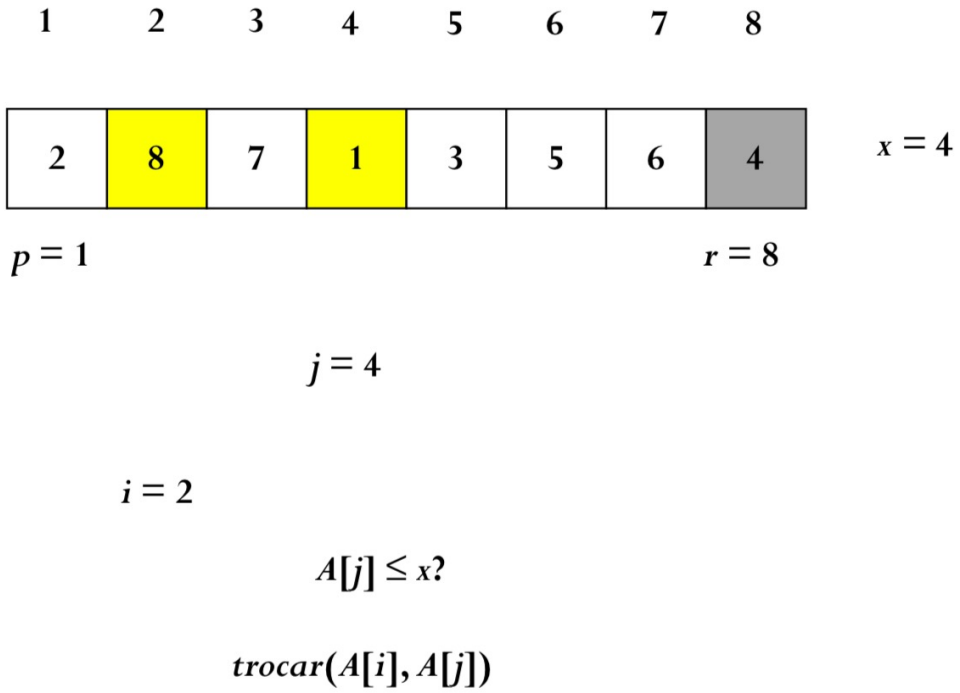
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

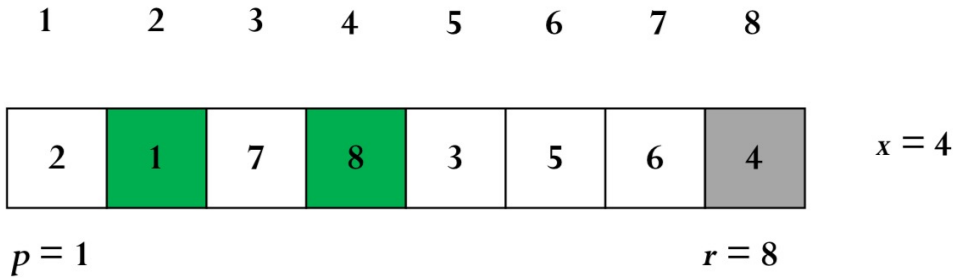
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

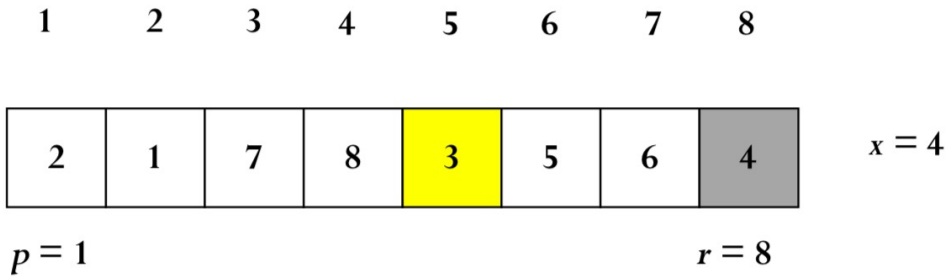
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

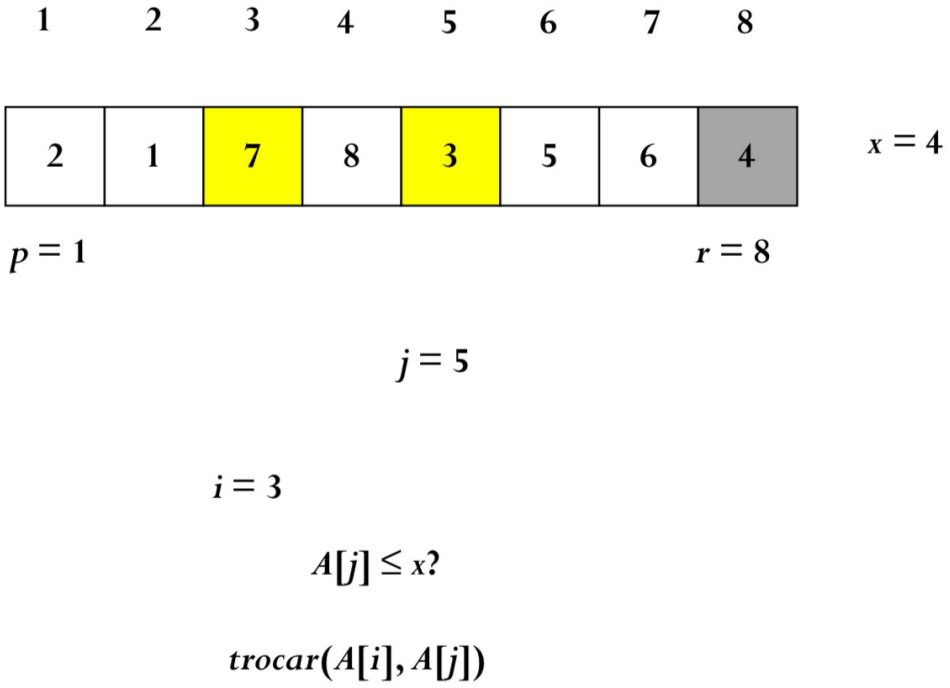
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

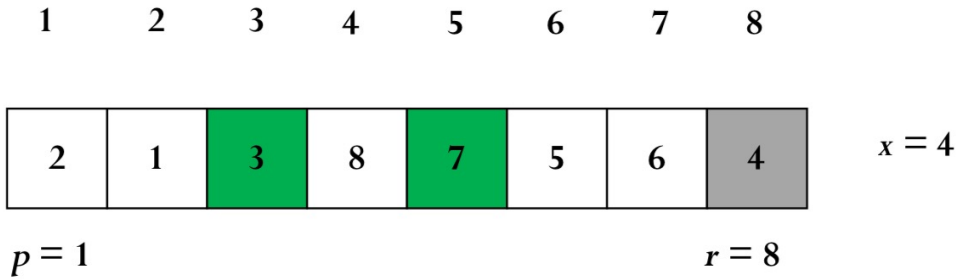
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

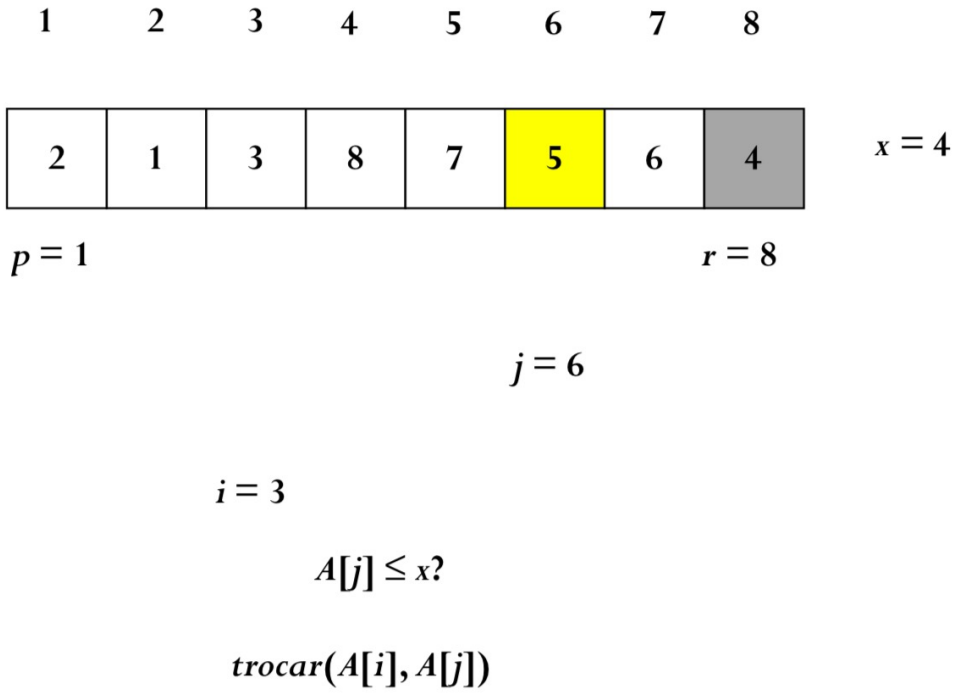
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

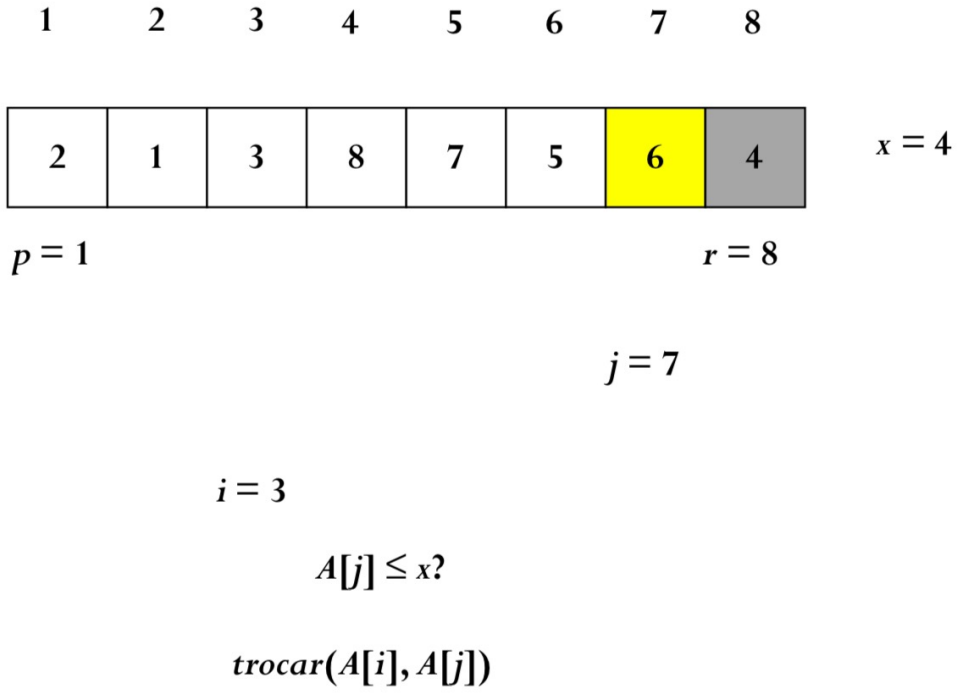
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

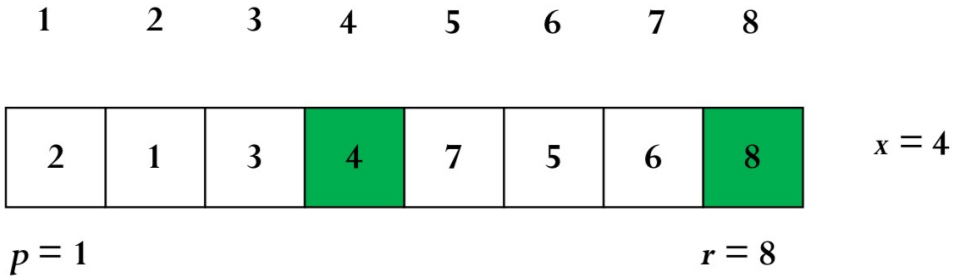
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 8):

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



i = 3

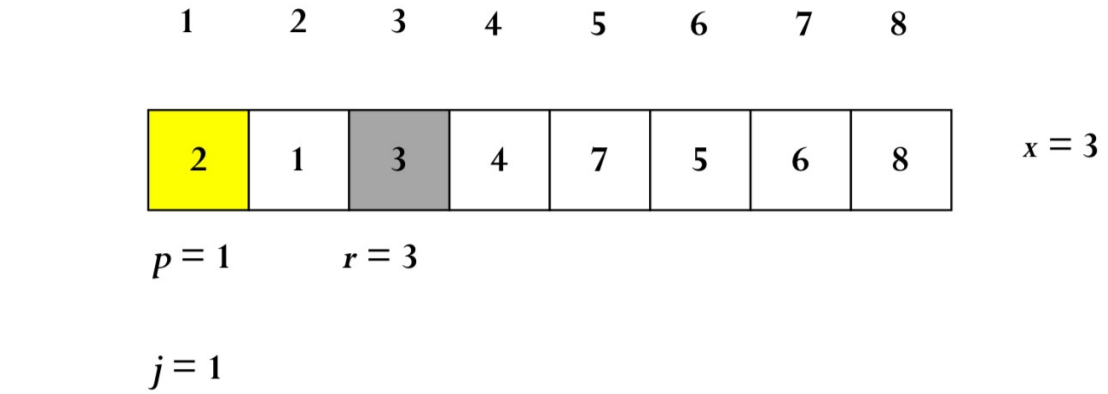
***trocar*(*A*[*i*+*1*], *A*[*r*])**

***retornar i* + 1**

Quicksort

- Particionar(A, 1, 3):

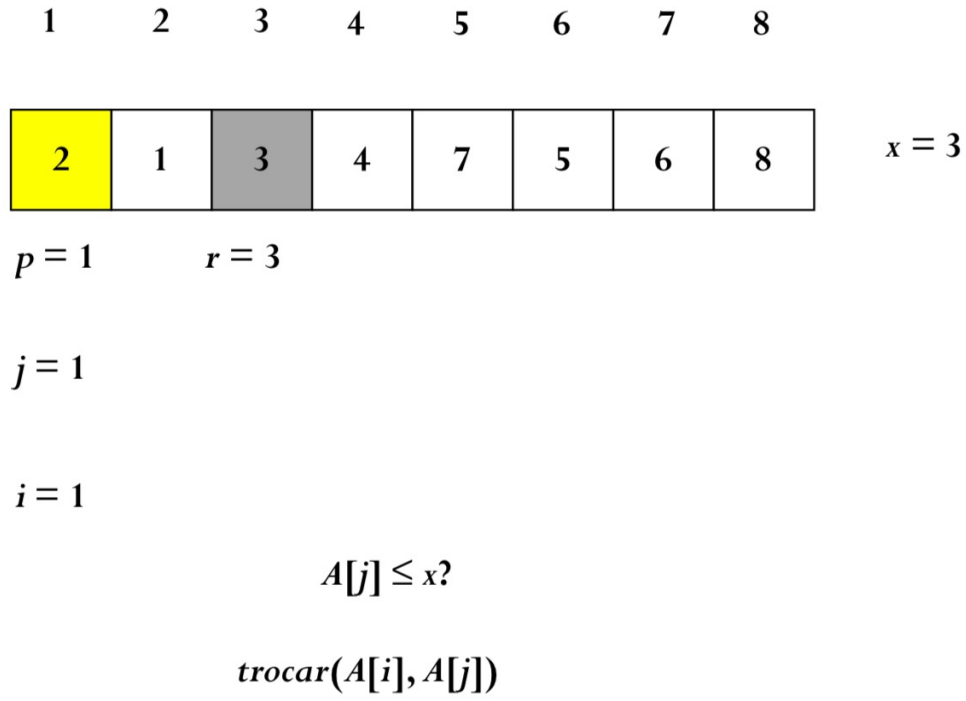
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 3):

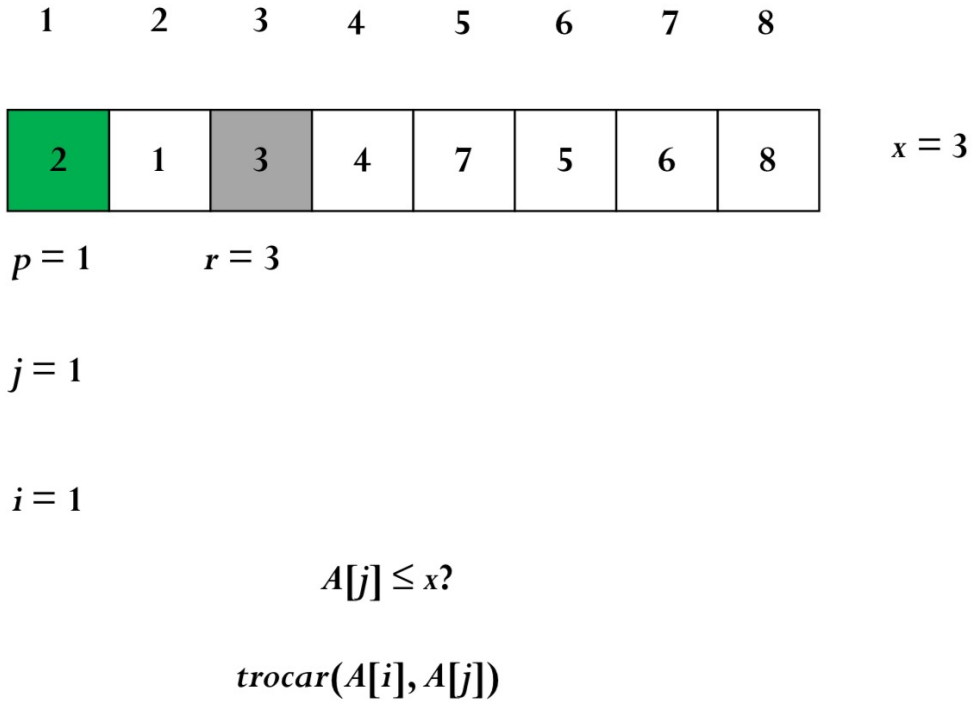
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 3):

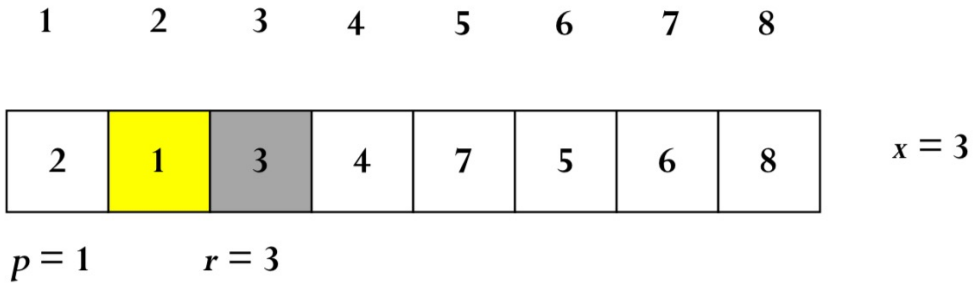
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 3):

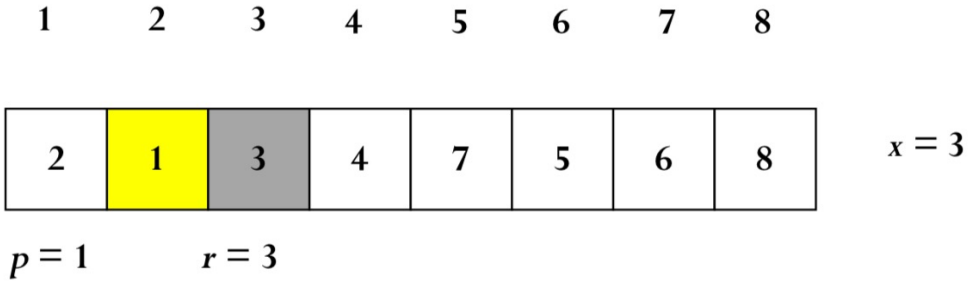
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r – 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 3):

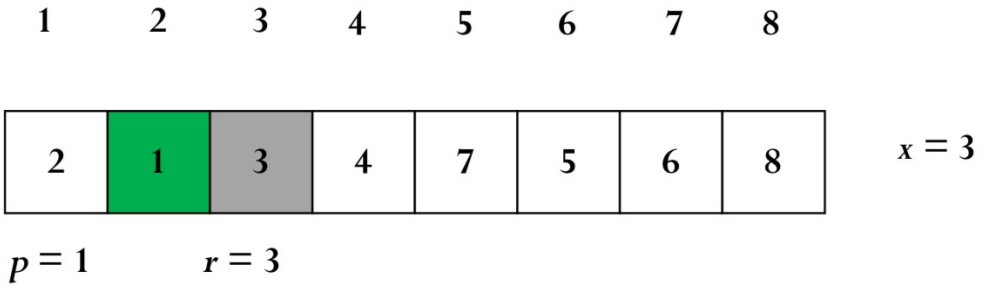
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 1, 3):

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



$j = 2$

$i = 2$

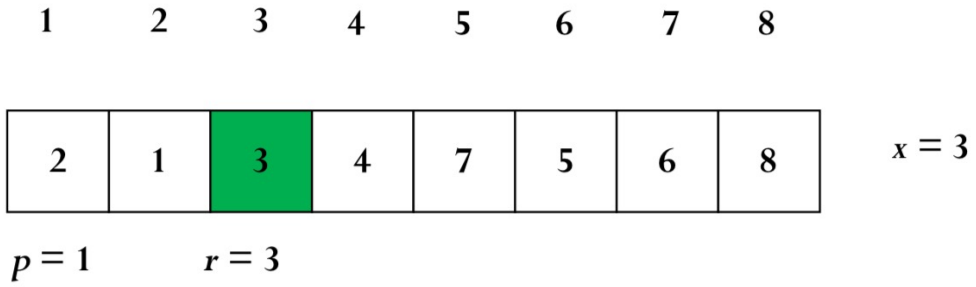
$A[j] \leq x$?

$trocar(A[i], A[j])$

Quicksort

- Particionar(A, 1, 3):

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r – 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



i = 2

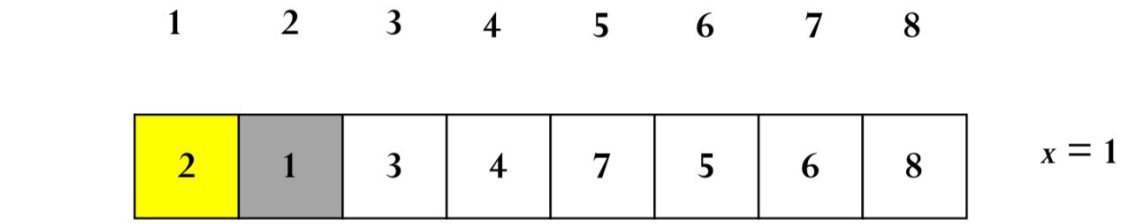
***trocar*(*A*[*i*+*1*], *A*[*r*])**

***retornar i* + 1**

Quicksort

- Particionar(A, 1, 2):

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



$p = 1$ $r = 2$

$j = 1$

$i = 0$

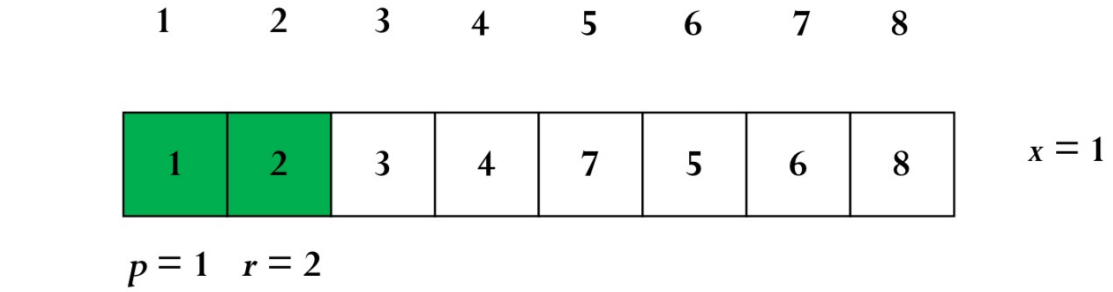
$A[j] \leq x$?

$trocar(A[i], A[j])$

Quicksort

- Particionar(A, 1, 2):

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



i = 0

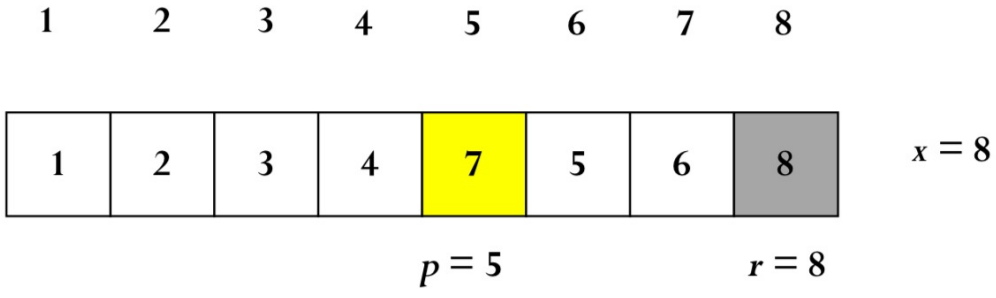
***trocar*(*A*[*i*+*1*], *A*[*r*])**

***retornar i* + 1**

Quicksort

- Particionar(A, 5, 8):

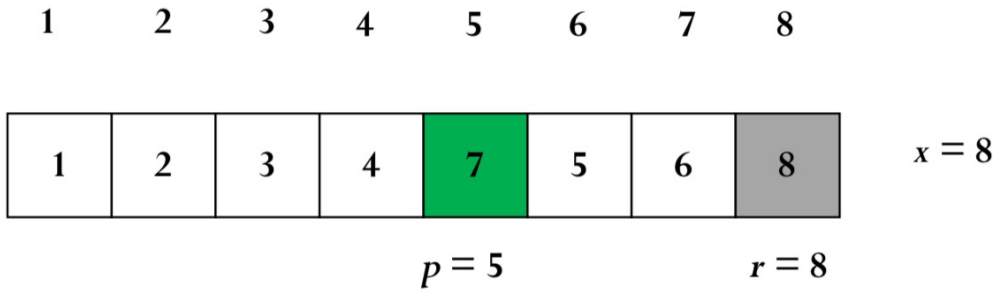
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 5, 8):

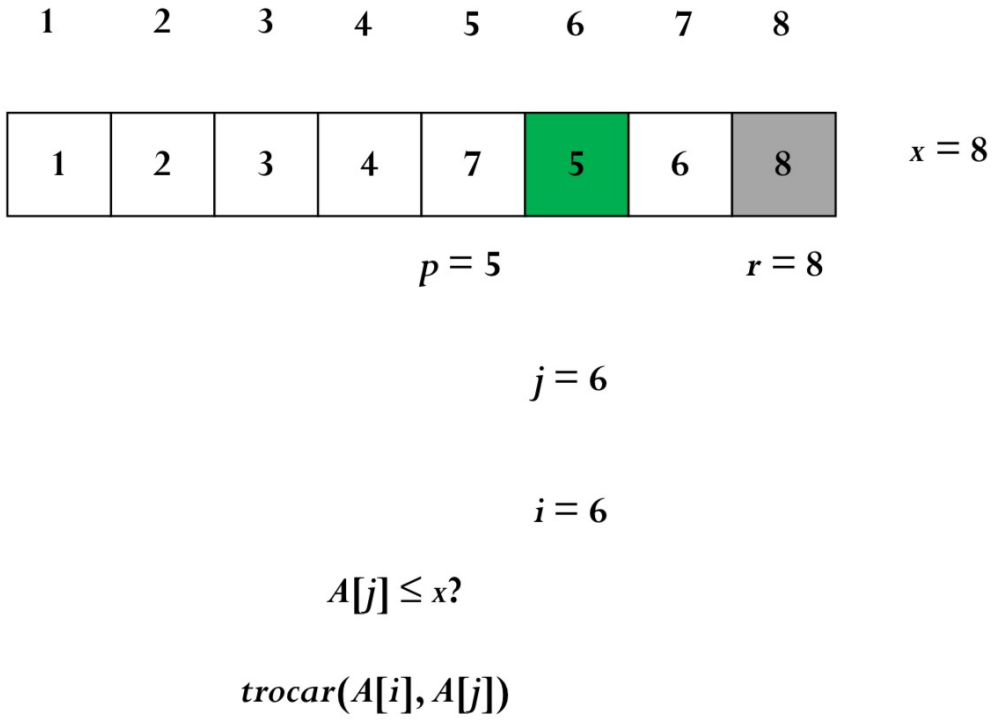
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 5, 8):

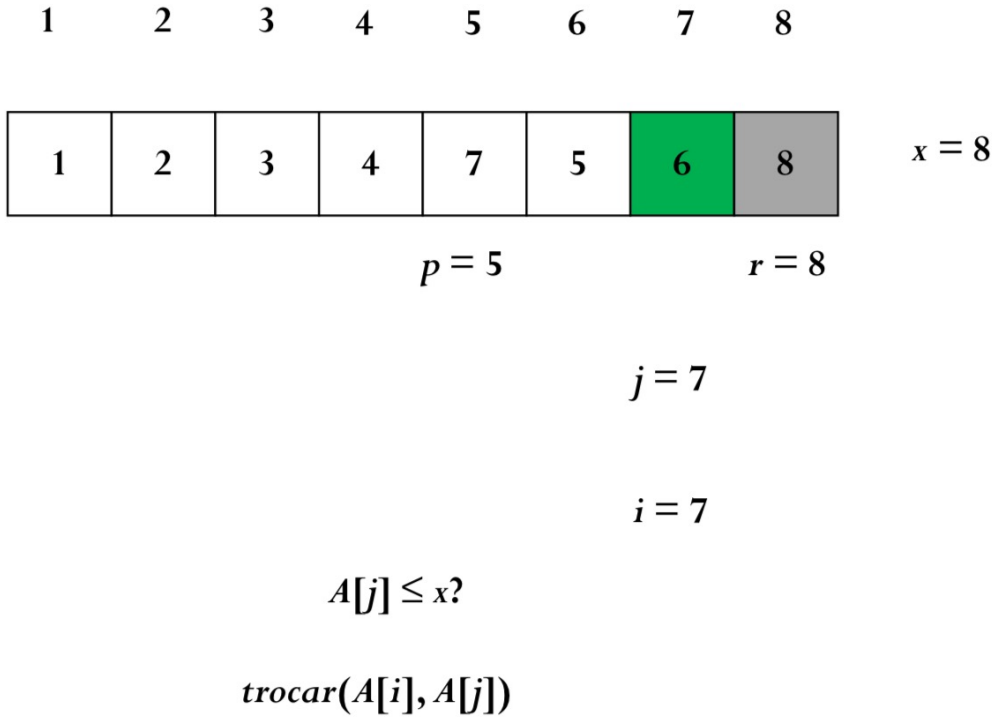
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 5, 8):

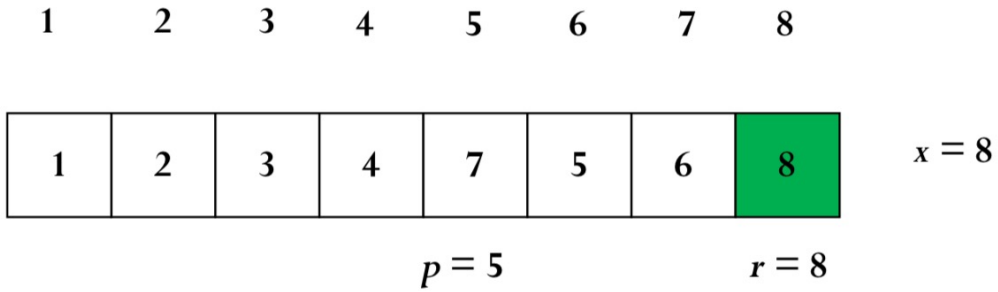
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 5, 8):

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



$i = 7$

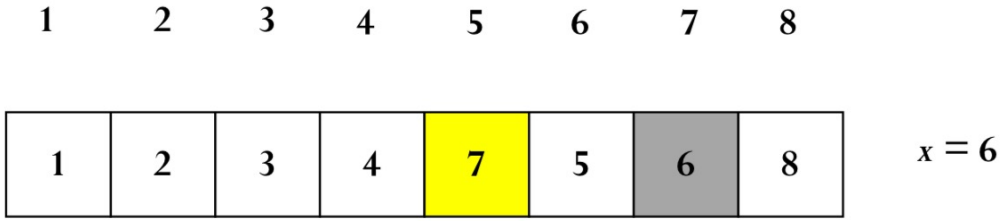
trocar(A[i+1], A[r])

retornar i + 1

Quicksort

- Particionar(A, 5, 7):

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



$j = 5$

$i = 4$

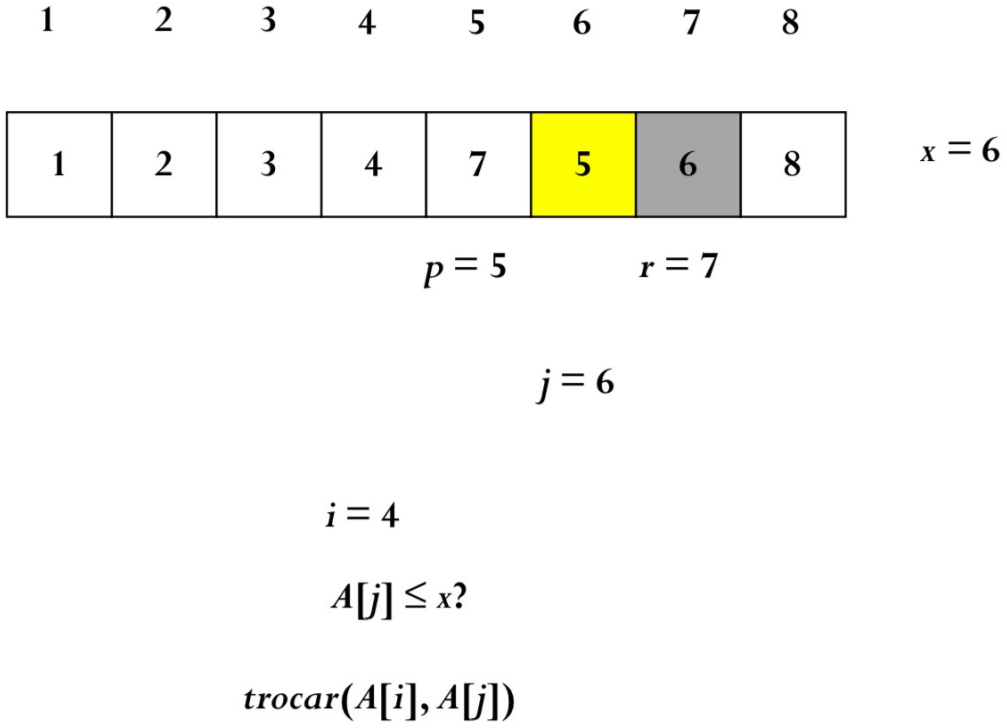
$A[j] \leq x$?

$trocar(A[i], A[j])$

Quicksort

- Particionar(A, 5, 7):

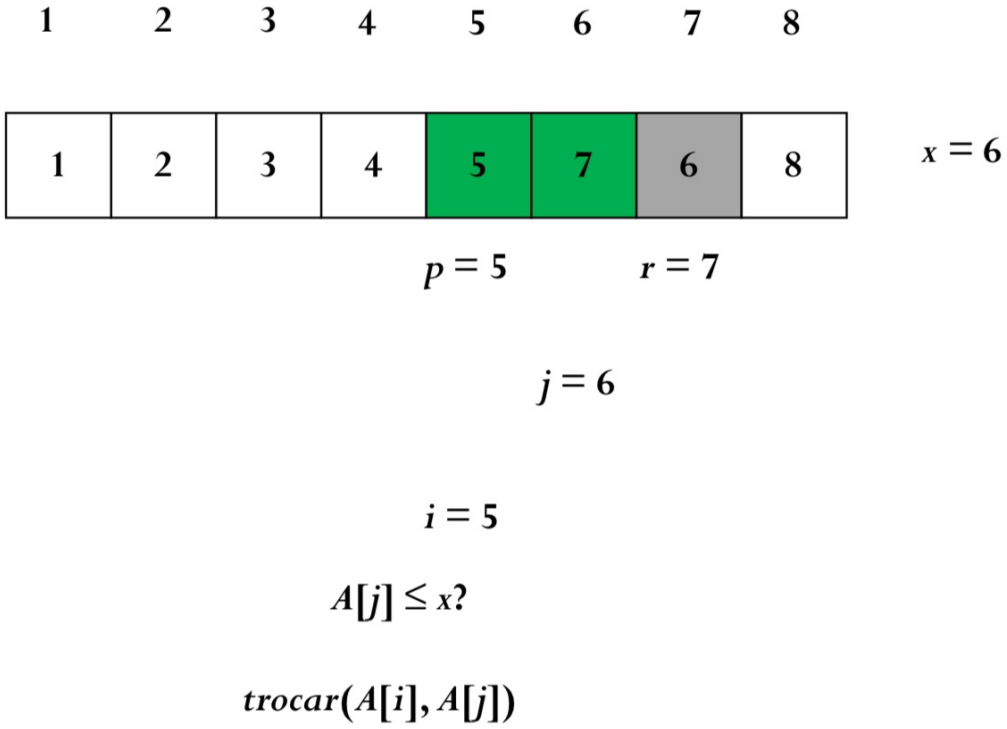
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 5, 7):

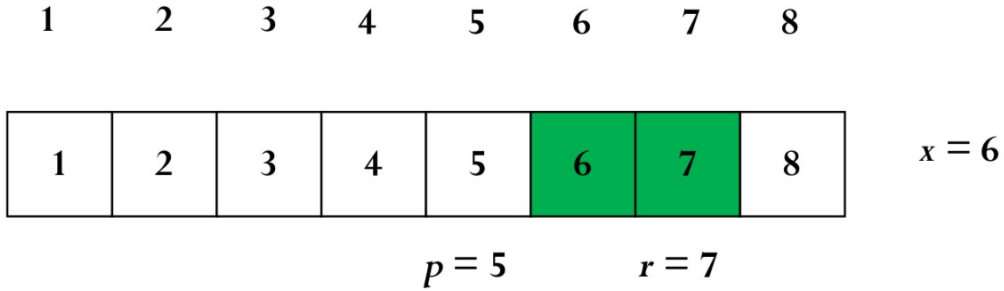
```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



Quicksort

- Particionar(A, 5, 7):

```
procedimento particionar(A, p, r)
    x ← A[r];
    i ← p − 1;
    para j ← p até r − 1 faça
        se A[j] ≤ x então
            i ← i + 1;
            trocar(A[i], A[j]);
    fim_para
    trocar(A[i+1], A[r]);
    retorne i + 1;
fim_particionar
```



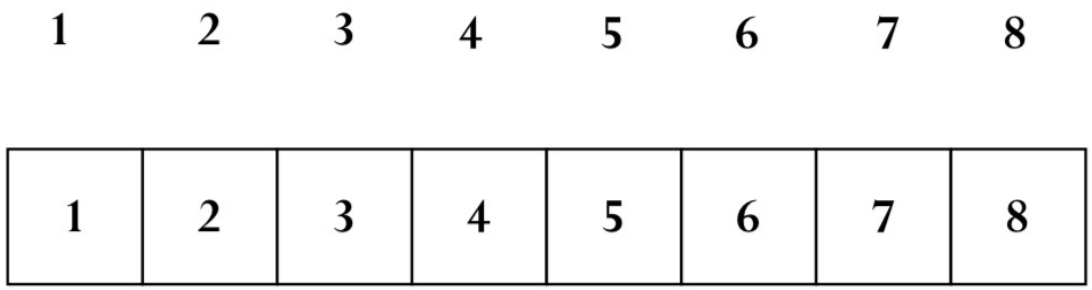
i = 5

trocar(*A*[*i*+1], *A*[*r*])

retornar i + 1

Quicksort

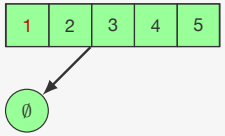
- Não há mais nenhuma chamada recursiva



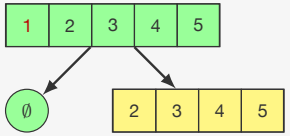
Pior caso do QuickSort

Pior caso do QuickSort

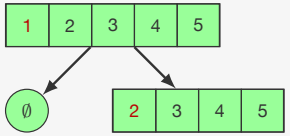
Pior caso do QuickSort



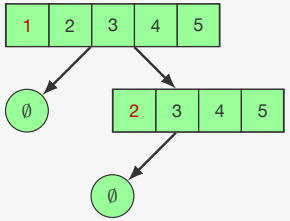
Pior caso do QuickSort



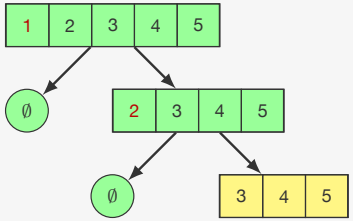
Pior caso do QuickSort



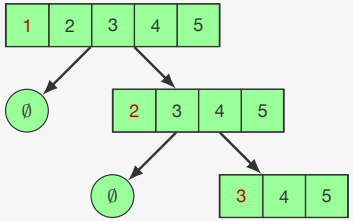
Pior caso do QuickSort



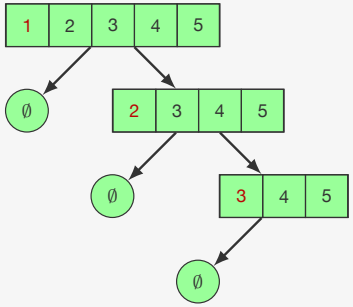
Pior caso do QuickSort



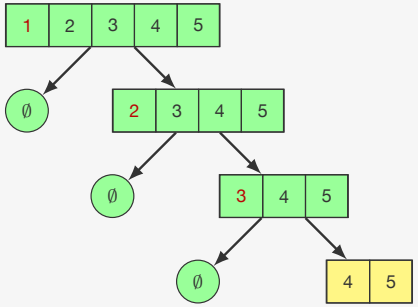
Pior caso do QuickSort



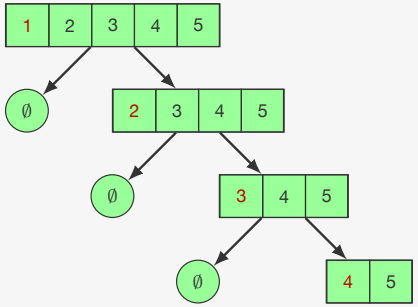
Pior caso do QuickSort



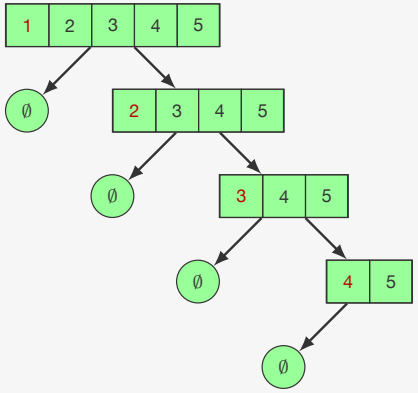
Pior caso do QuickSort



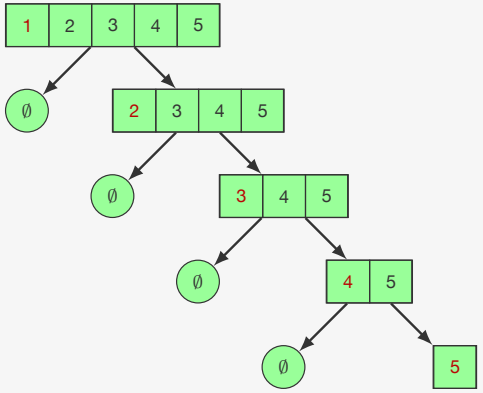
Pior caso do QuickSort



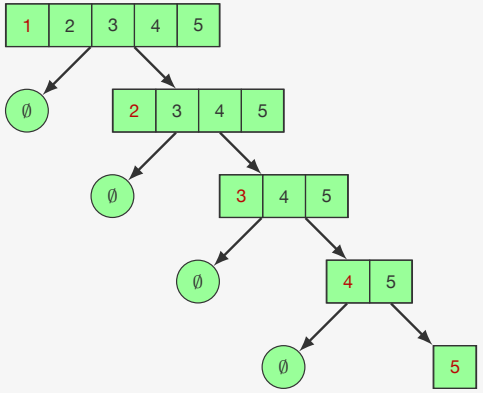
Pior caso do QuickSort



Pior caso do QuickSort

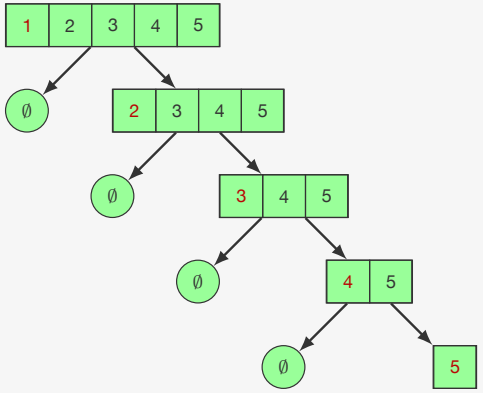


Pior caso do QuickSort



$c \cdot n$

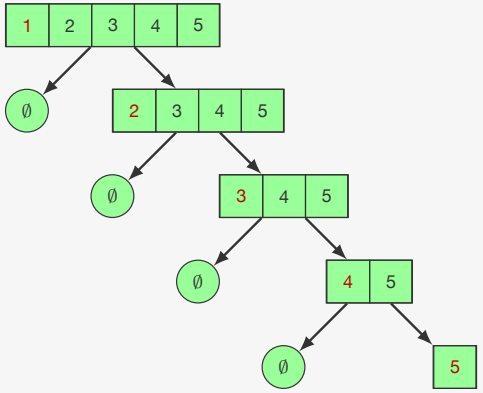
Pior caso do QuickSort



$c \cdot n$

$c \cdot (n-1)$

Pior caso do QuickSort

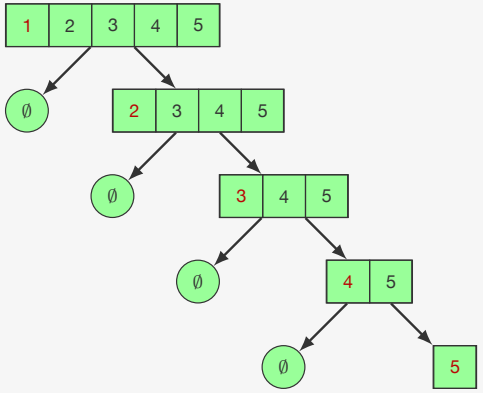


$c \cdot n$

$c \cdot (n-1)$

$c \cdot (n-2)$

Pior caso do QuickSort



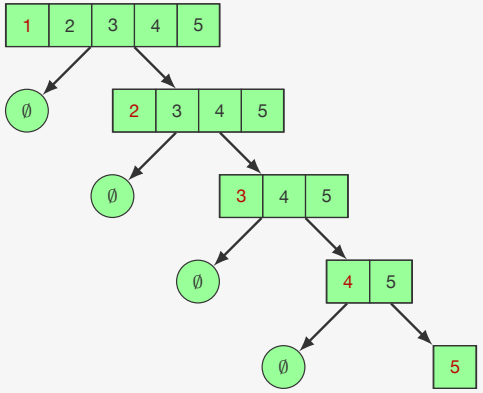
$c \cdot n$

$c \cdot (n-1)$

$c \cdot (n-2)$

$\cdots$

Pior caso do QuickSort



$c \cdot n$

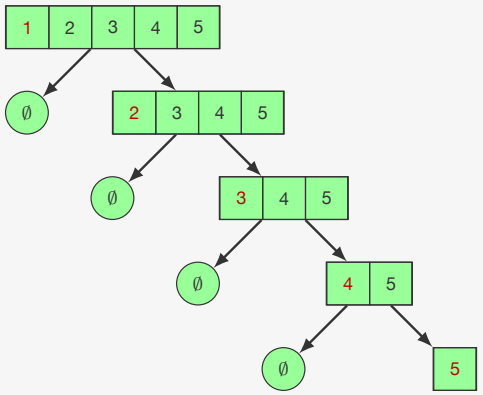
$c \cdot (n-1)$

$c \cdot (n-2)$

$\cdots$

c

Pior caso do QuickSort



$c \cdot n$

$c \cdot (n-1)$

$c \cdot (n-2)$

$\cdots$

c

O tempo de execução do Quicksort é, no pior caso:

$$c{\cdot}n{+}c{\cdot}(n{-}1){+}\cdots{+}c = c\sum_{i=0}^{n-1}(n-i) = c\sum_{j=1}^{n} j = c\frac{n(n+1)}{2} = \mathrm{O}(n^2)$$

Desempenho no Melhor Caso

- Se sempre escolher elemento do meio
  - O(n log n)
- Caso Médio:
  - O(n log n)

Quicksort

- Considerações de implementação:
- Como a escolha do pivô é crucial para um bom desempenho do algoritmo, as implementações usam métodos mais sofisticados para a sua escolha, sendo os mais populares:
    - Mediana(i[p], A[(p+r)/2], A[r])
    - Aleatório (p..r)

Parte 3

Leitura

Ler material do Paulo Feofiloff:
https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/quick.html

Atividade

1. Implemente o Algoritmo Quicksort visto em sala
2. Adapte o algoritmo para o pivô ser o primeiro element