Algoritmos de ordenação – parte II

Prof. Dr. Wesin Ribeiro

Nessa aula você irá aprender

- O que é recursão
- Estratégia divisão e conquista
- Ordenação mergesort
- Ordenação quicksort
- Analisar a complexidade dos algoritmos

Recursão

O que é recursão ?

A recursão deixa o código mais elegante

- Sem garantia de melhora de desempenho
- Os loops as vezes são mais rápidos
- “os loops melhoram o desempenho do programa, a recursão melhora o desempenho do programador”. (Leigh Caldwell, Stackoverflow)

Uma função recursiva chama a si mesma

- Perigo de criar loops infinitos
- Caso base
- Caso recursivo

Exemplo clássico de recursividade

- Calcular o Fatorial de n
- Caso base
  - !0 ou !1 => 1
- Caso recursivo
  - !n = n * !(n-1)

Merge sort

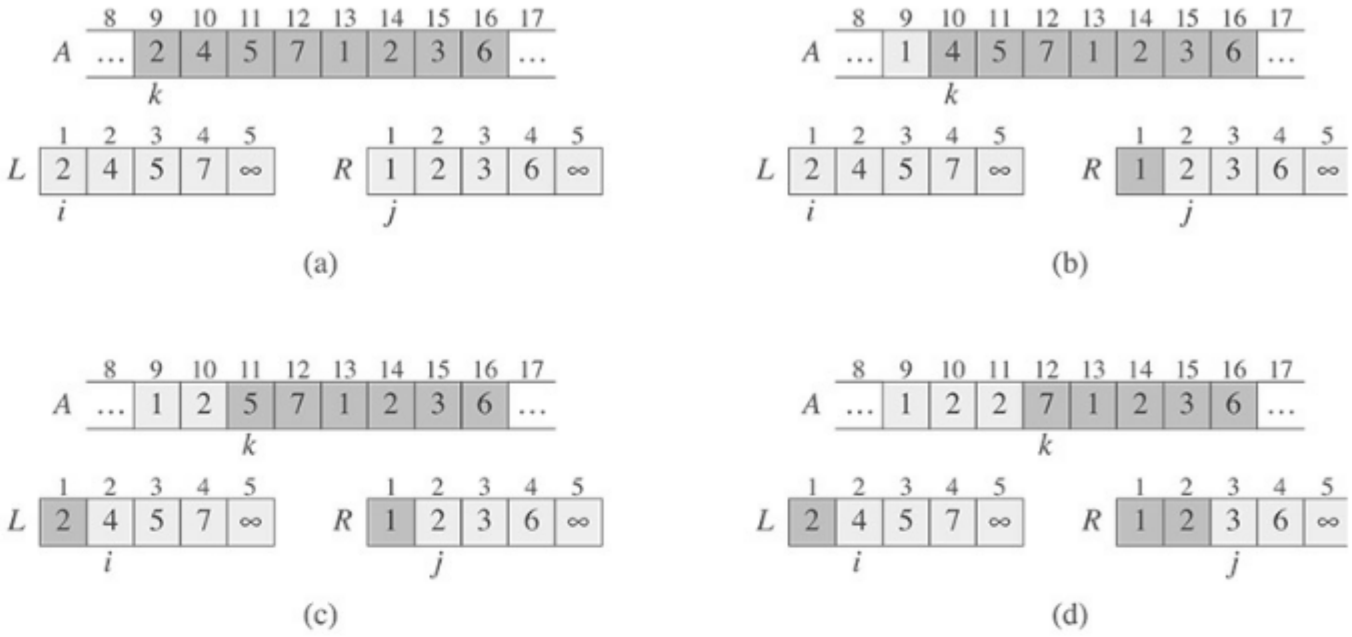
Merge sort

- Criado em 1945 pelo matemático americano John Von Neumann
- faz uso da estratégia “dividir para conquistar” para resolver problemas

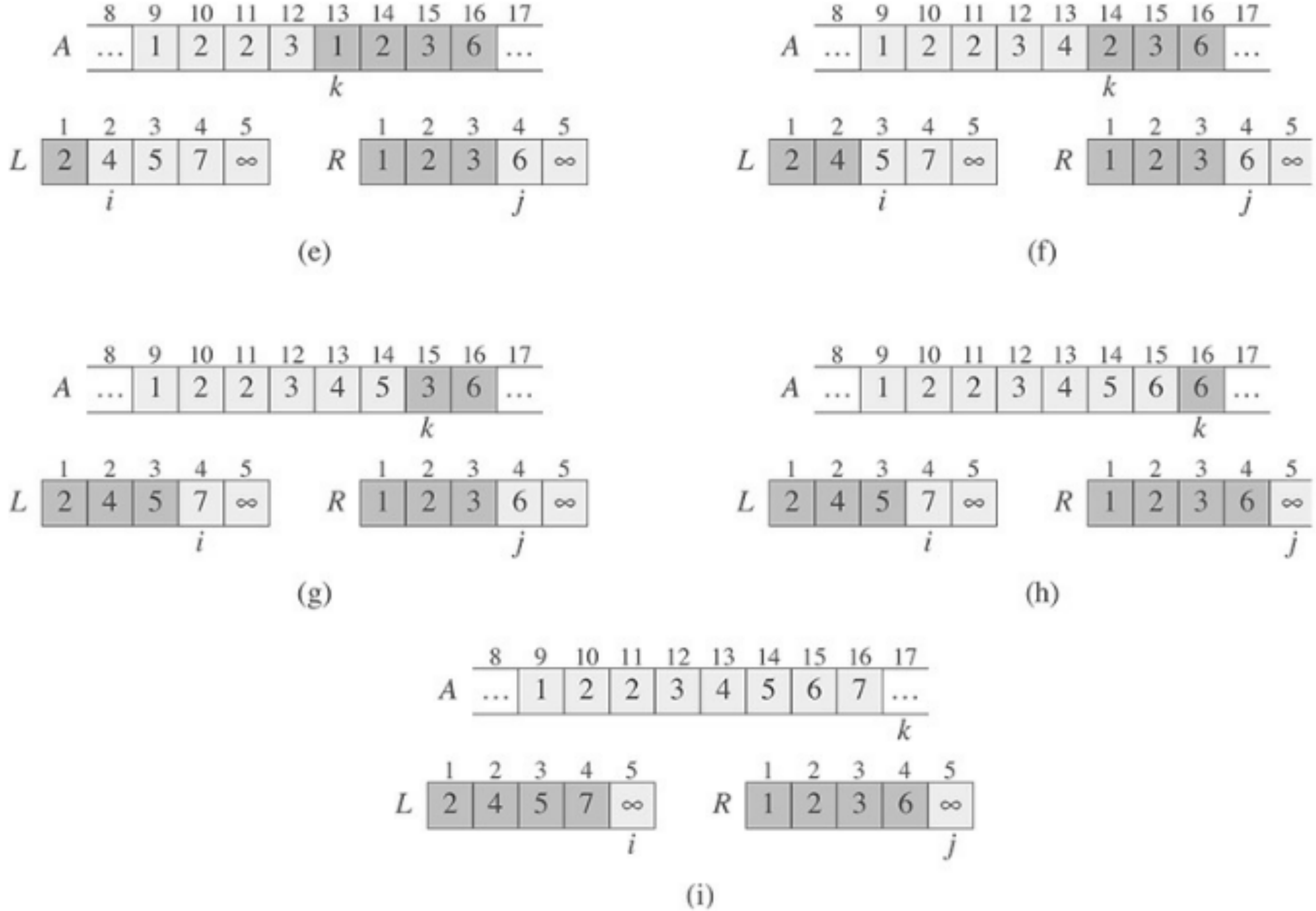
A estratégia divisão e conquista do mergesort:

- Divisão
  - Divide o conjunto de n elementos em dois subconjuntos de n/2 elementos
- Conquista
  - Ordena cada subconjunto recursivamente, intercalando os elementos
- Combinação
  - Intercala os dois subconjuntos ordenados

Considerando um subconjunto A



Considerando um subconjunto A



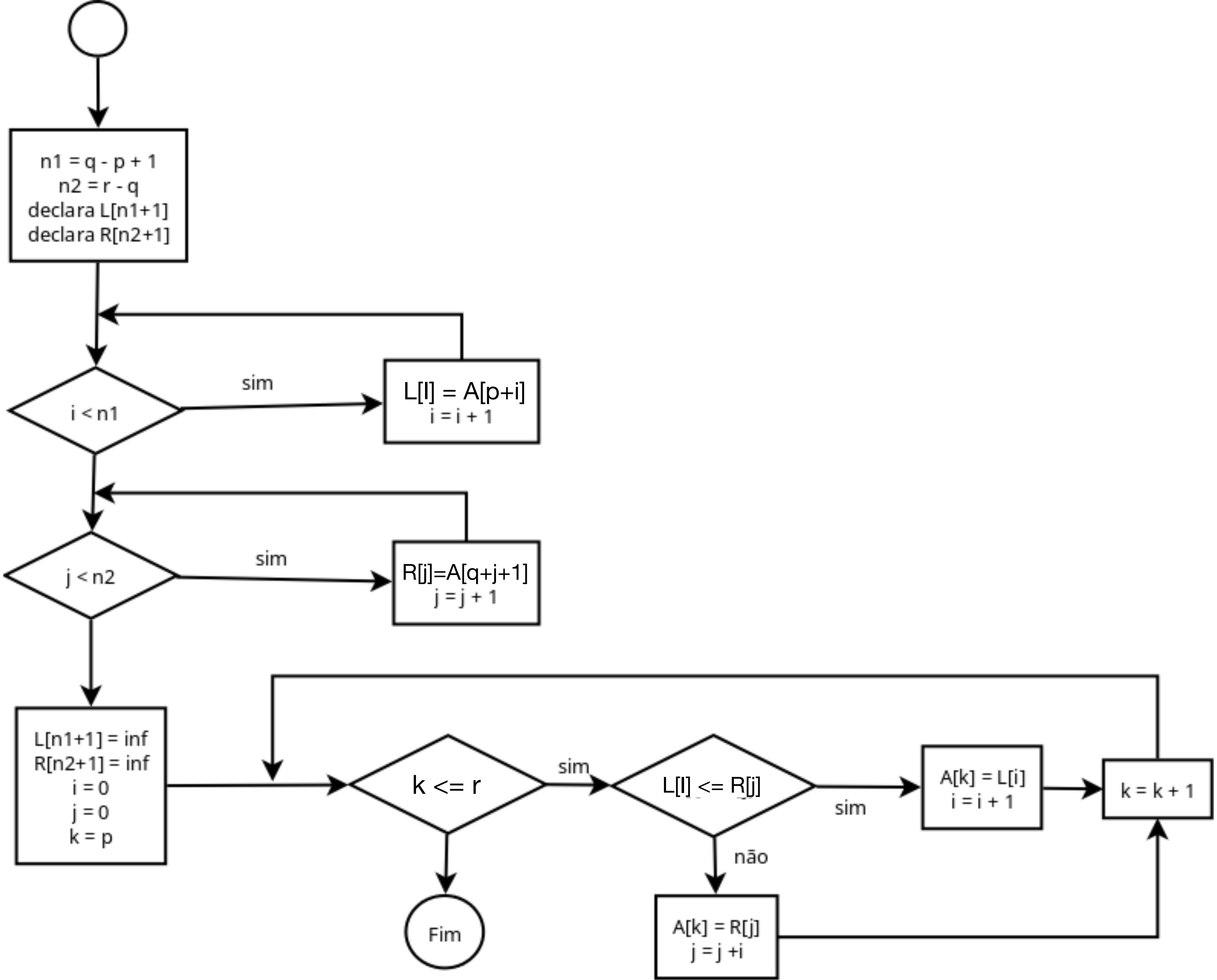
Pseudo código

```
MERGE(A, p, q, r)
n1 = q − p + 1
n2 = r − q
sejam L[1..n1 + 1] e R[1.. n2 + 1] novos arranjos
for i = 1 to n1
    L[i] = A[p + i − 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        then A[k] = L[i]
             i = i + 1
        else A[k] = R[j]
             j = j + 1
```

```
MERGE-SORT(A, p, r)
if p < r
    then q = ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

A[1..n]

Fluxograma da operação merge



Tempo de execução

- Melhor caso
  - Quando os elementos já estão ordenados
  - om(n log n)
- Pior caso
  - Quando os elementos estão ordenados na ordem inversa
  - O(n log n)

Quicksort

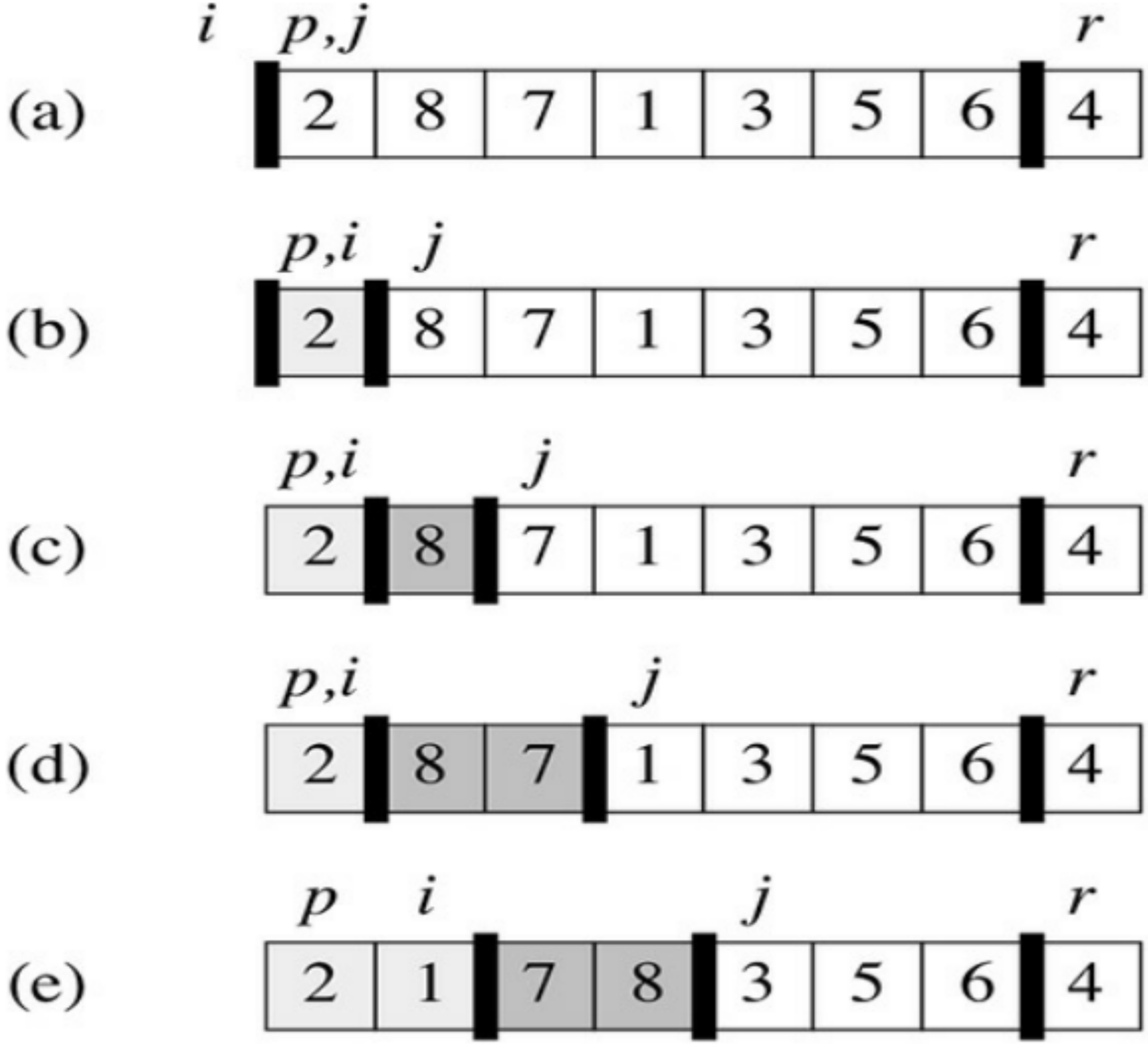
Quicksort

- É o algoritmo mais eficiente
- Possui um elemento central (pivô)
- Todos os elementos a esquerda são menores que o pivô
- Todos os elementos a direita são maiores que o pivô
- Esse processo se repete de maneira recursiva até que o conjunto seja ordenado

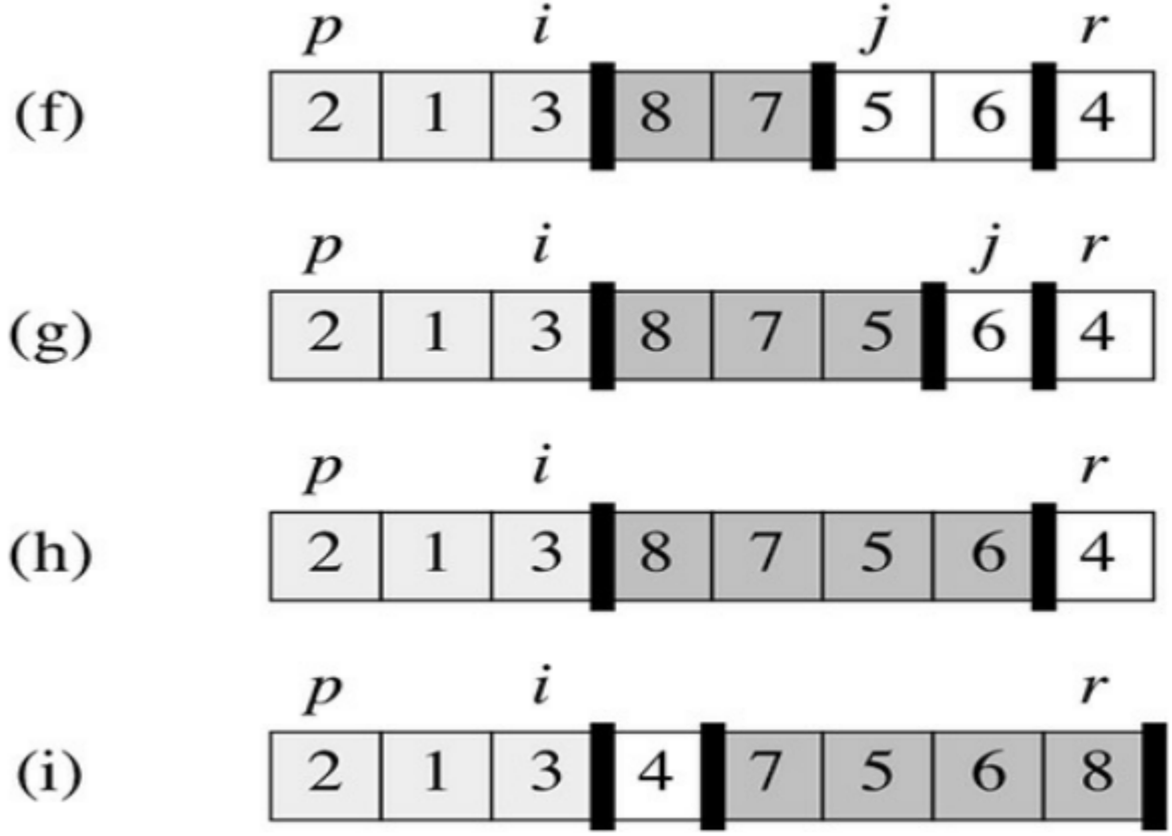
A estratégia divisão e conquista do quicksort:

- Divisão
  - Particionar o arranjo A[p...r], tal que A[p..q-1] < A[q] < A[q+1 .. r].
  - Calcular o índice q
- Conquista
  - Ordena os subconjuntos A[p..q-1] e A[q+1..r] por chamadas recursivas
- Combinação
  - Não é necessário nenhum trabalho para combinar os subconjuntos pois já estão ordenados

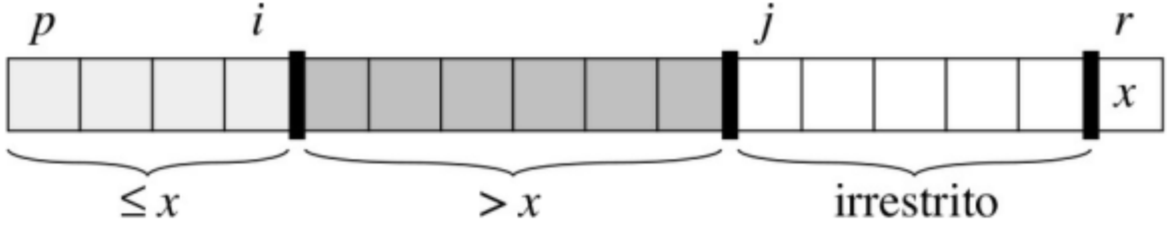
Exemplo da operação partition



Exemplo da operação partition

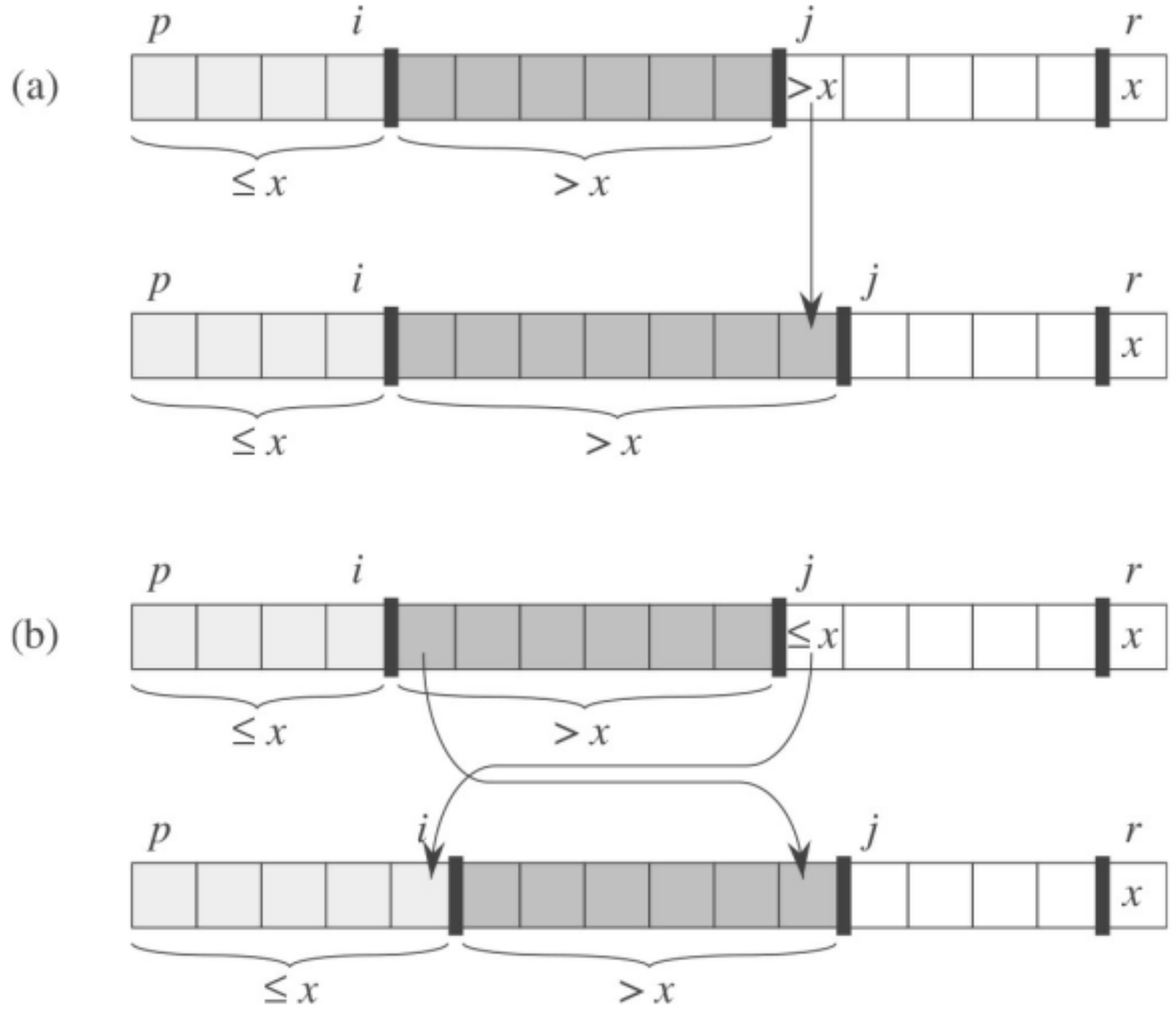


Regiões mantidas pelo particionamento



Se p <= k <=i, então A[k] <= x
Se i+i <= k <= j-1, então A[k] > x
Se k = r, então A[k] = x

Os dois casos para uma iteração procedimento partition

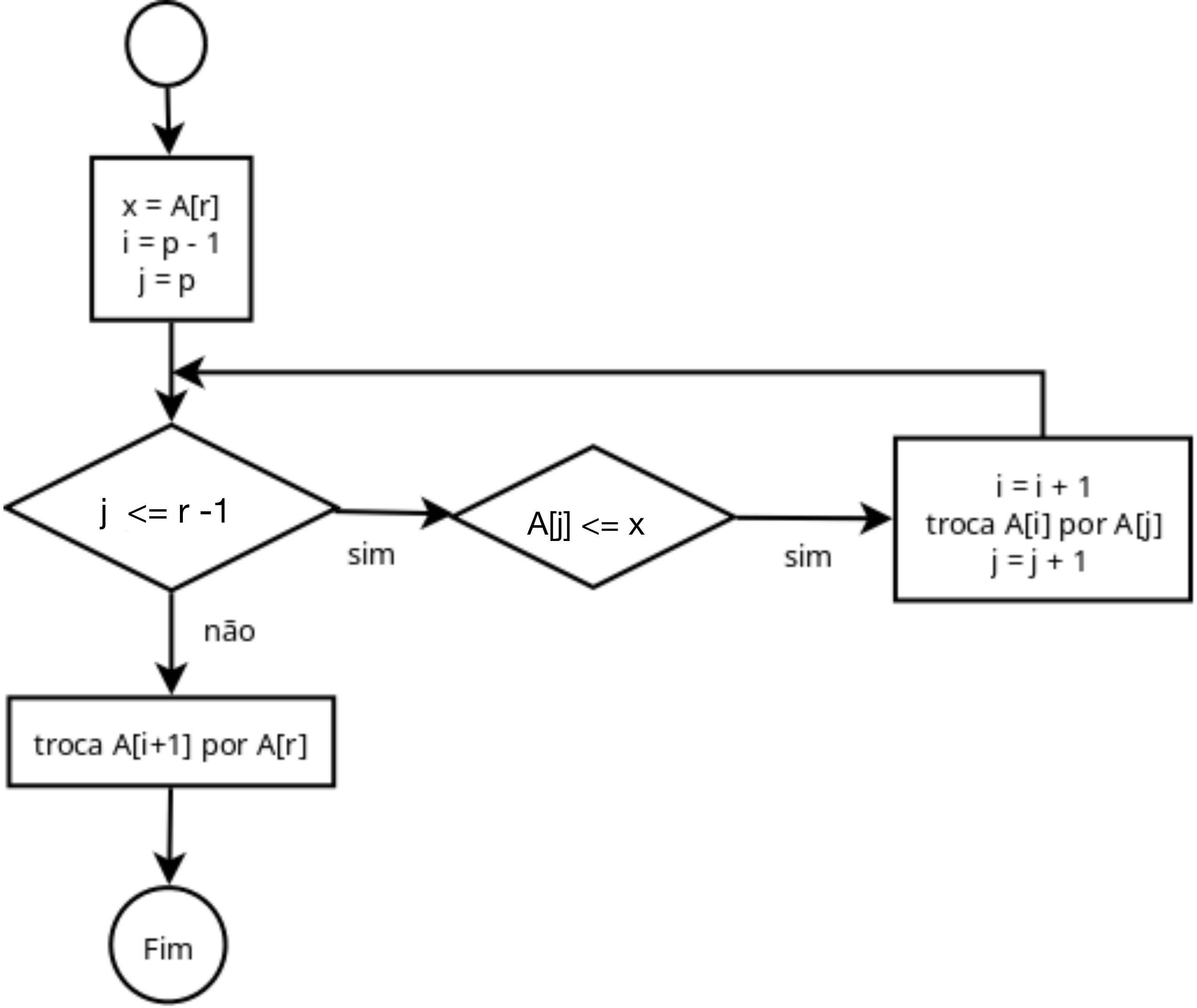


Pseudo código

```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        trocar A[i] por A[j]
trocar A[i + 1] por A[r]
return i + 1
```

```
QUICKSORT(A, p, r)
if p < r
    q = PARTITION(A, p, r)
    QUICKSORT(A, p, q − 1)
    QUICKSORT(A, q + 1, r)
```

Fluxograma de partition



Tempo de execução

- Melhor caso
  - Quando os elementos divididos em um subconjunto de tamanho n/2 e outro de tamanho n/2 - 1
  - om(n log n)
- Pior caso
  - Quando o particionamento produz um subconjunto com n-1 elementos e outro com 0 elementos.
  - O(n^2) (raro)

Desafio

- Modifique o mergesort e o quicksort para ordenar de forma decresente.