Estrutura de Dados

Prof. Rafael Nunes

Algoritmos de Busca e Ordenação

Search and Sorting Algorithms

Parte 2

Ordenação

Algoritmos de Ordenação Interna

Continuação

Algoritmos de Ordenação Interna

- Complexidade ***O(n^2)***
  - Bubble Sort – Método da Bolha
  - Insertion Sort – Inserção direta
  - Selection Sort – Seleção direta
  - Shell Sort – Incrementos Decrescentes
- Complexidade ***O(n log n)***
  - Merge Sort – Metodo da Intercalação
  - Quick Sort – Método da Troca e Partição
  - Heap Sort – Seleção em Árvore

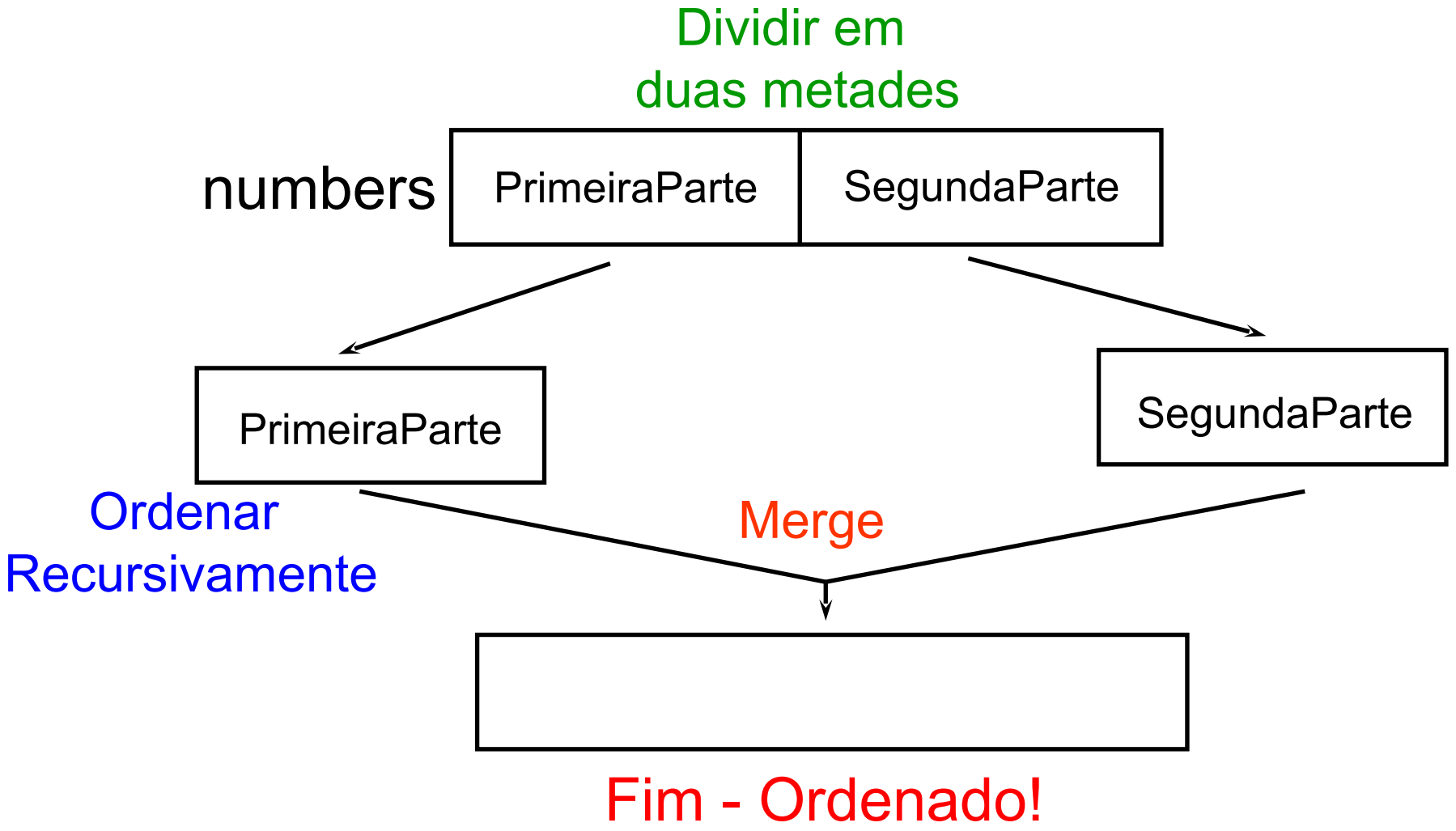
Algoritmos de Ordenação Interna

Complexidade O(n log n)

Merge Sort

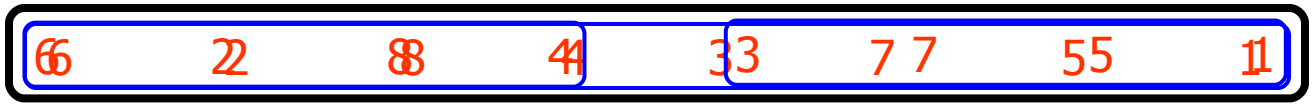
Método da intercalação

Merge Sort: Fucionamento



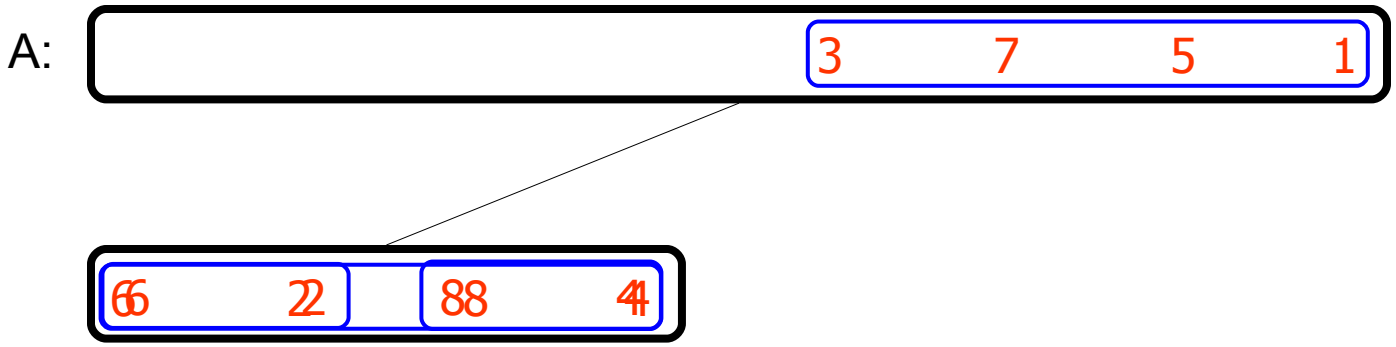
Merge-Sort(numbers, 0, 7)

Divide

A: 

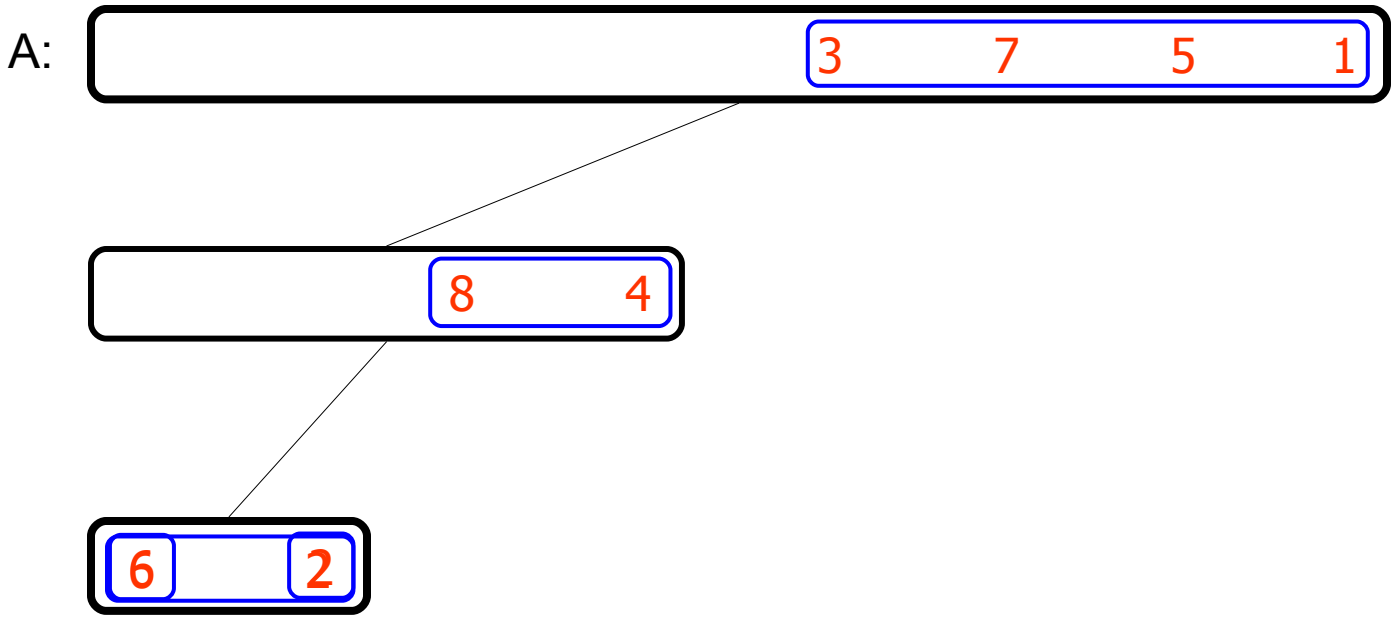
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 0, 3) , divide



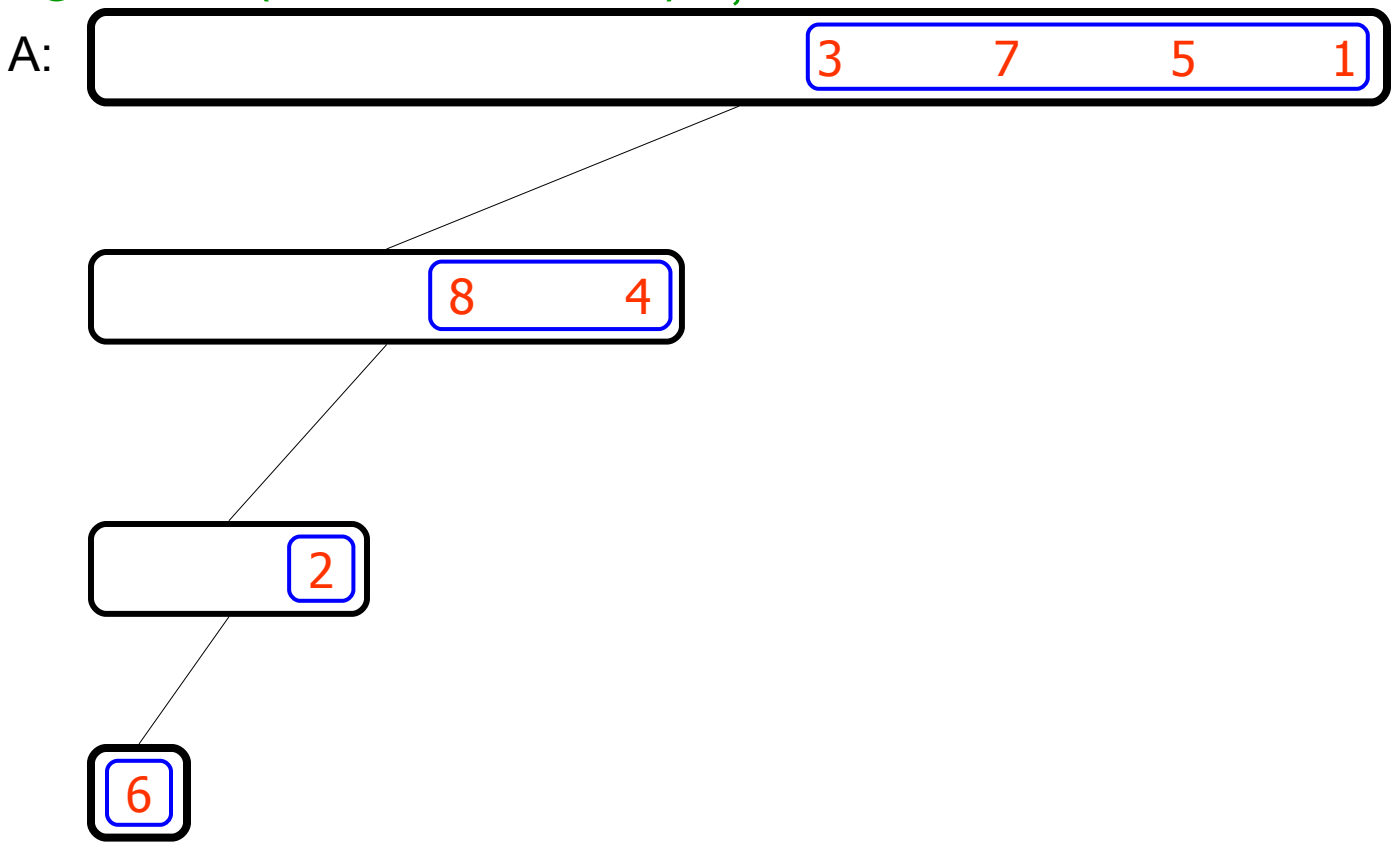
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 0, 1) , divide



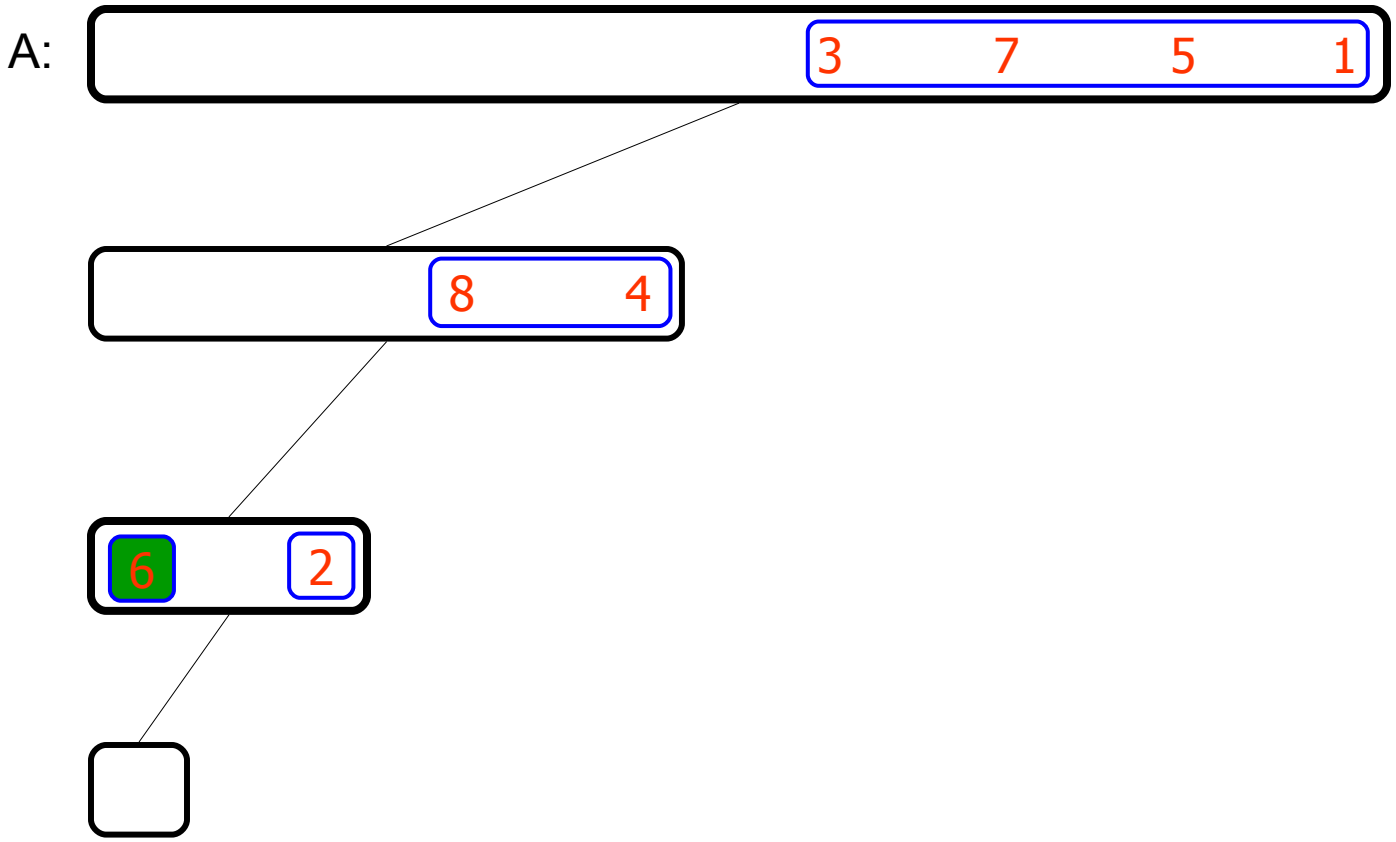
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 0, 0) , caso base



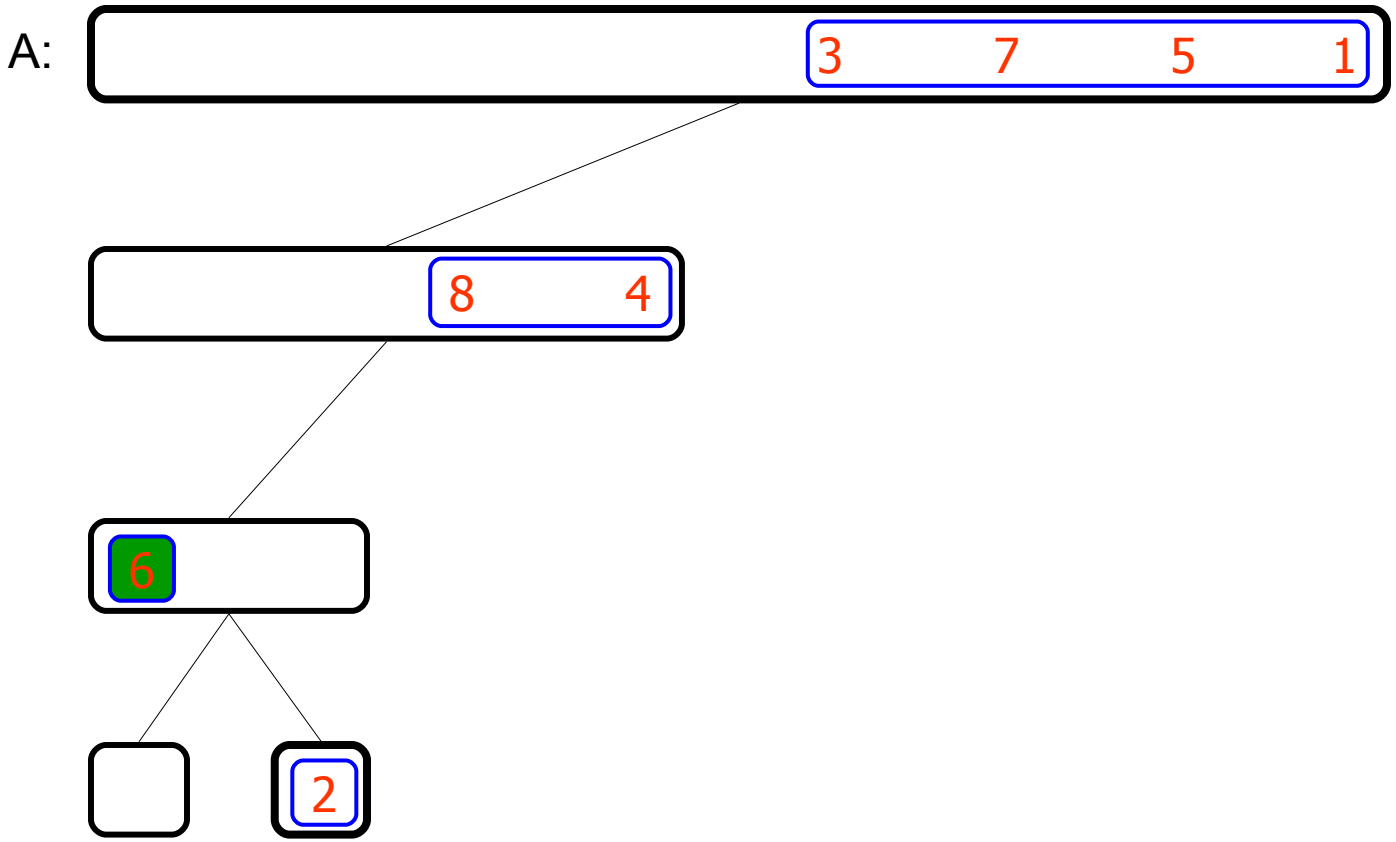
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 0, 0), retorna



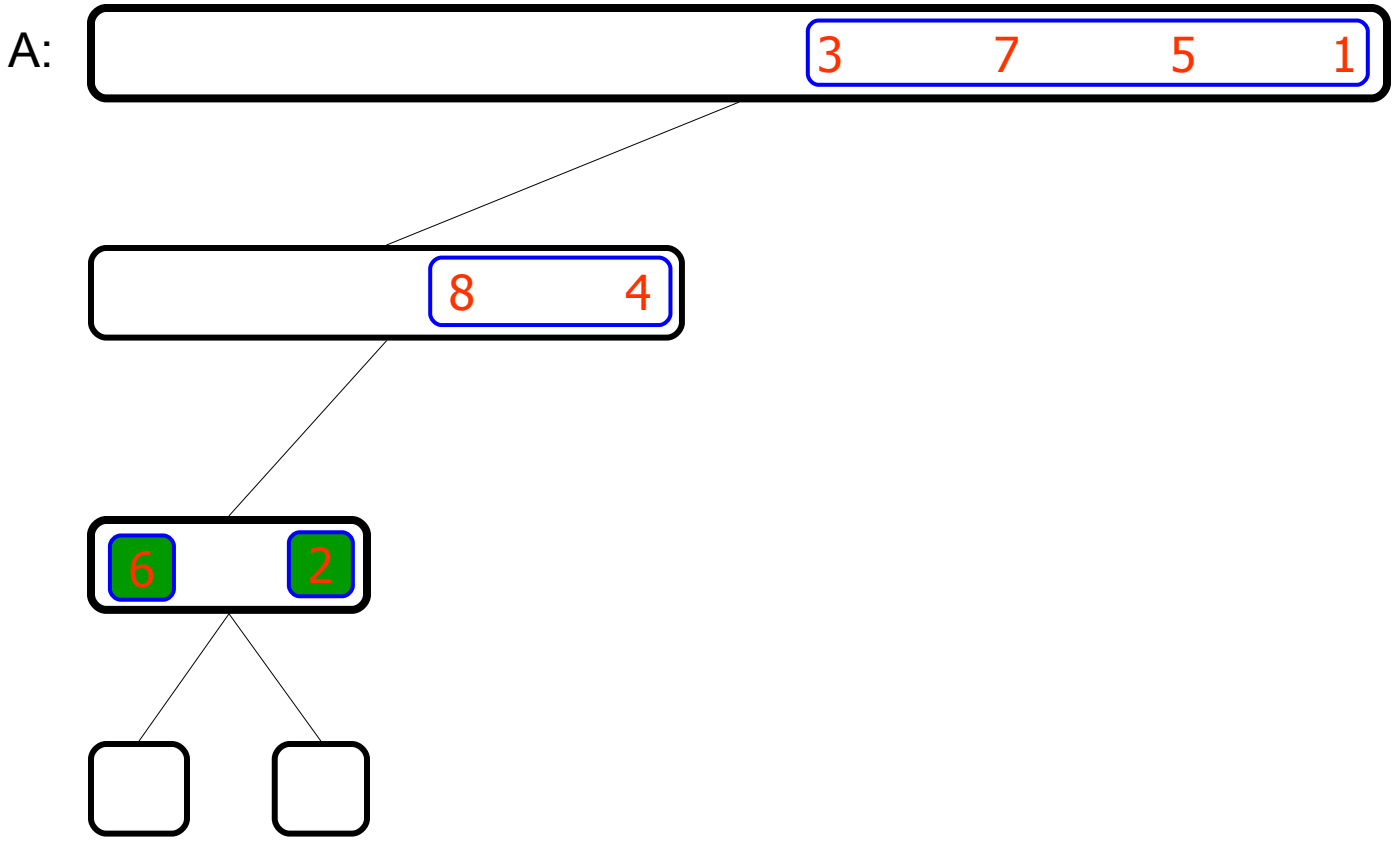
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 1, 1) , caso base



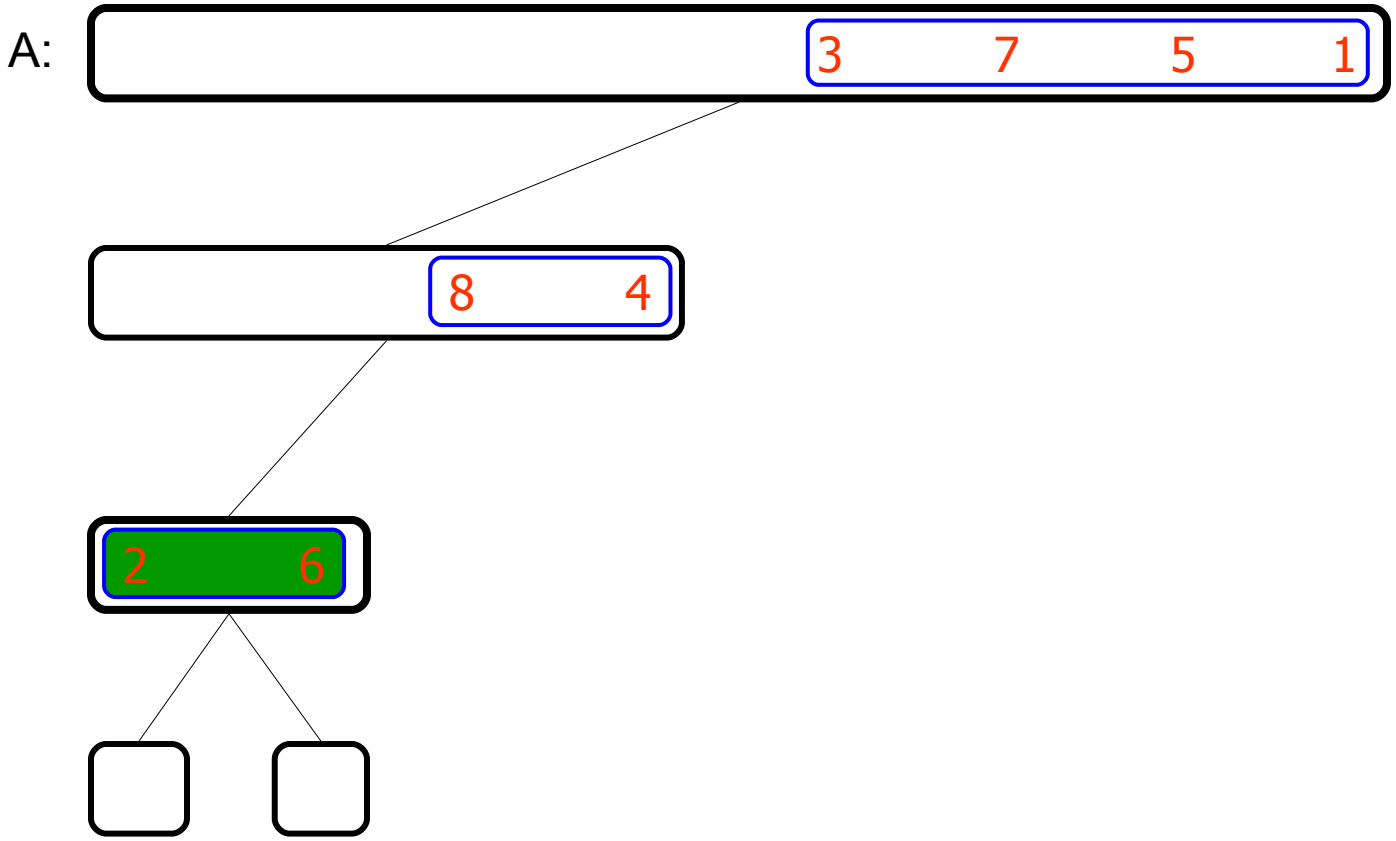
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 1, 1), retorna



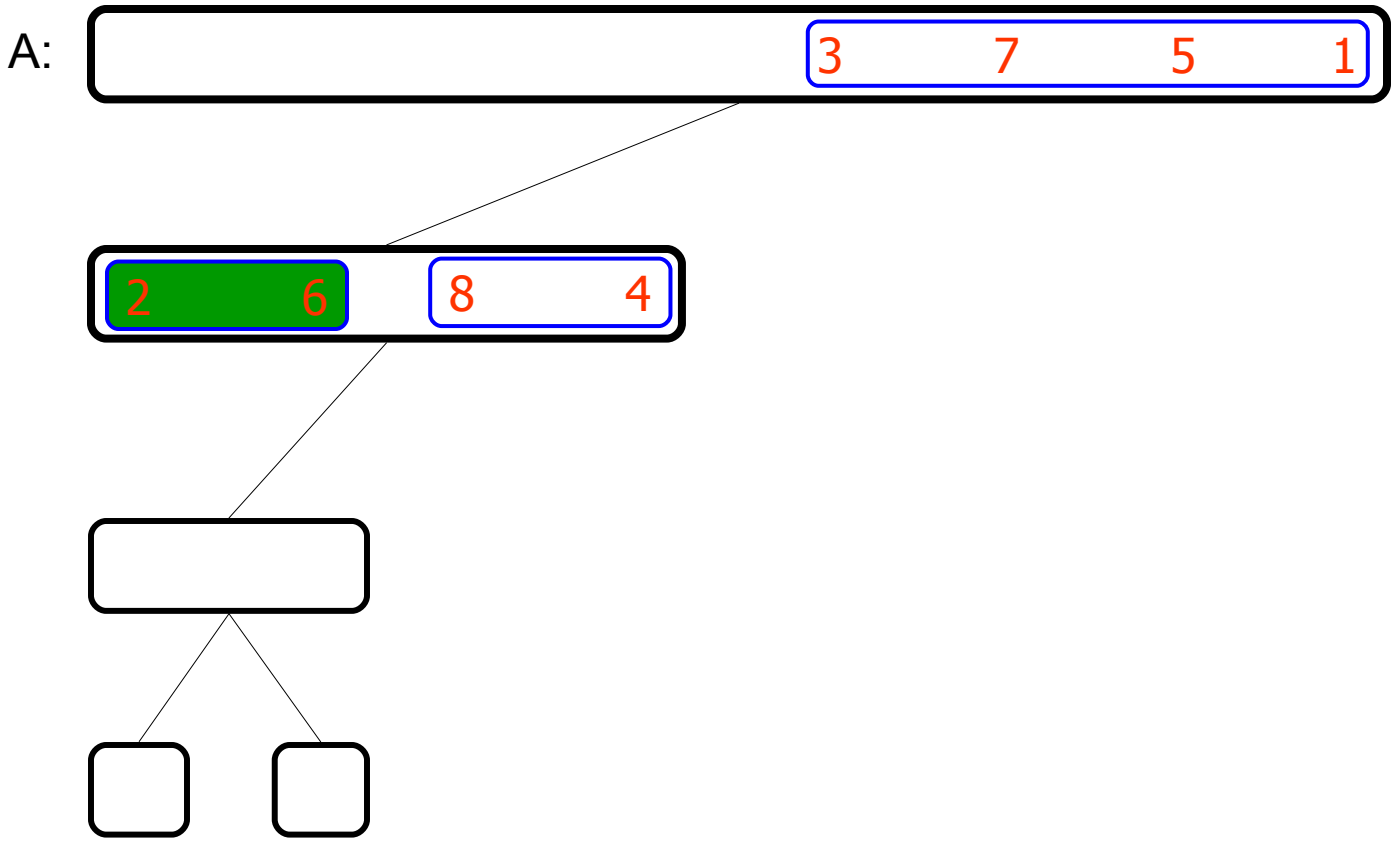
Merge-Sort(numbers, 0, 7)

Merge(numbers, 0, 0, 1)



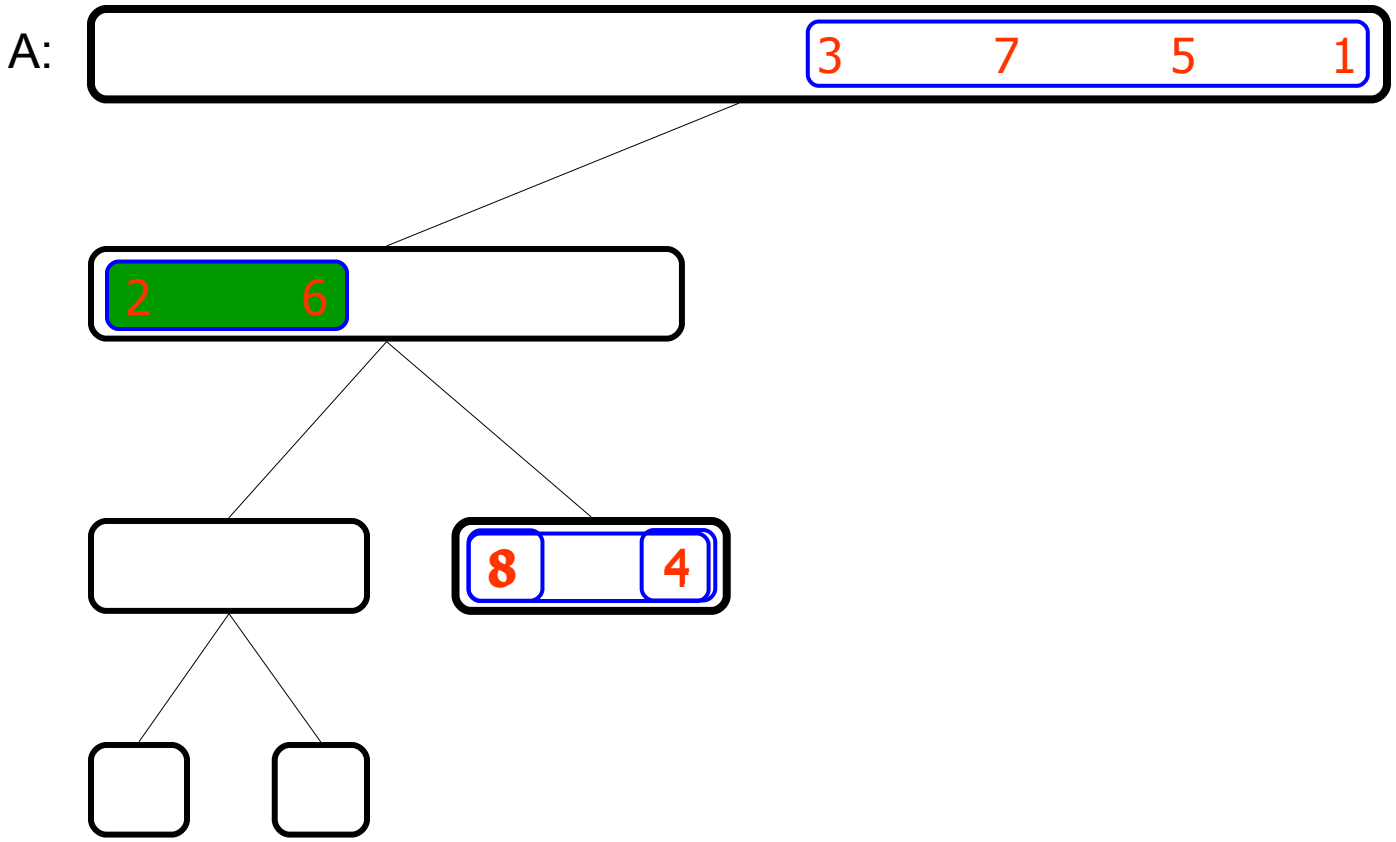
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 0, 1), retorna



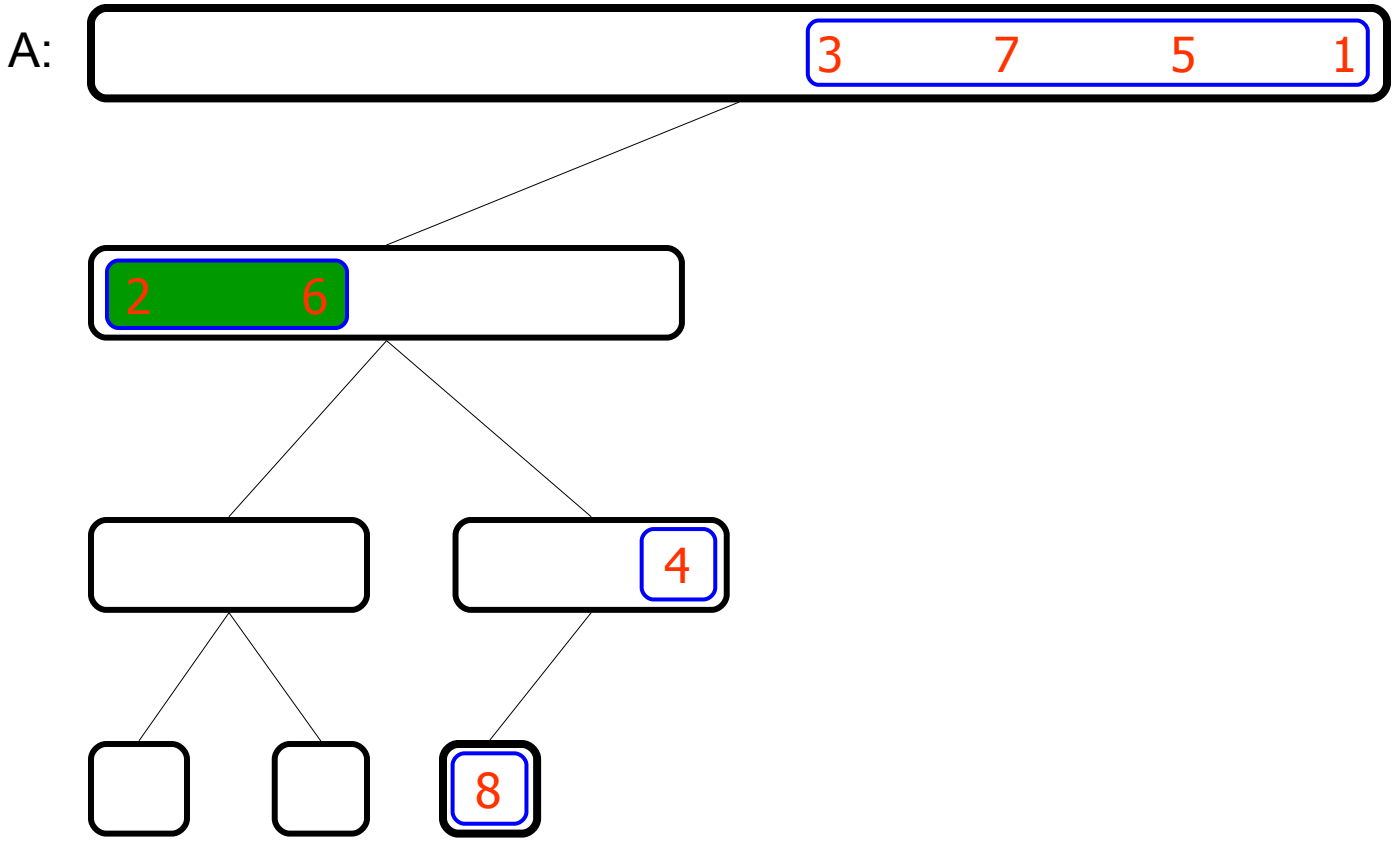
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 2, 3) , divide



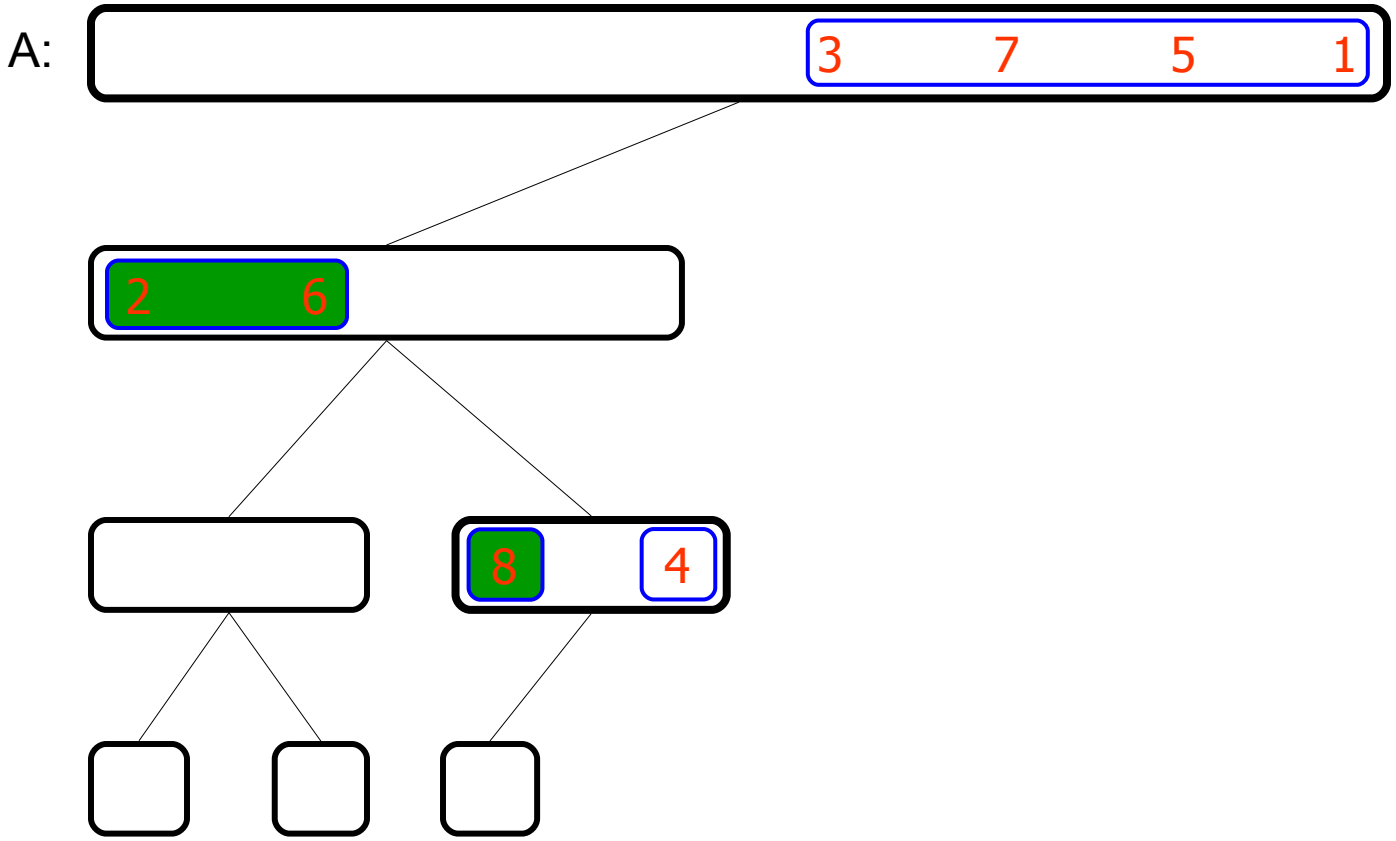
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 2, 2), caso base



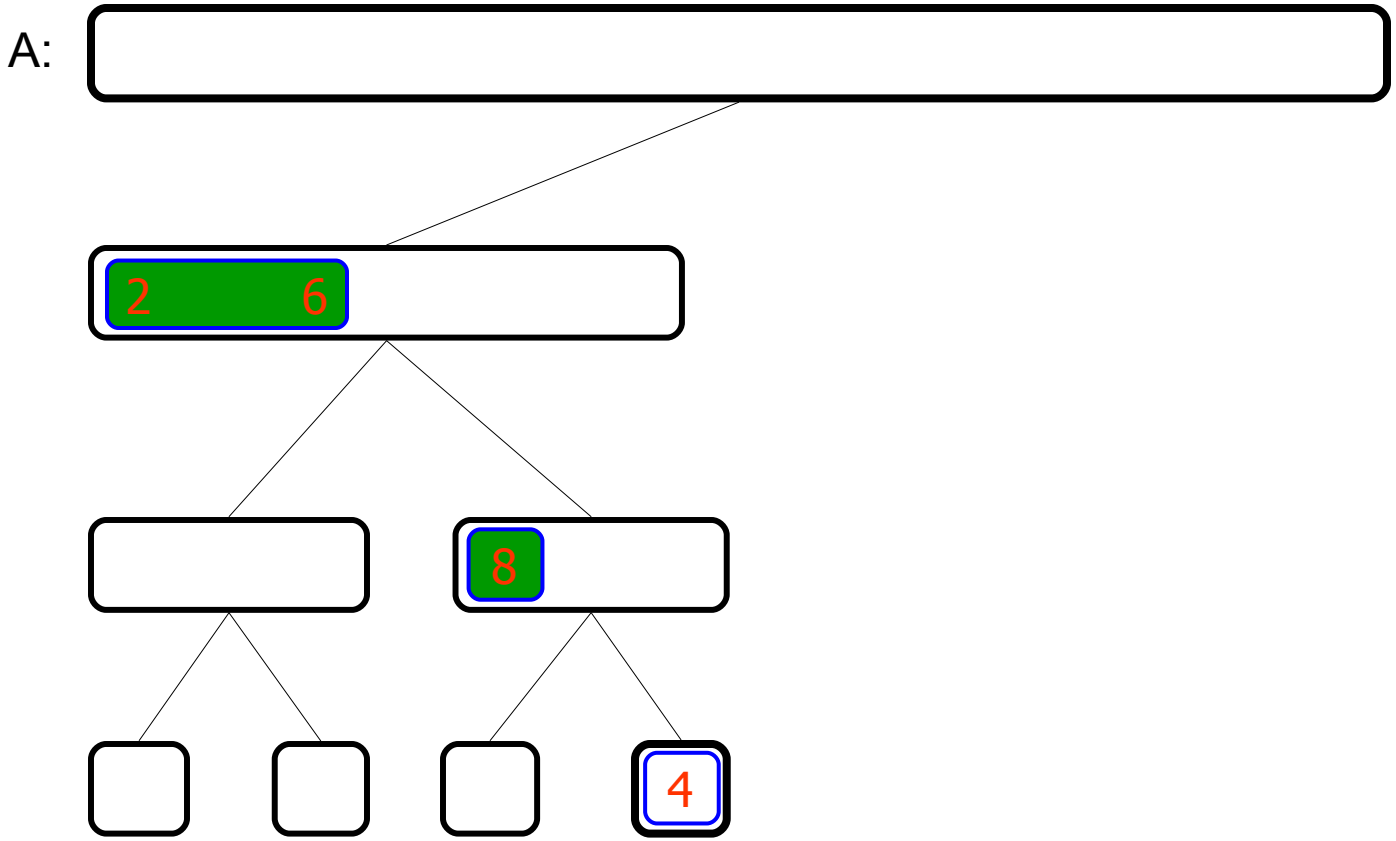
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 2, 2), retorna



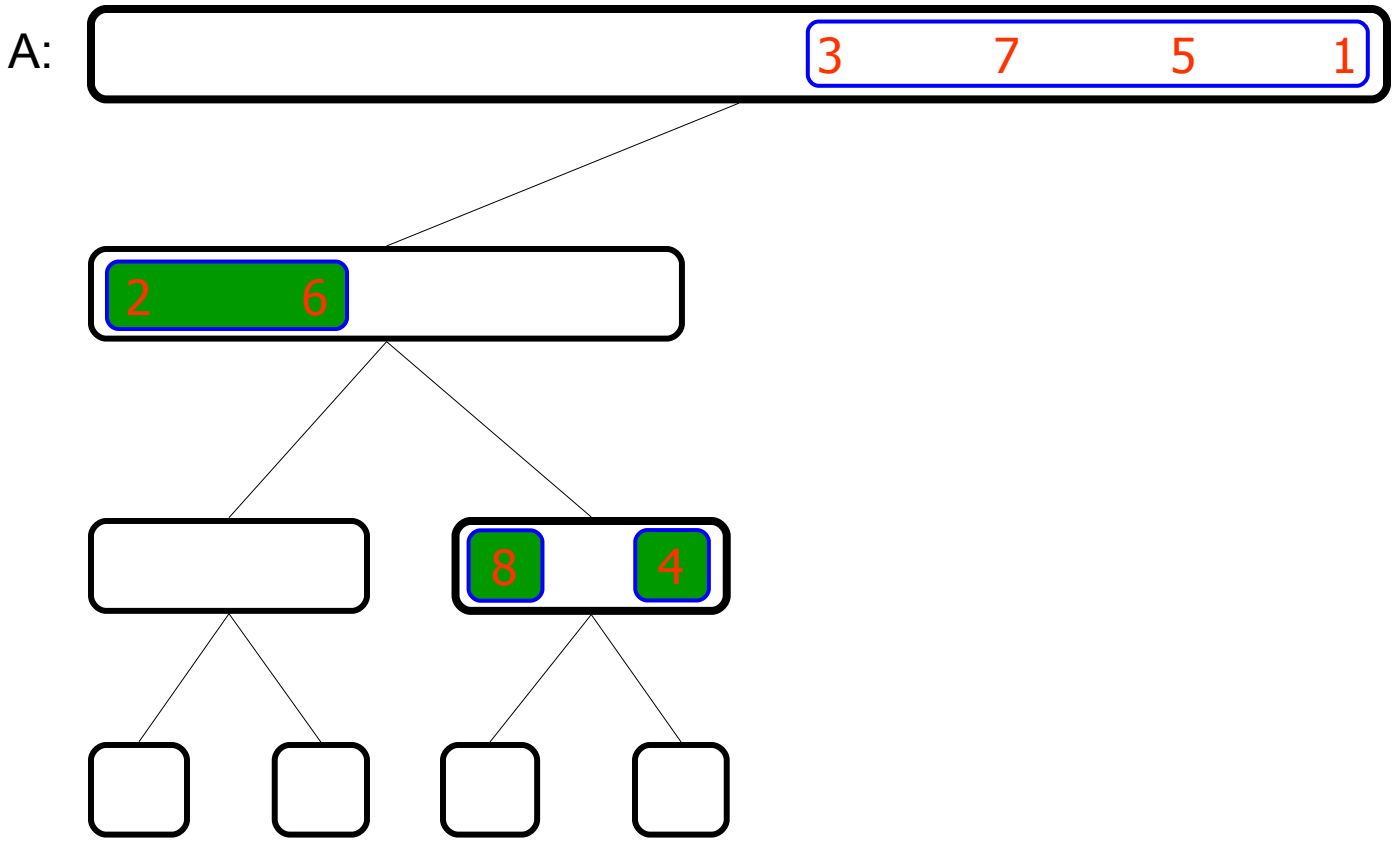
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 3, 3), caso base



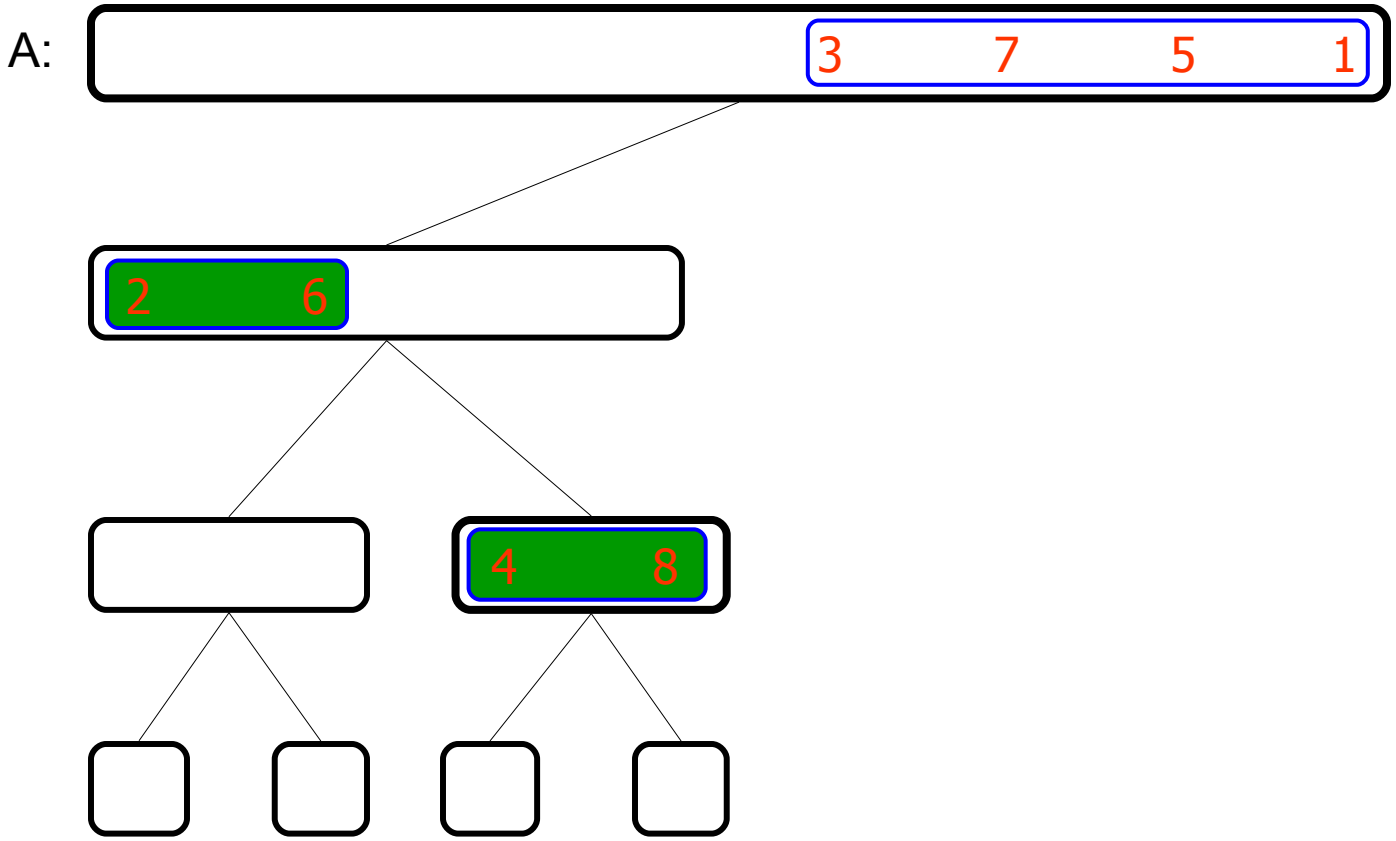
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 3, 3), retorna



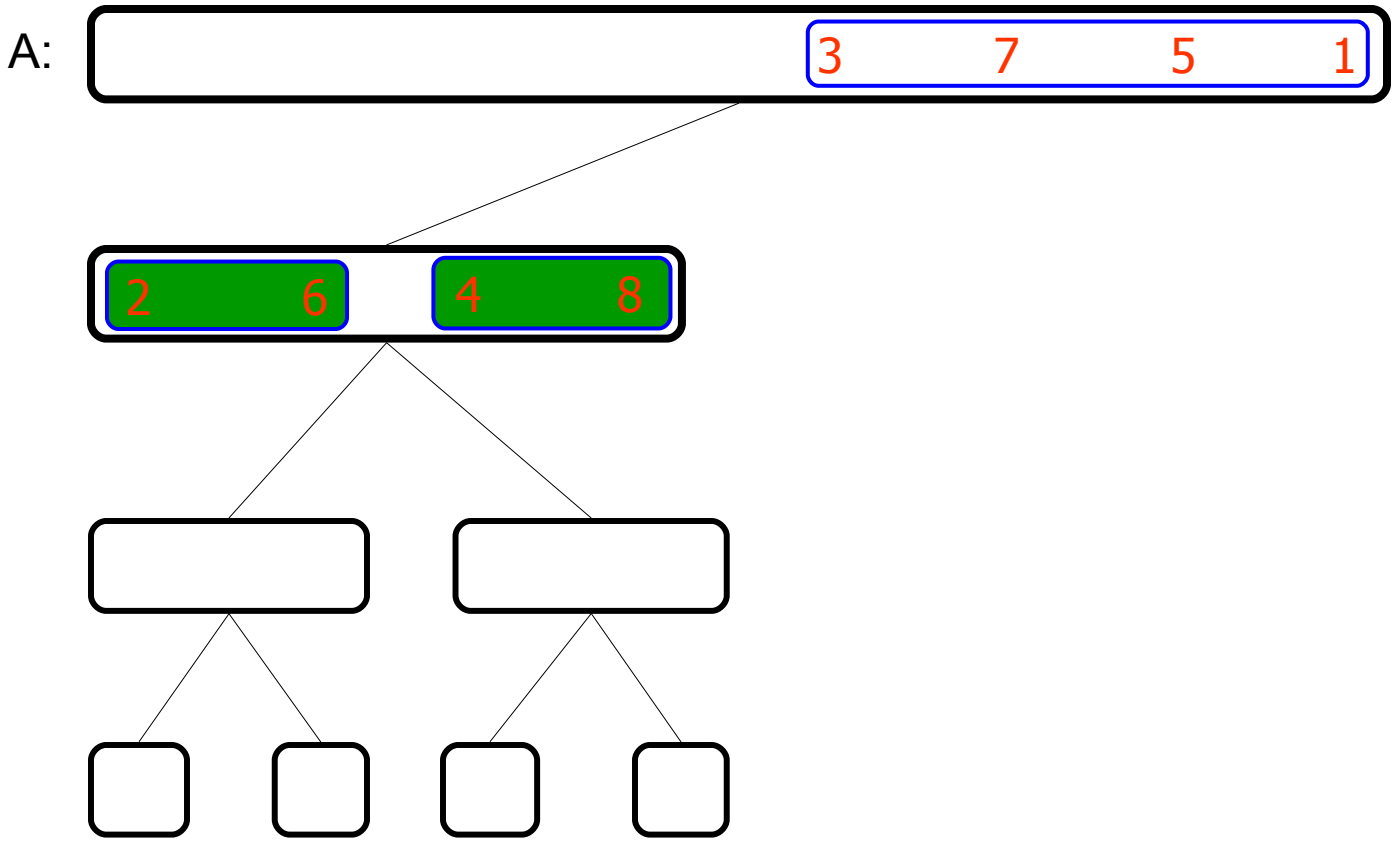
Merge-Sort(numbers, 0, 7)

Merge(numbers, 2, 2, 3)



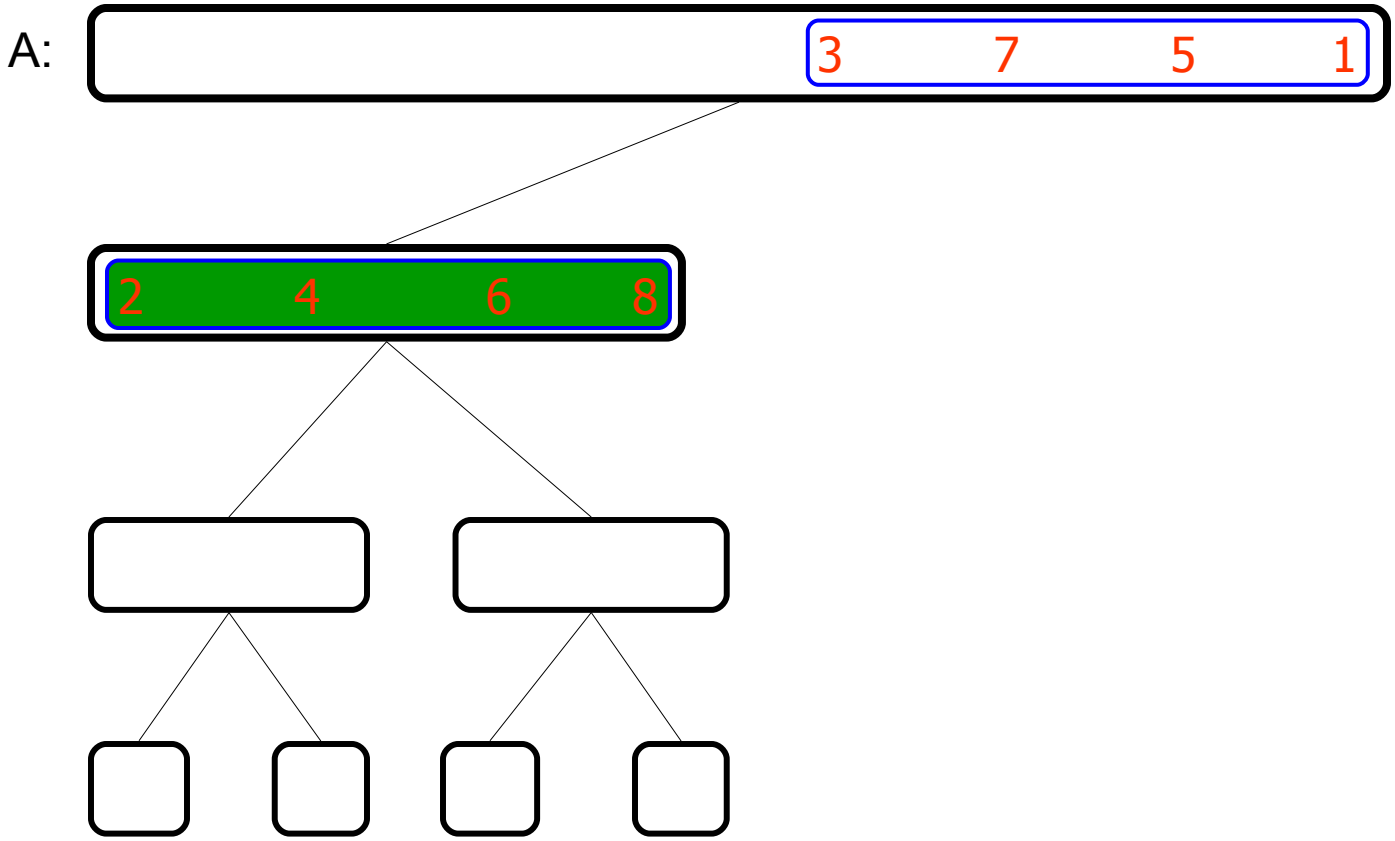
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 2, 3), retorna



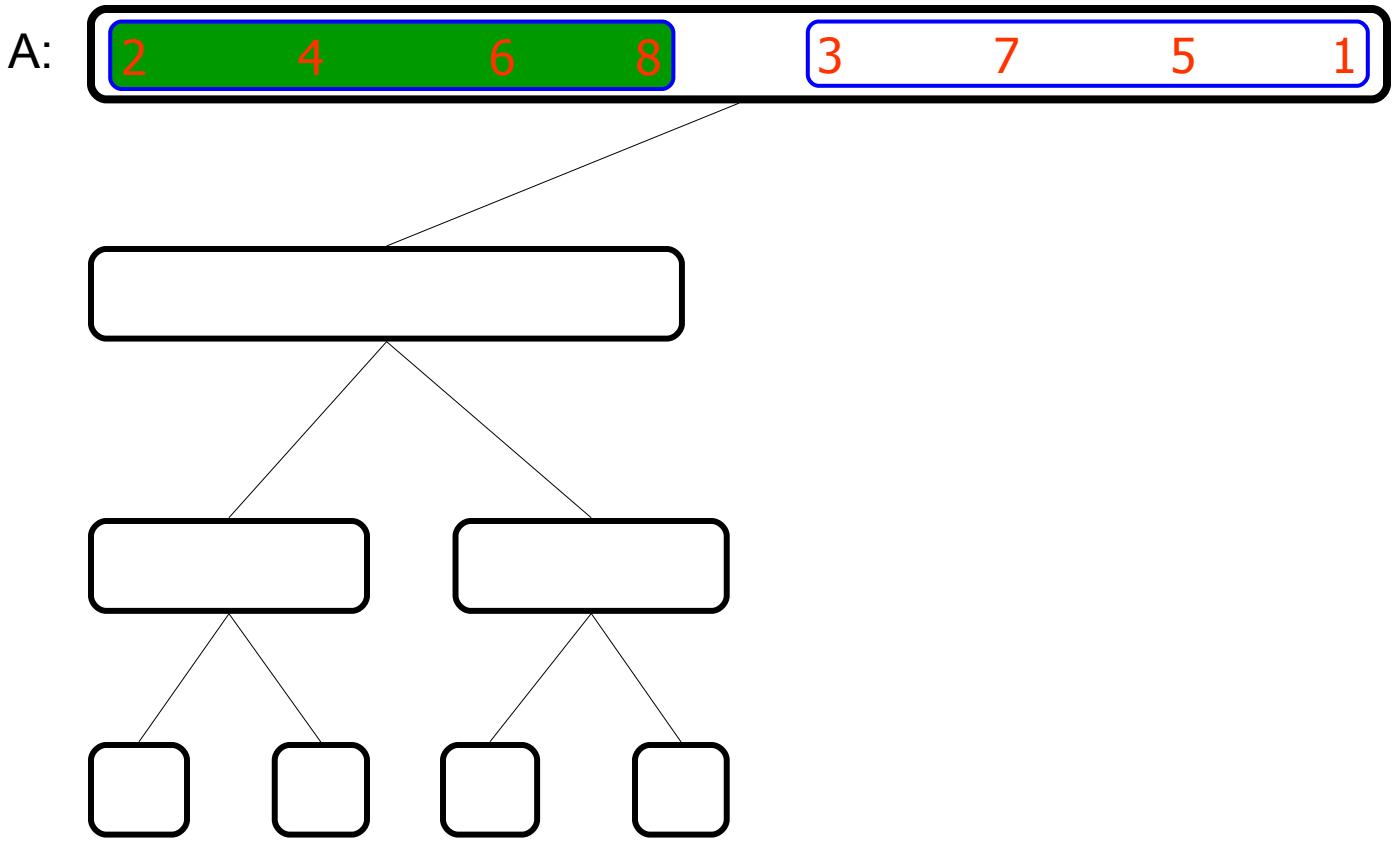
Merge-Sort(numbers, 0, 7)

Merge(numbers, 0, 1, 3)



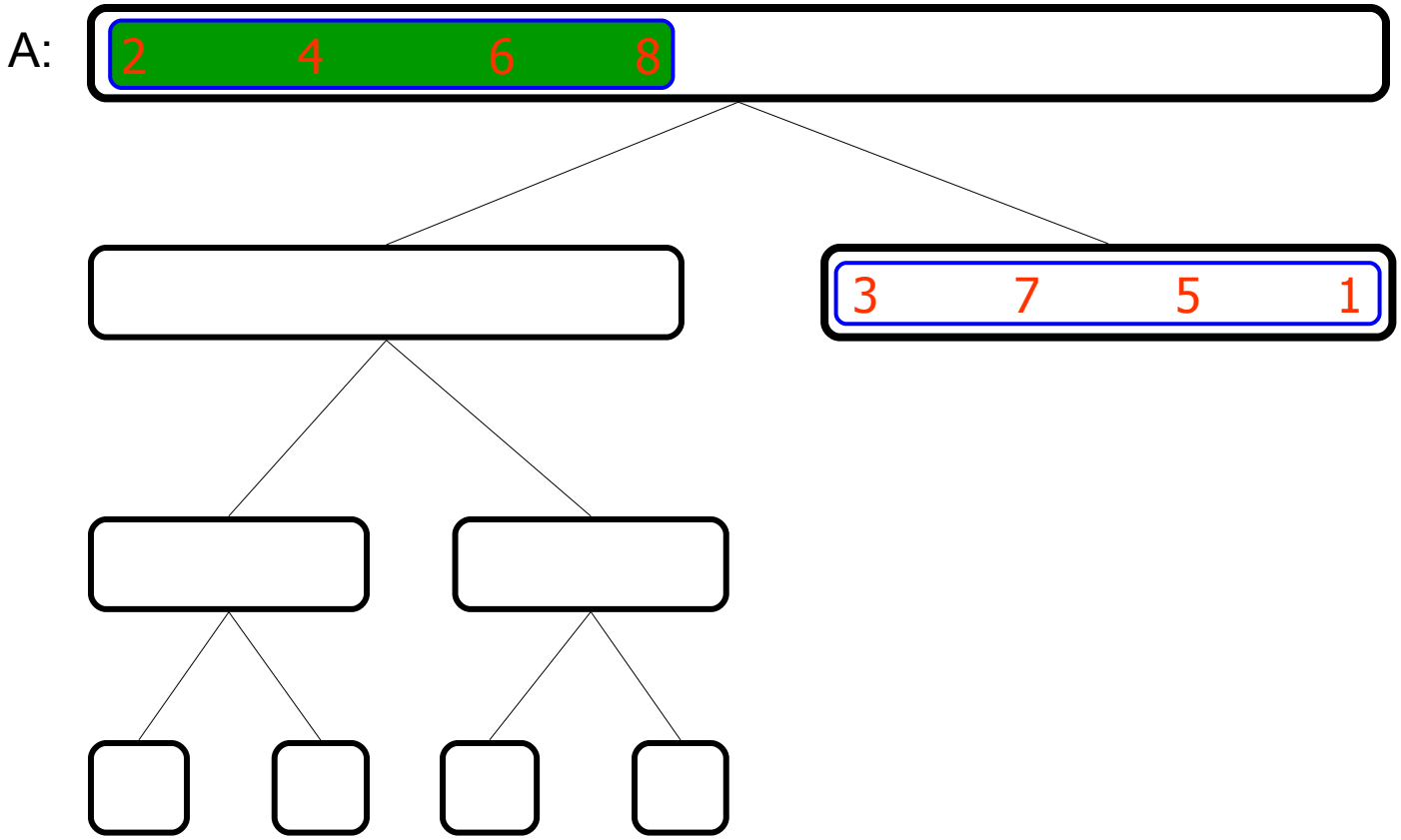
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 0, 3), retorna



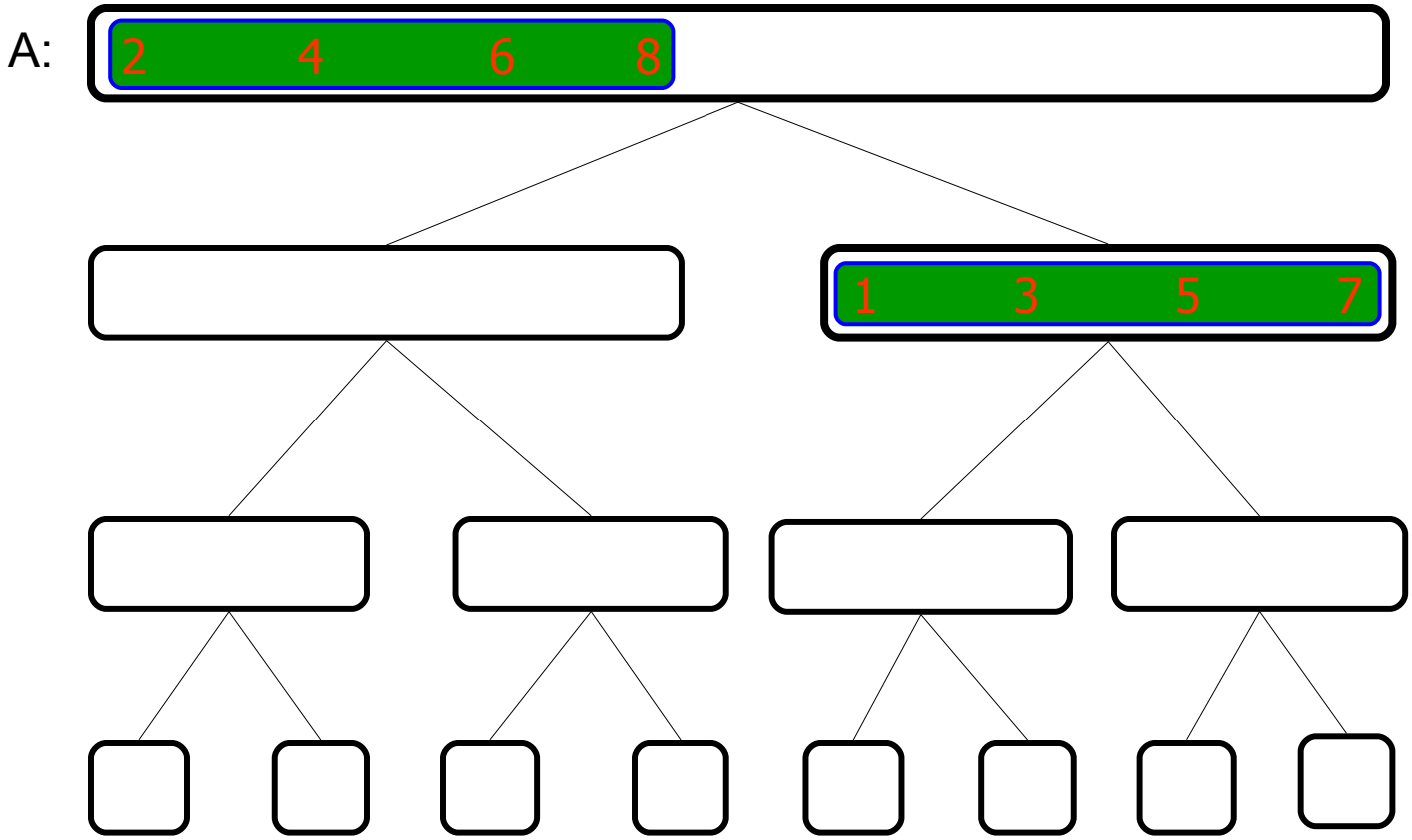
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 4, 7)



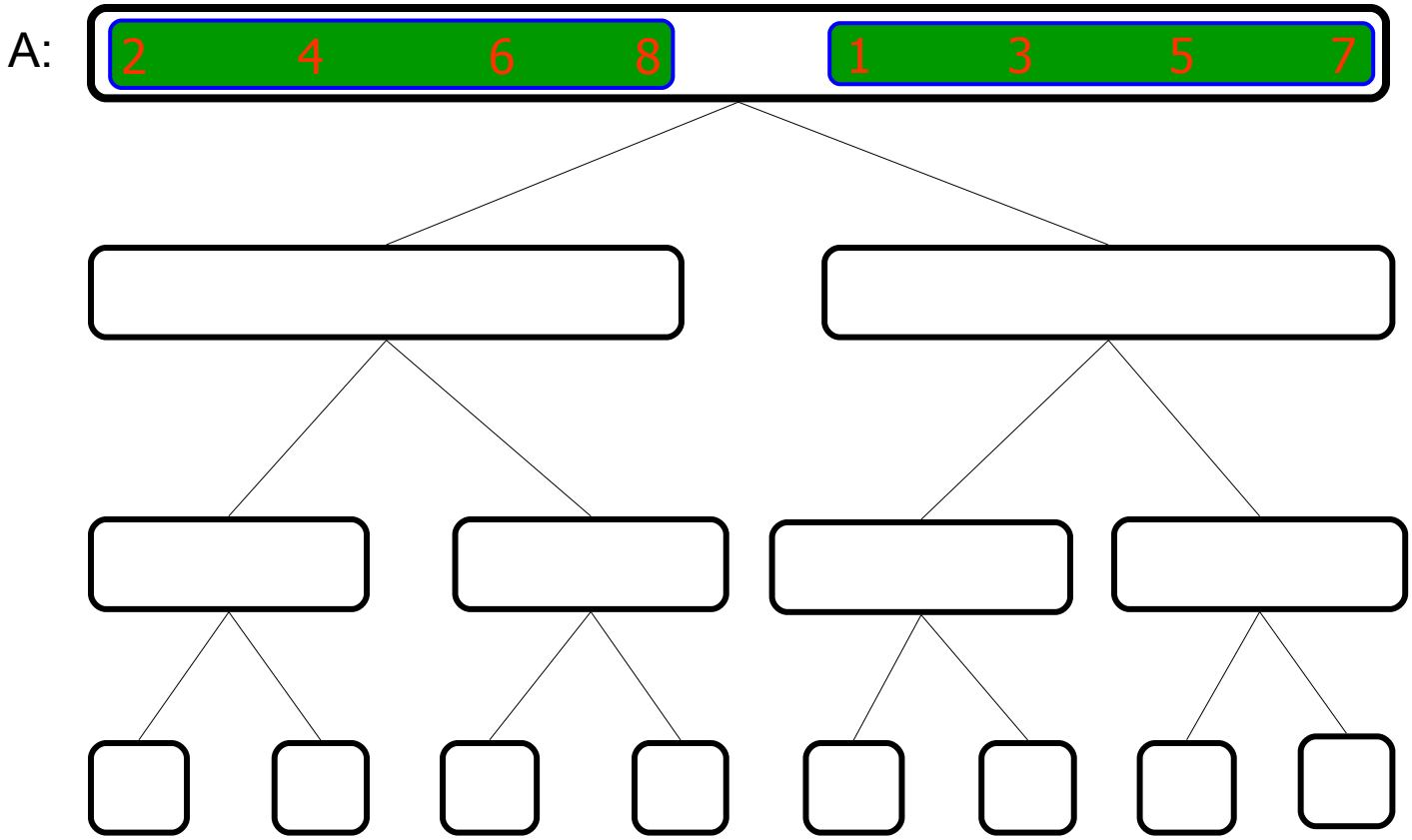
Merge-Sort(numbers, 0, 7)

Merge (numbers, 4, 5, 7)



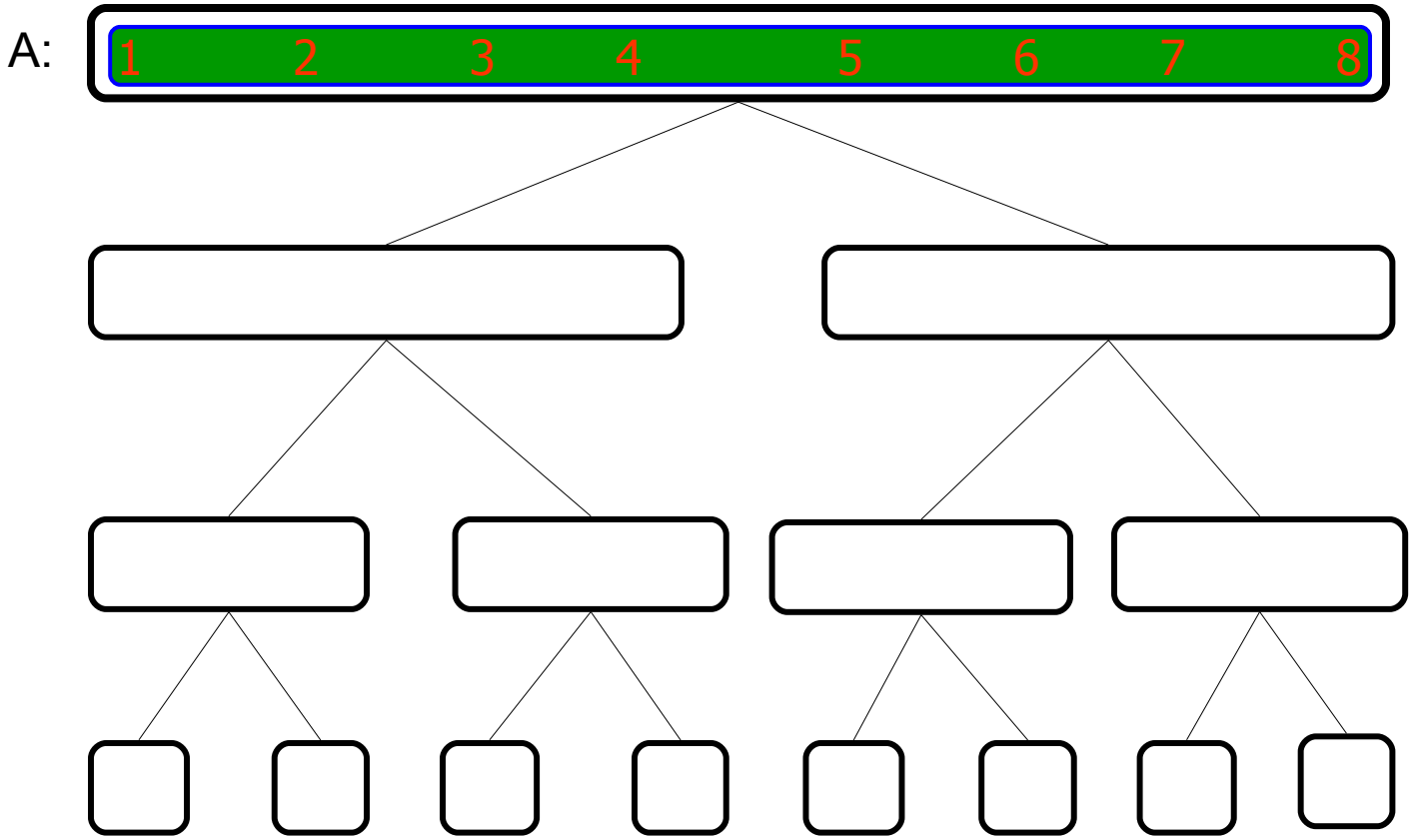
Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 4, 7), retorna



Merge-Sort(numbers, 0, 7)

Merge-Sort(numbers, 0, 7), Fim! Ordenado!



Merge Sort: Algoritmo

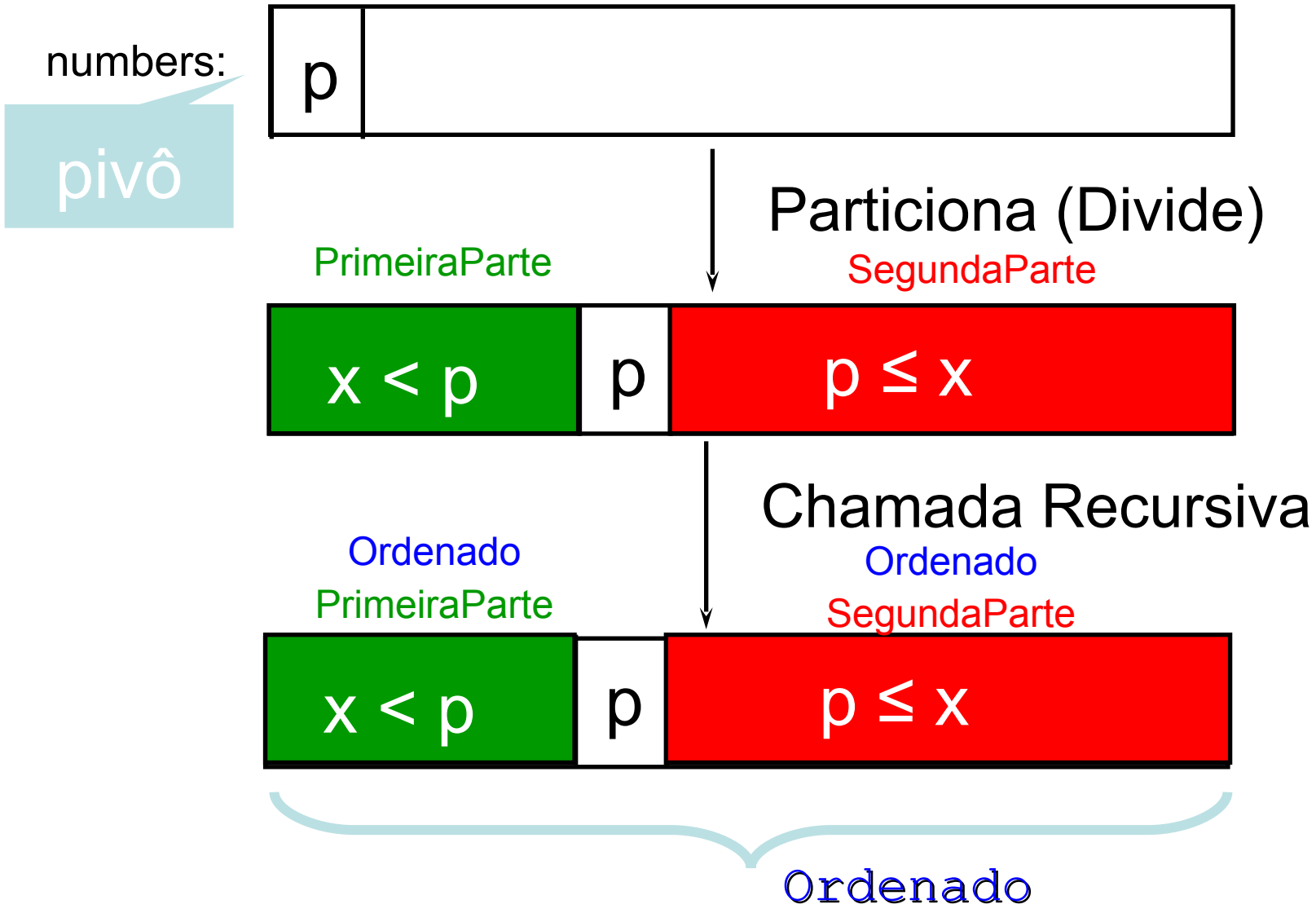
```
Merge-Sort (numbers, esq, dir)
  if    esq ≥ dir   (Não faz Nada!)
  else
        meio ←⇐esq+dir)/2
        Merge-Sort(numbers, esq, meio)
        Merge-Sort(numbers, meio+1, dir)
        Merge(numbers, esq, meio, dir)
```

Chamada Recursiva

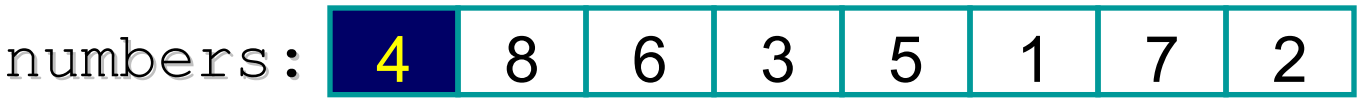
Quick Sort

Método da troca e partição

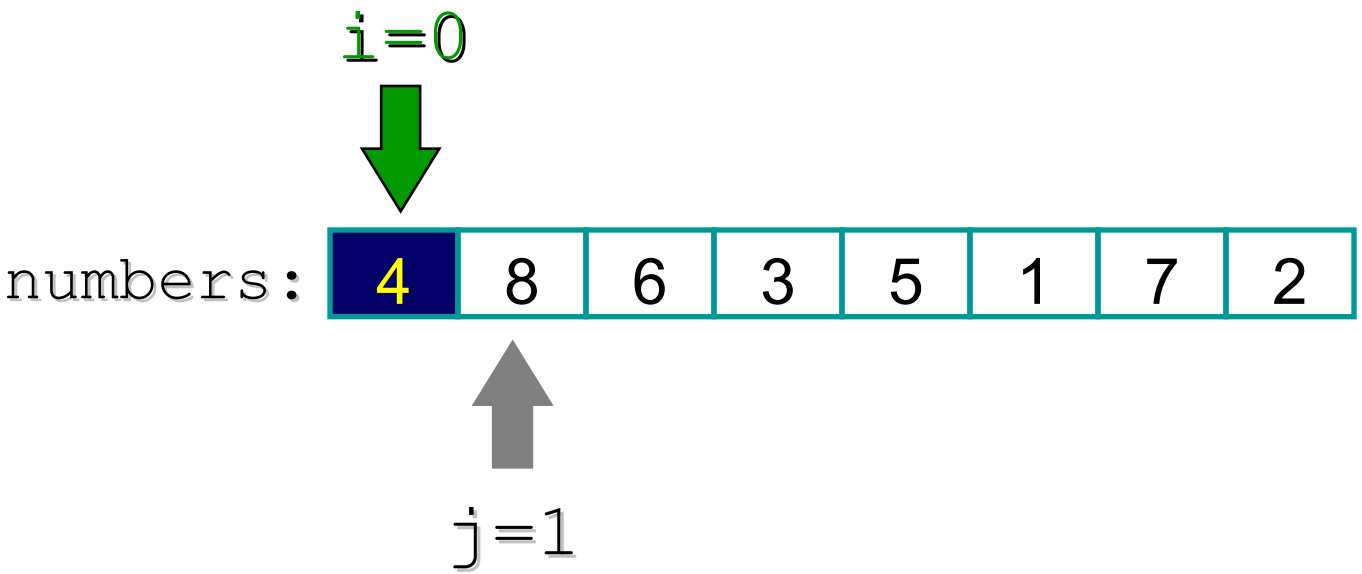
Quick Sort


numbers:
pivô
p
Particiona (Divide)
PrimeiraParte
SegundaParte
x < p
p
p ≤ x
Chamada Recursiva
Ordenado
PrimeiraParte
Ordenado
SegundaParte
x < p
p
p ≤ x
Ordenado

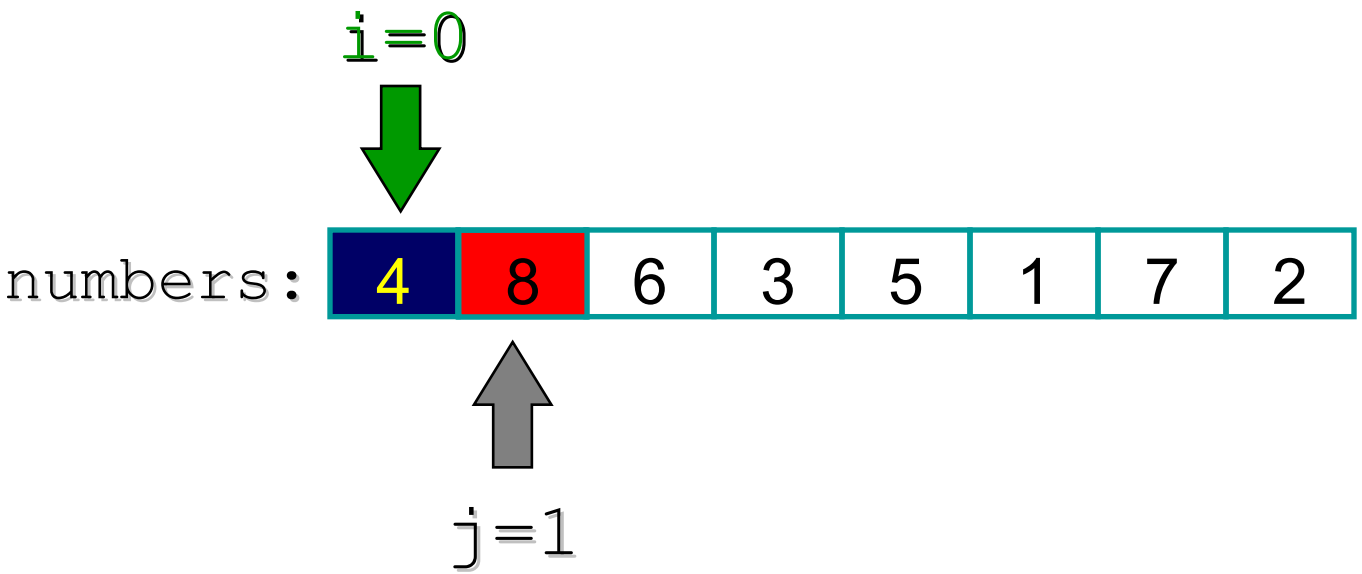
Exemplo de como Particionar



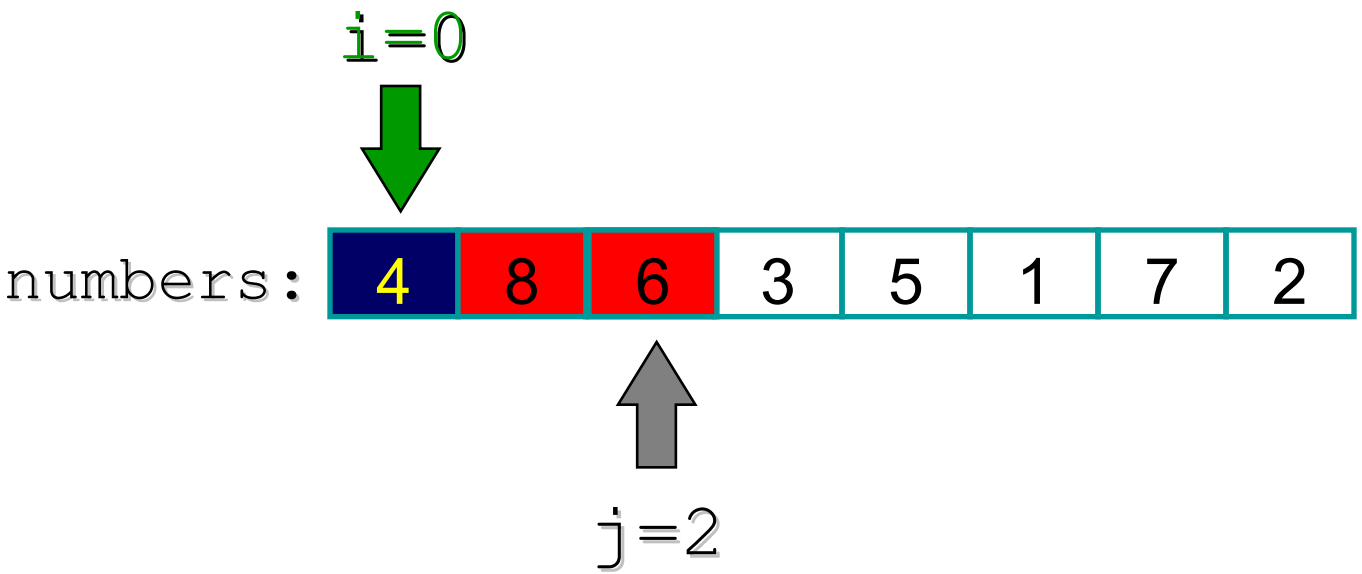
Exemplo de como Particionar



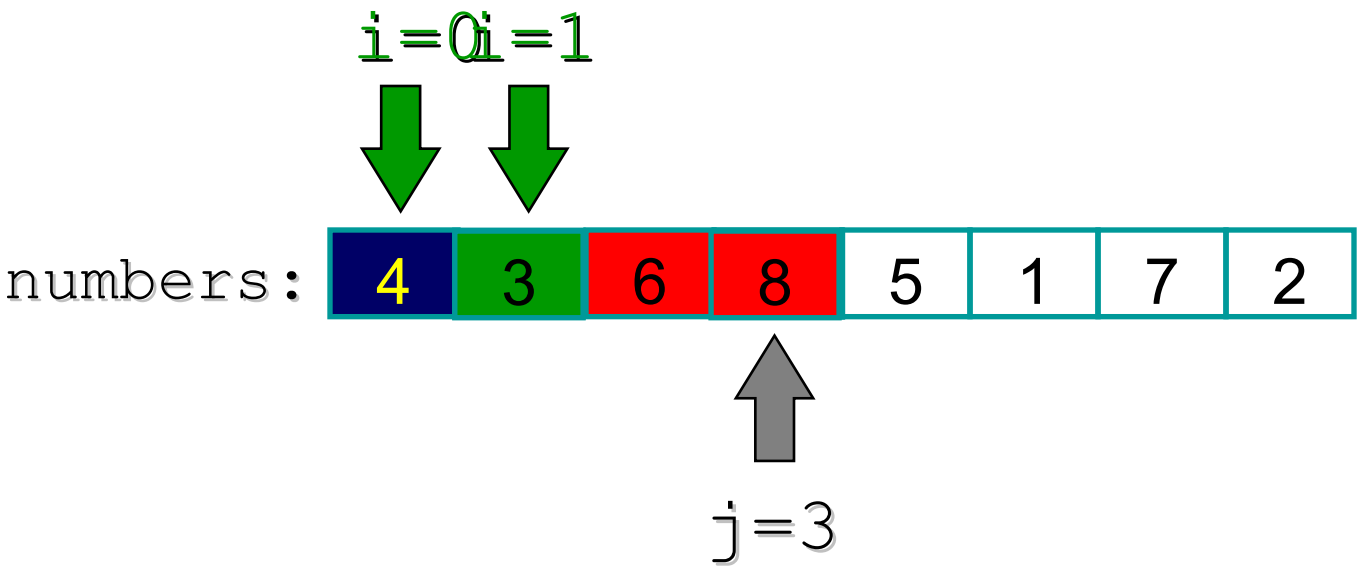
Exemplo de como Particionar



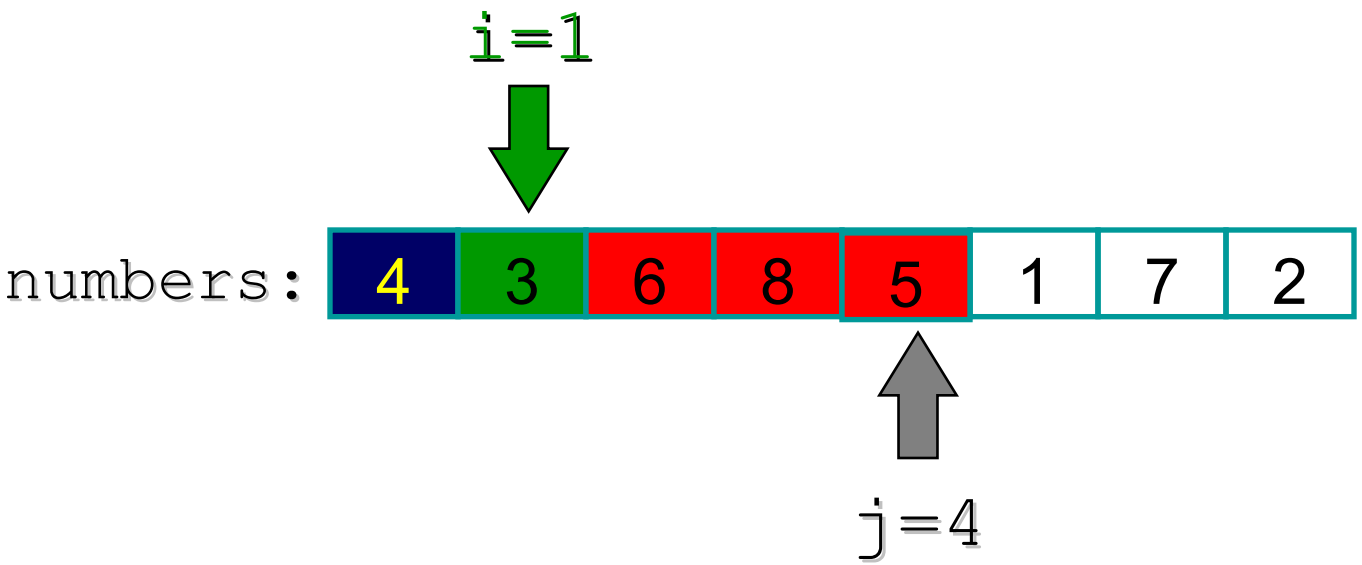
Exemplo de como Particionar



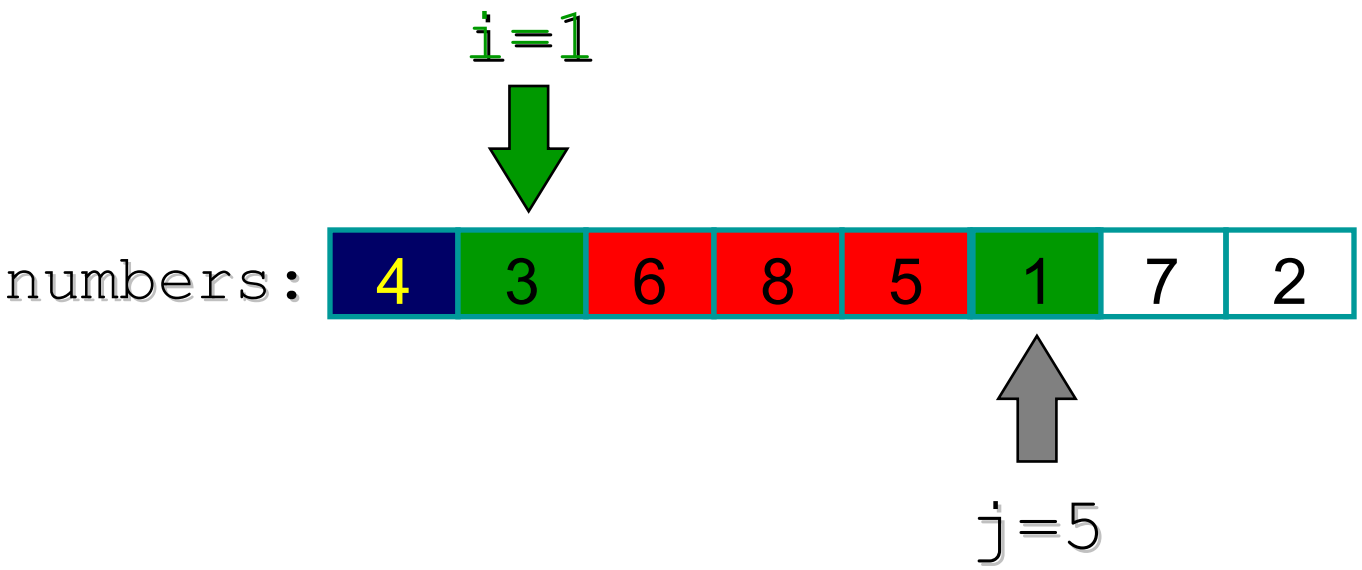
Exemplo de como Particionar



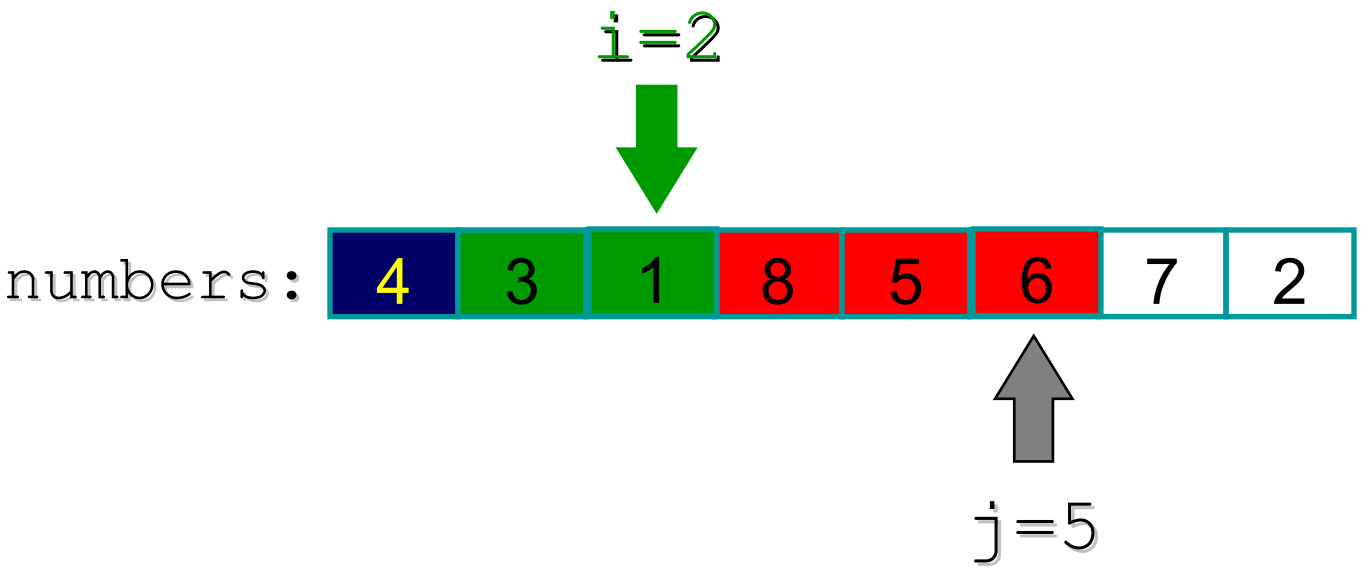
Exemplo de como Particionar



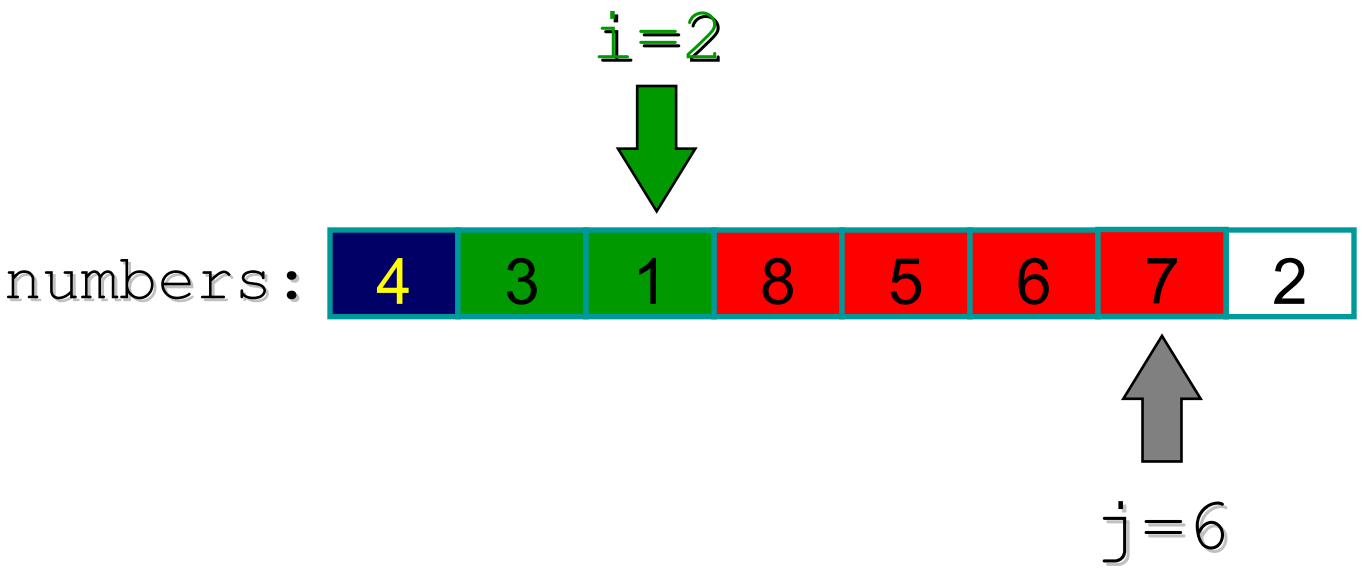
Exemplo de como Particionar



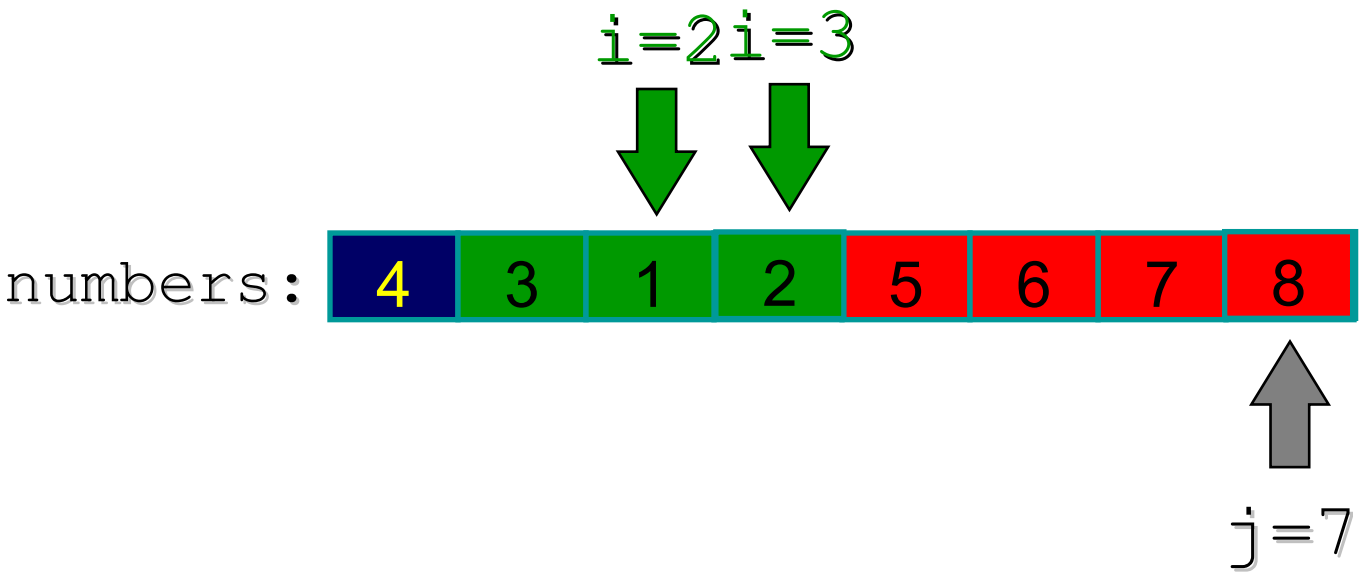
Exemplo de como Particionar



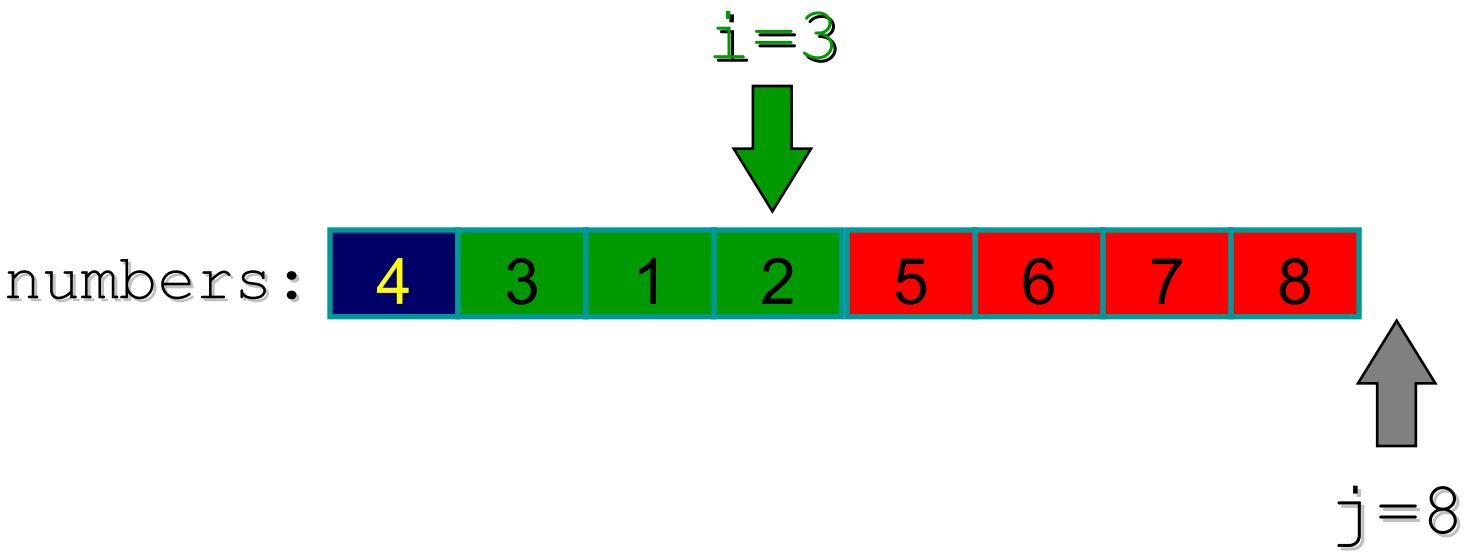
Exemplo de como Particionar



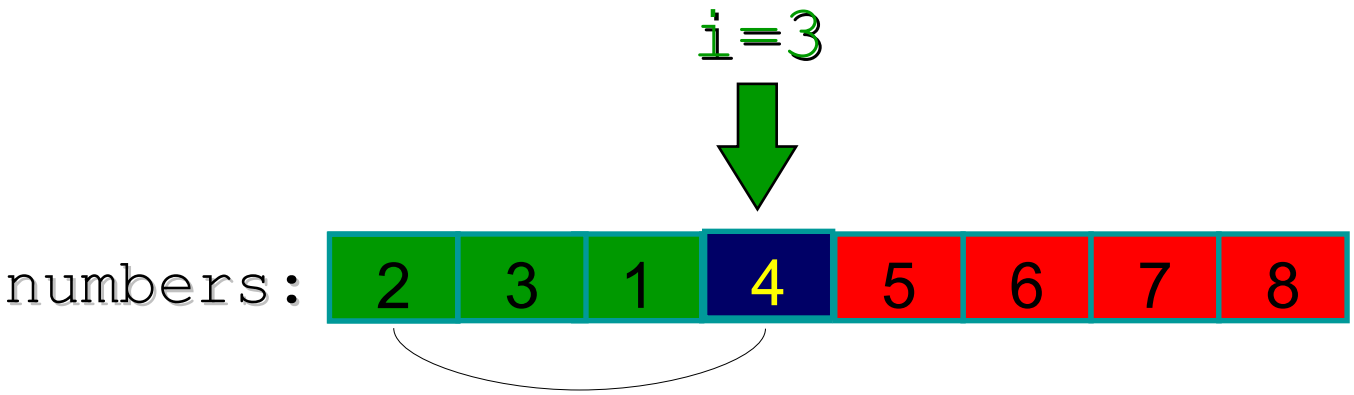
Exemplo de como Particionar



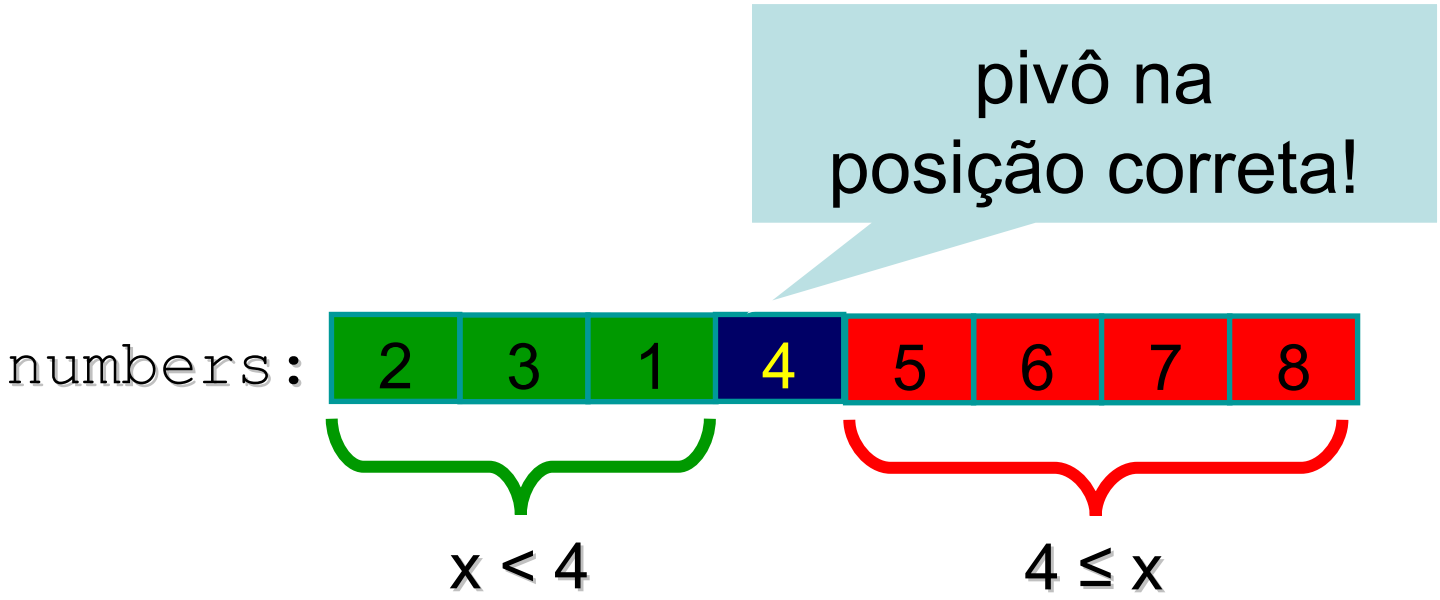
Exemplo de como Particionar



Exemplo de como Particionar



Exemplo de como Particionar

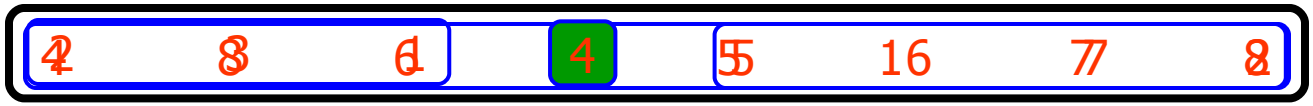


```
particiona(numbers, esq, dir)
1.    x ← numbers[esq]
2.    i ← esq
3.    for j ← esq+1 até dir
4.      if numbers[j] < x then
5.        i ← i + 1
6.        troque(numbers[i], numbers[j])
7.      end if
8.    end for j
9.    troque(numbers[i], numbers[esq])
10.   return i
```

Quick-Sort(numbers, 0, 7)

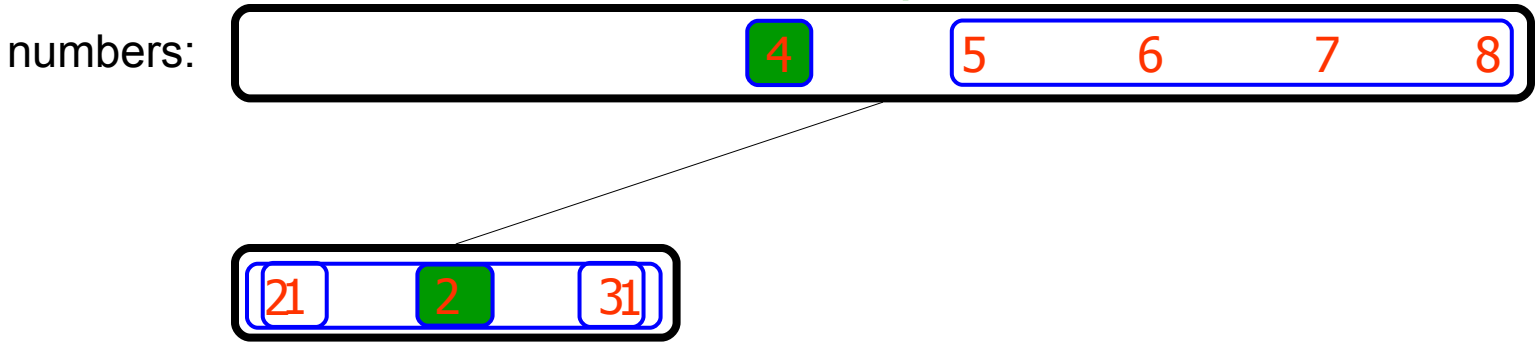
Particiona

numbers:



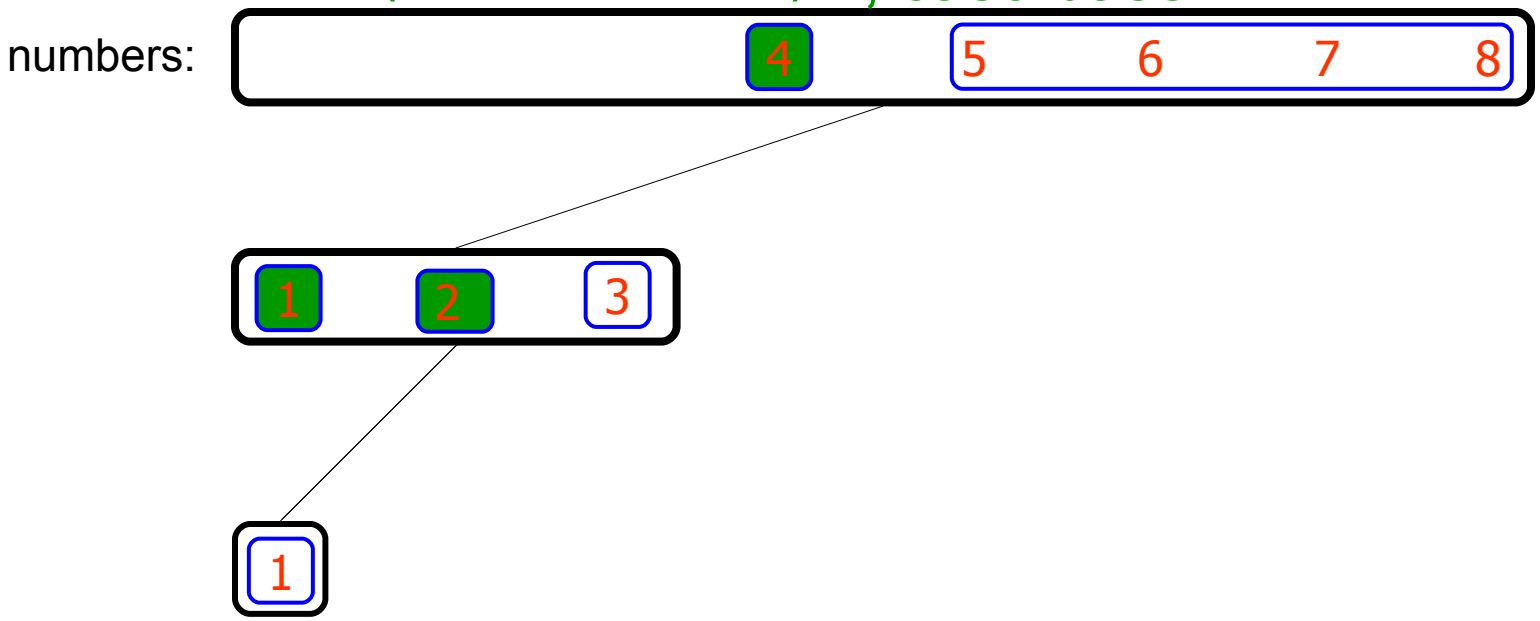
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 0, 2) , particiona



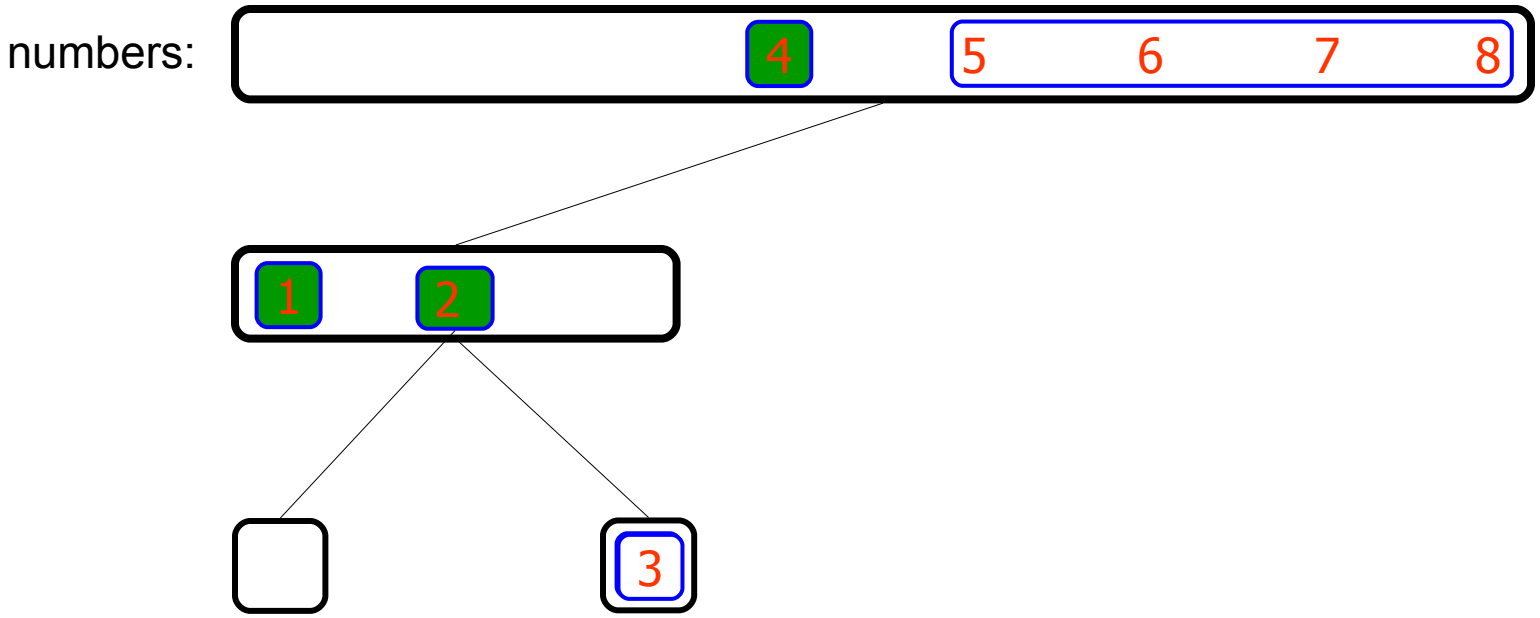
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 0, 0) , Quick-Sort(numbers, 2, 7)



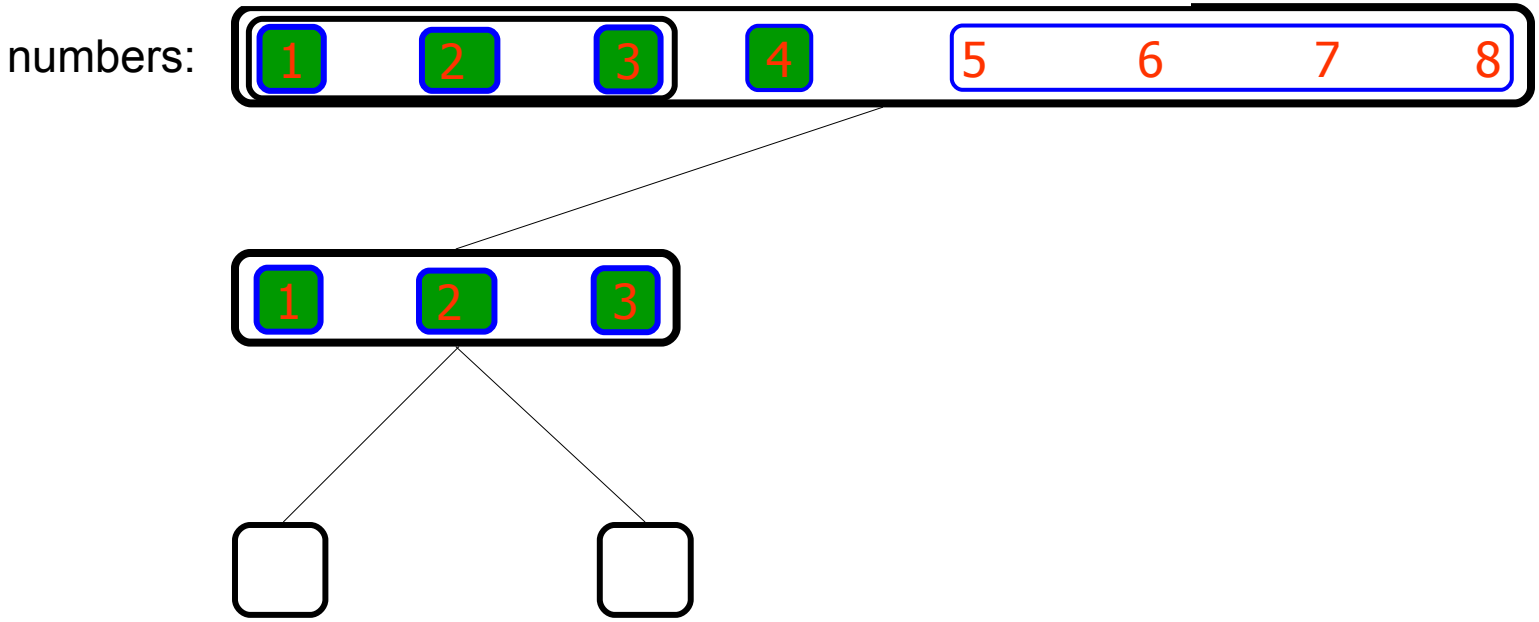
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 1, 1) , caso base



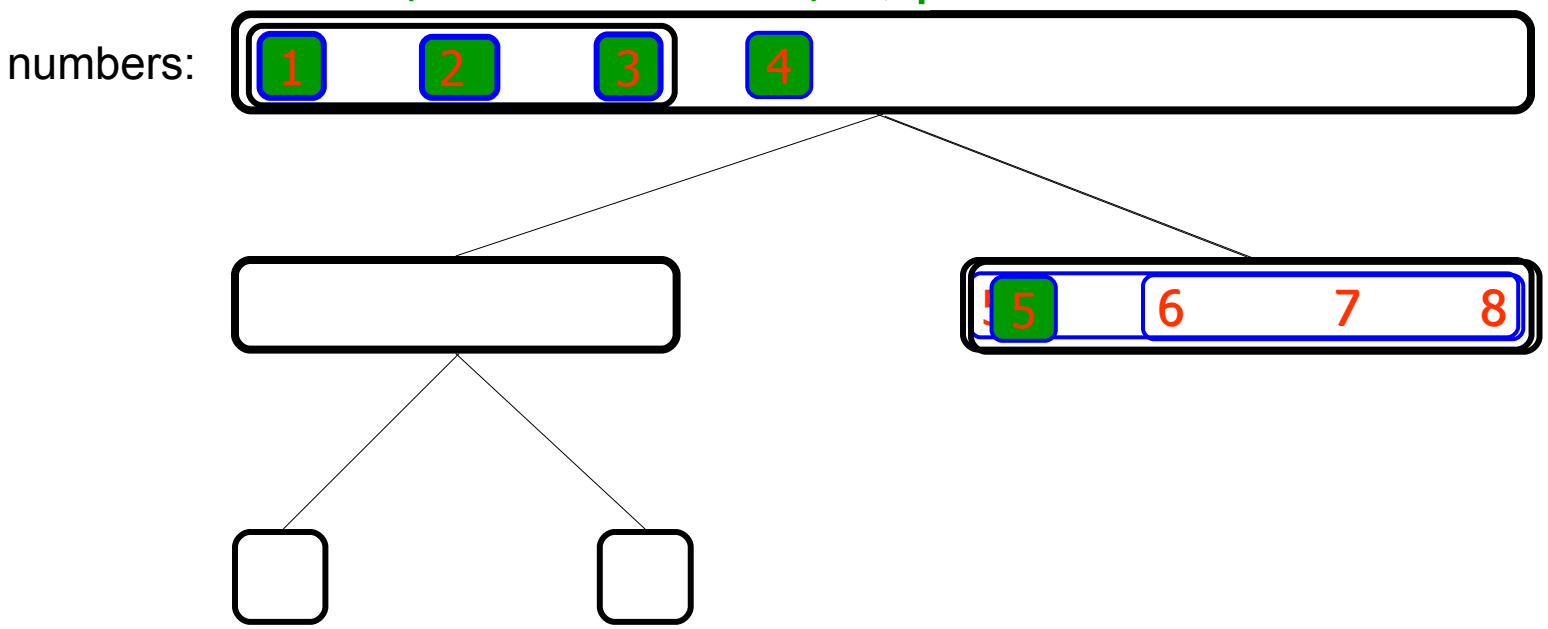
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 0, 2), retorna



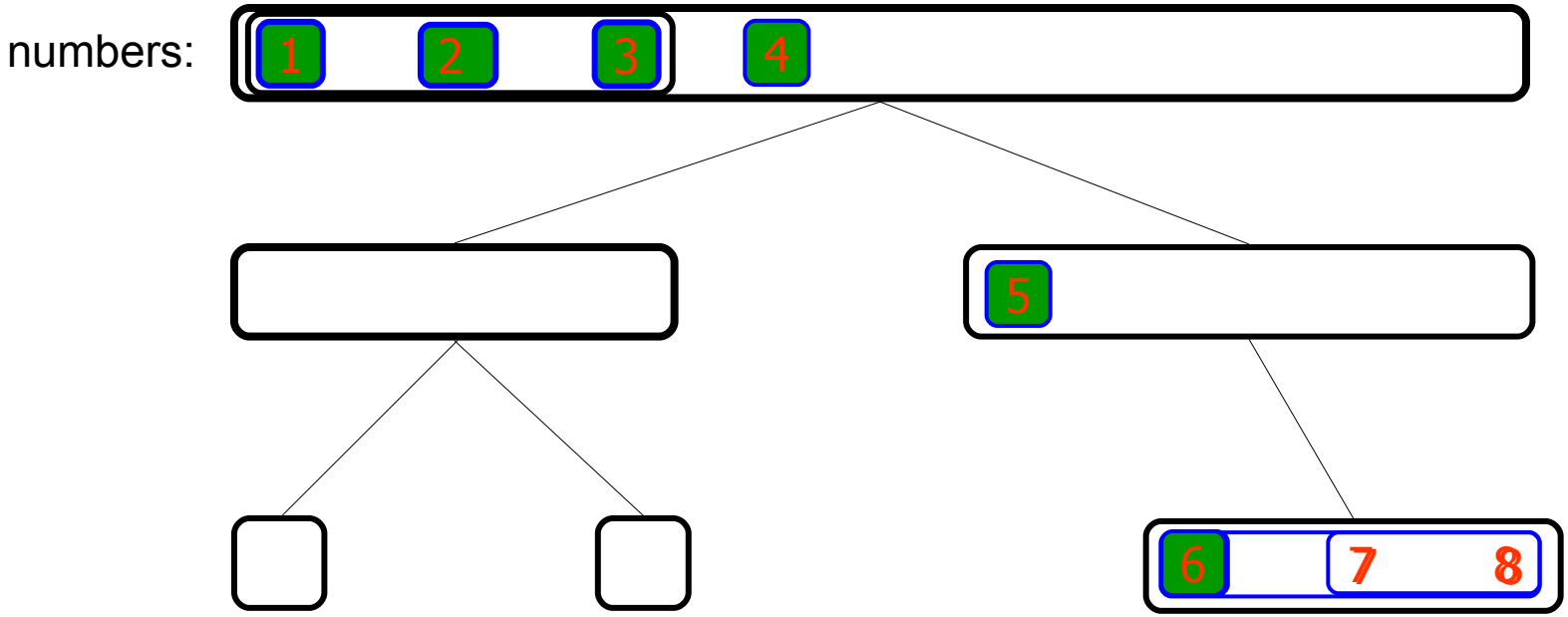
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 4, 7) , particiona



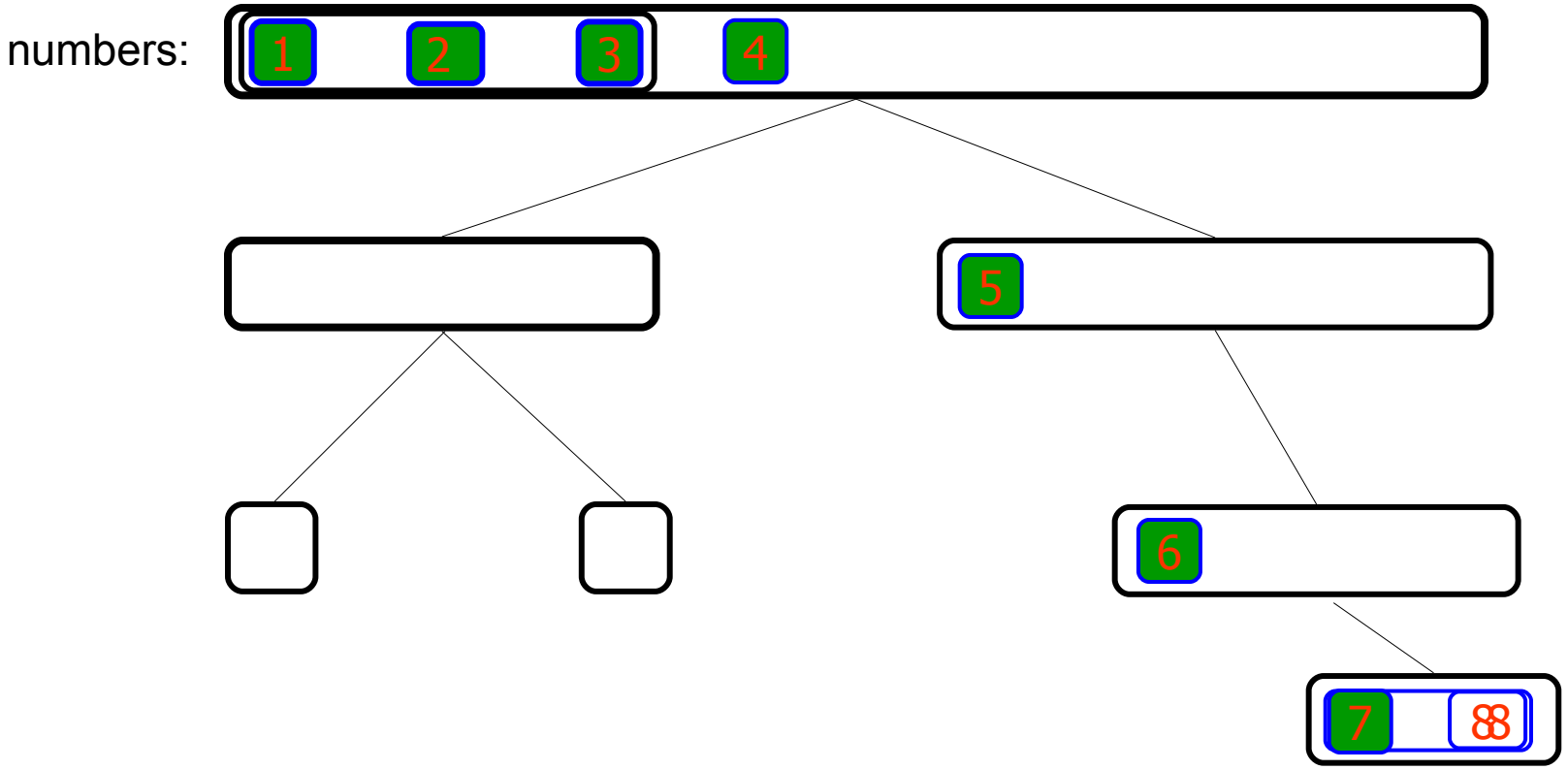
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 5, 7) , particiona



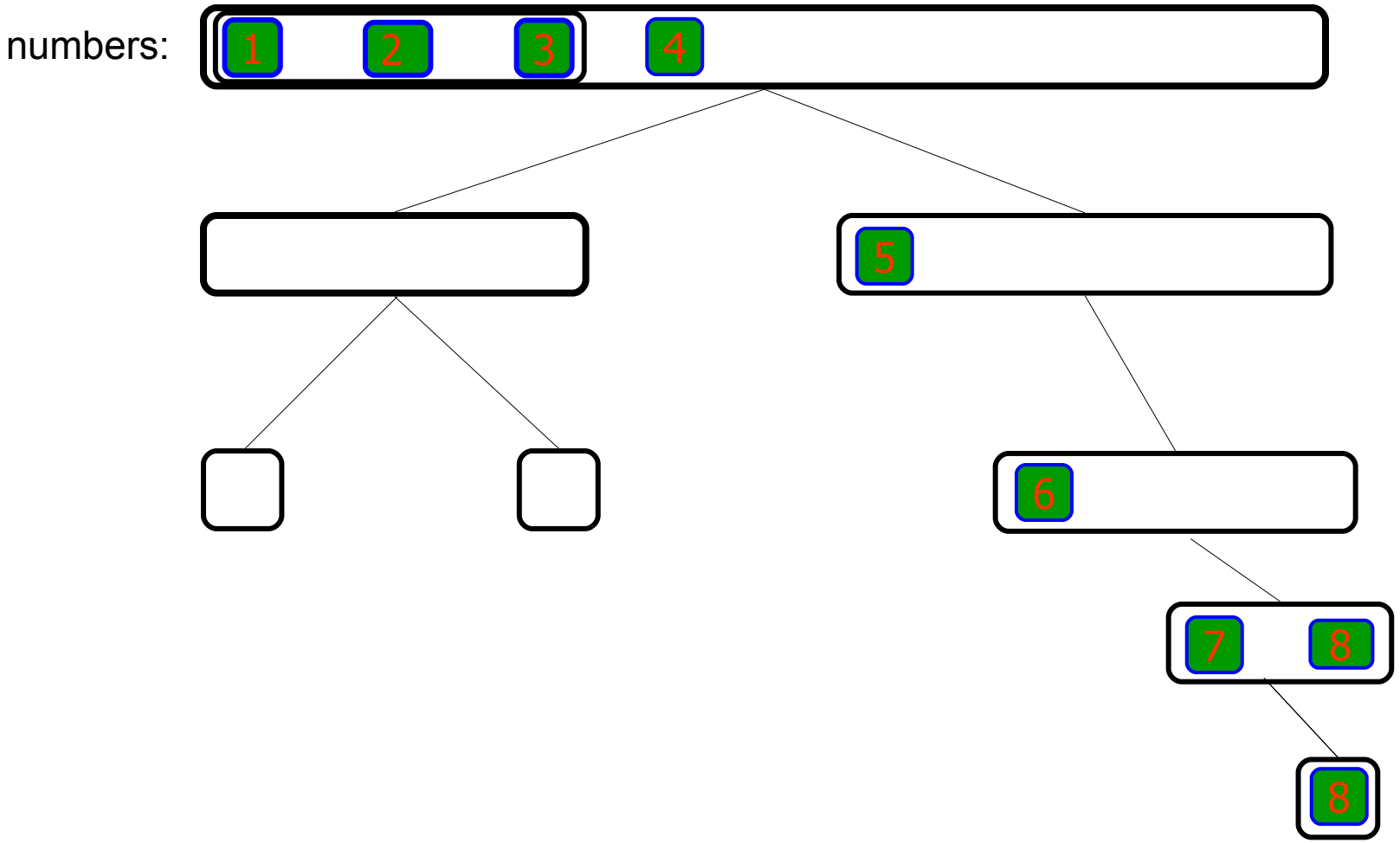
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 6, 7) , particiona



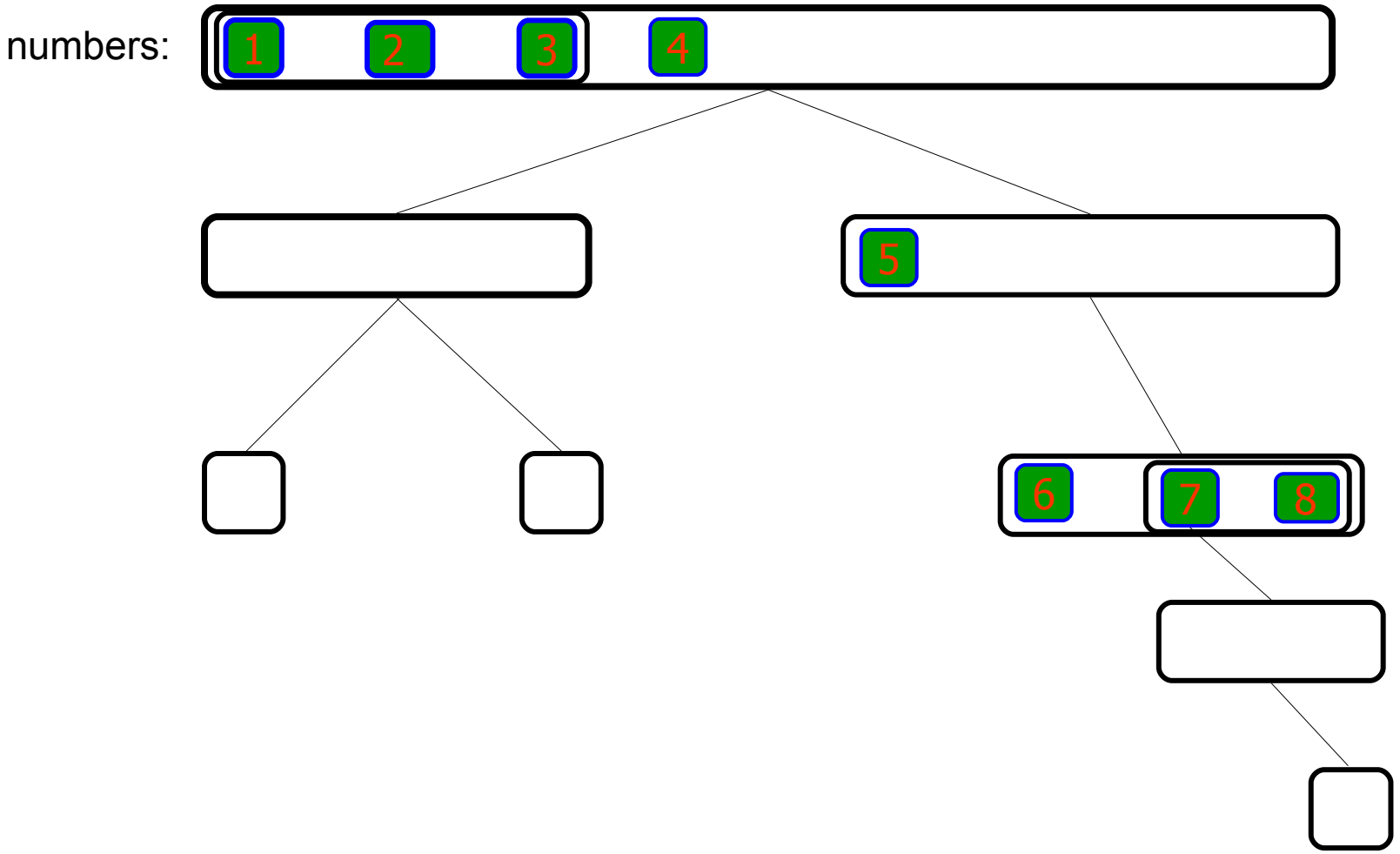
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 7, 7) , return base case



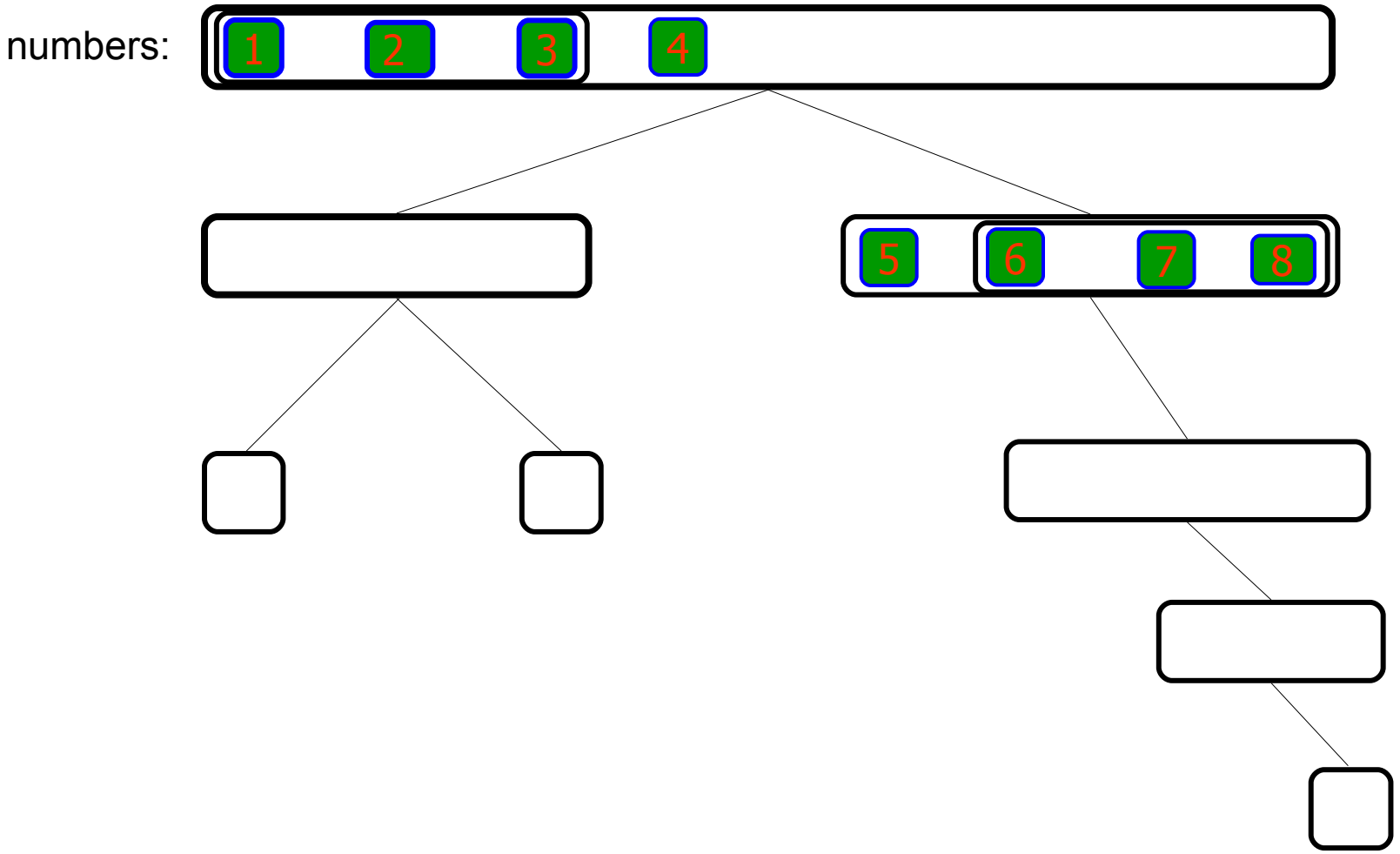
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 6, 7) , retorna



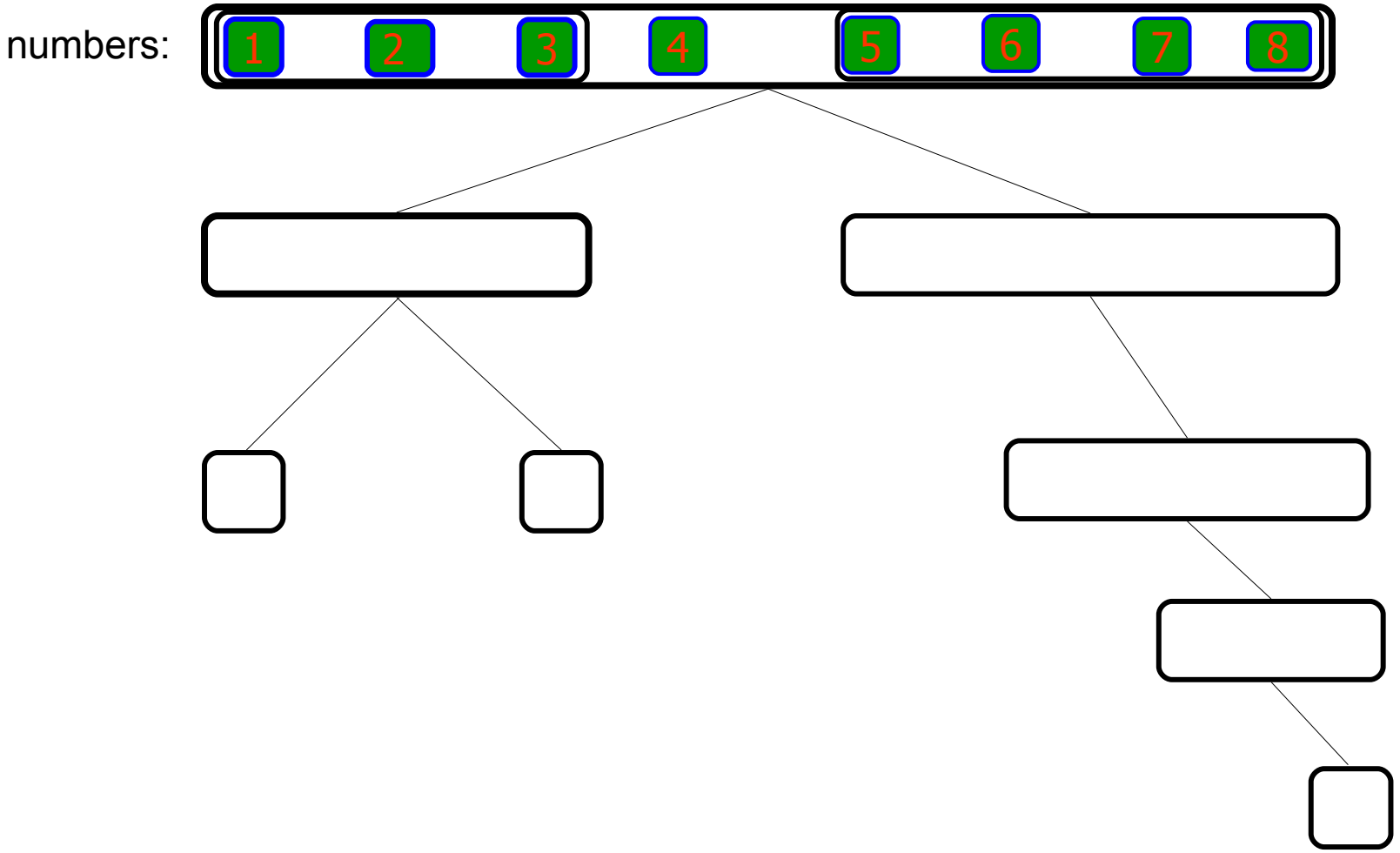
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 5, 7) , retorna



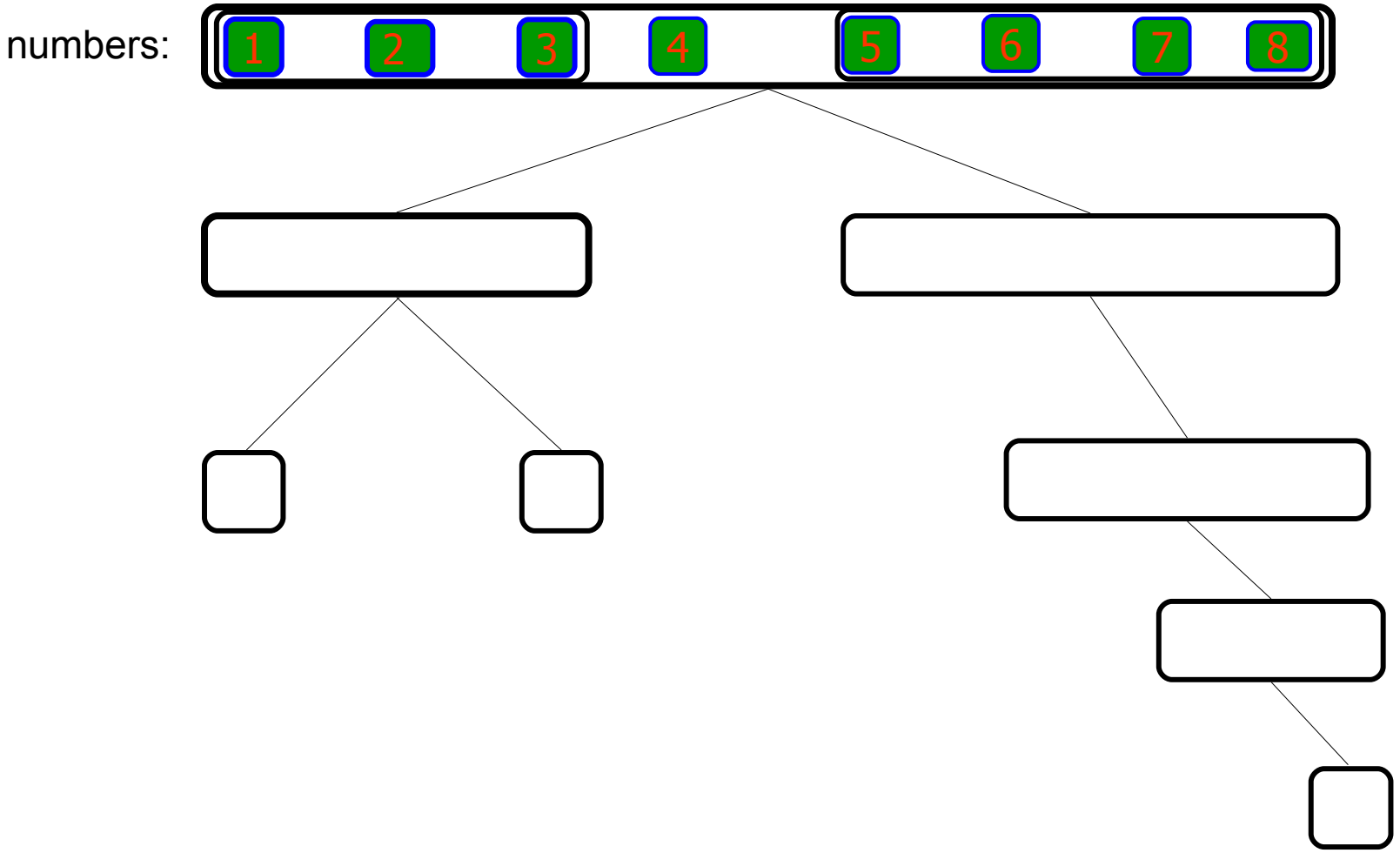
Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 4, 7) , retorna



Quick-Sort(numbers, 0, 7)

Quick-Sort(numbers, 0, 7) , Fim! Ordenado!



Quick Sort

```
Quick-Sort(numbers, esq, dir)
  if    esq ≥ dir  (Não faz Nada! – Ordenado!)
  else
      pivo ← Particiona(numbers, esq, dir)
      Quick-Sort(numbers, esq, pivo-1 )
      Quick-Sort(numbers, pivo+1, dir)
  end if
```

Heap Sort

Seleção em árvore

Até a próxima...