ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES II

Algoritmos clássicos de ordenação II

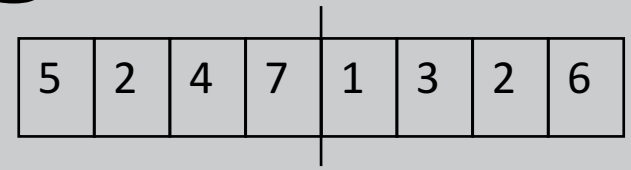
Merge sort

Também chamado de ordenação por intercalação.

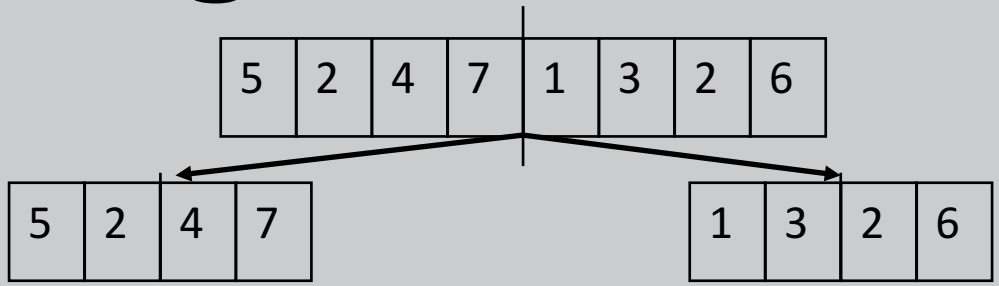
Ideia básica: dividir para conquistar

- Um vetor v é dividido em duas partes, recursivamente.
- Cada metade é ordenada e ambas são intercaladas formando o vetor ordenado.
- Usa um vetor auxiliar para intercalar.

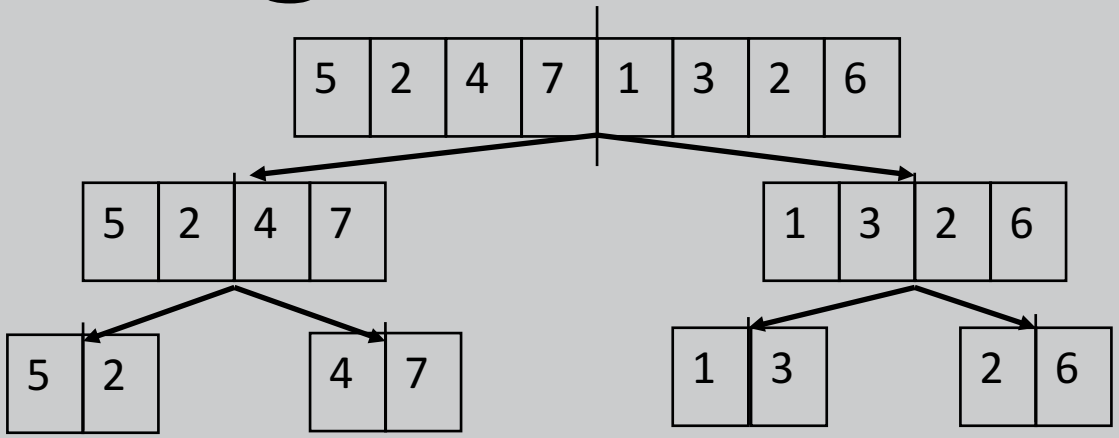
Merge sort



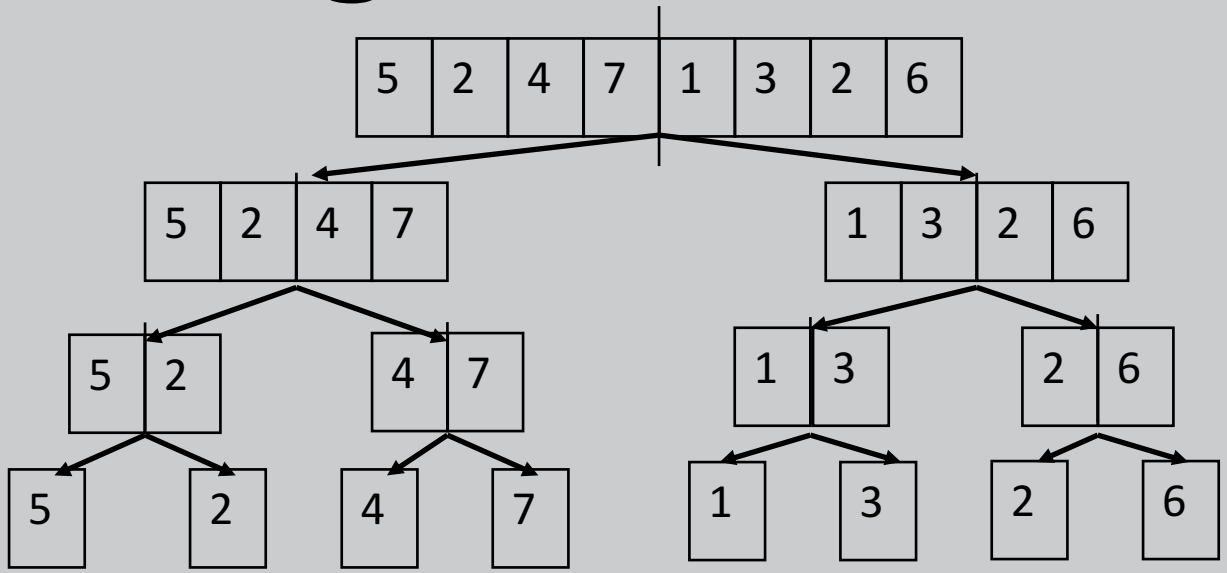
Merge sort



Merge sort

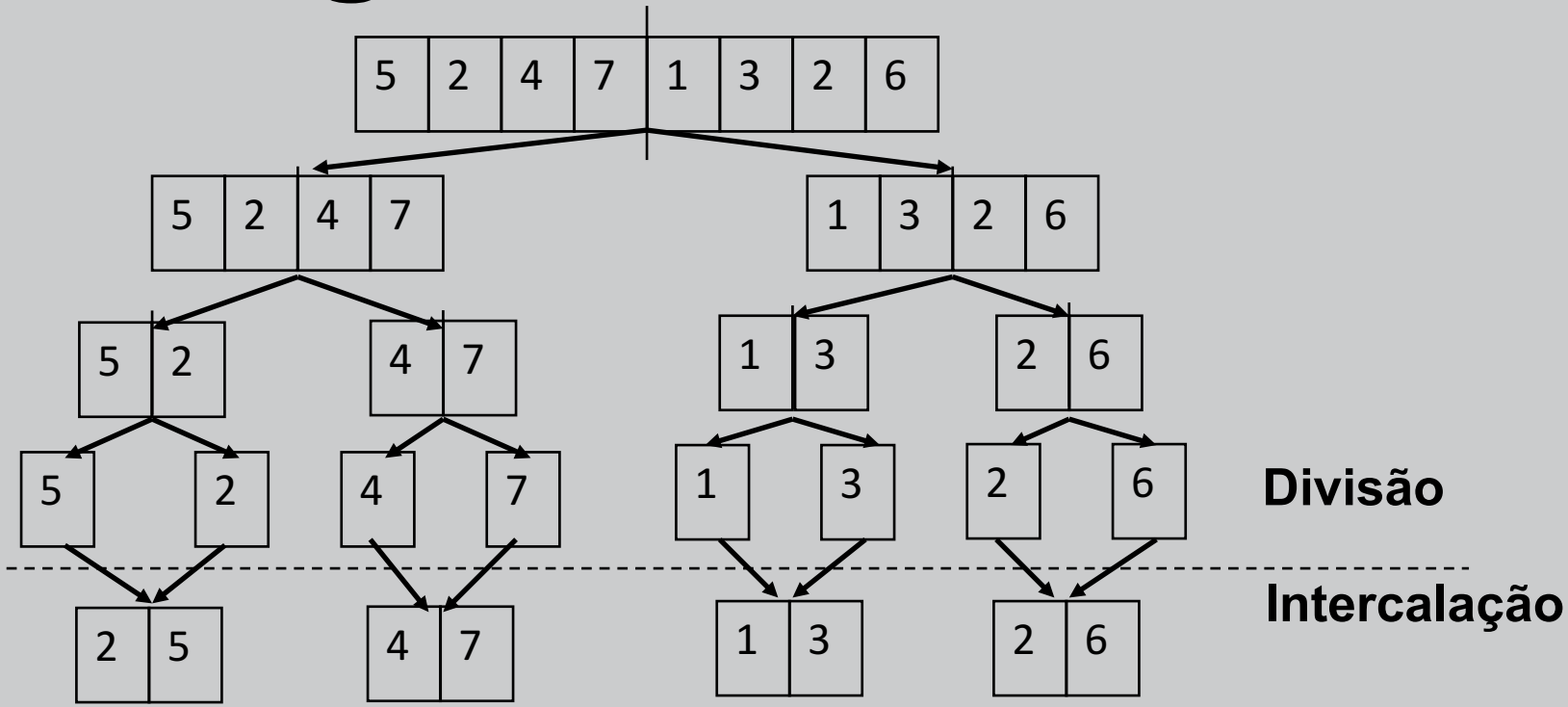


Merge sort

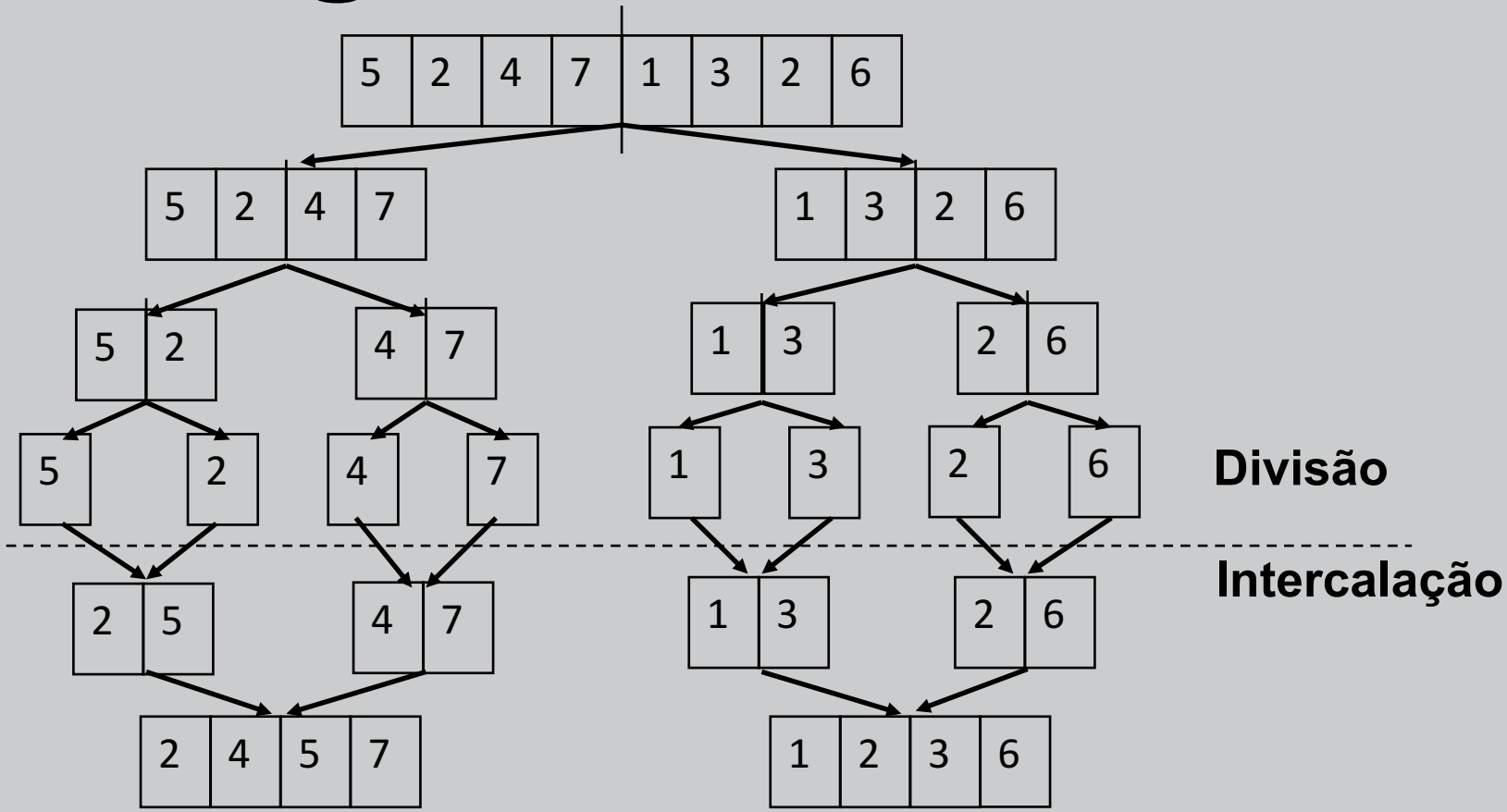


Divisão

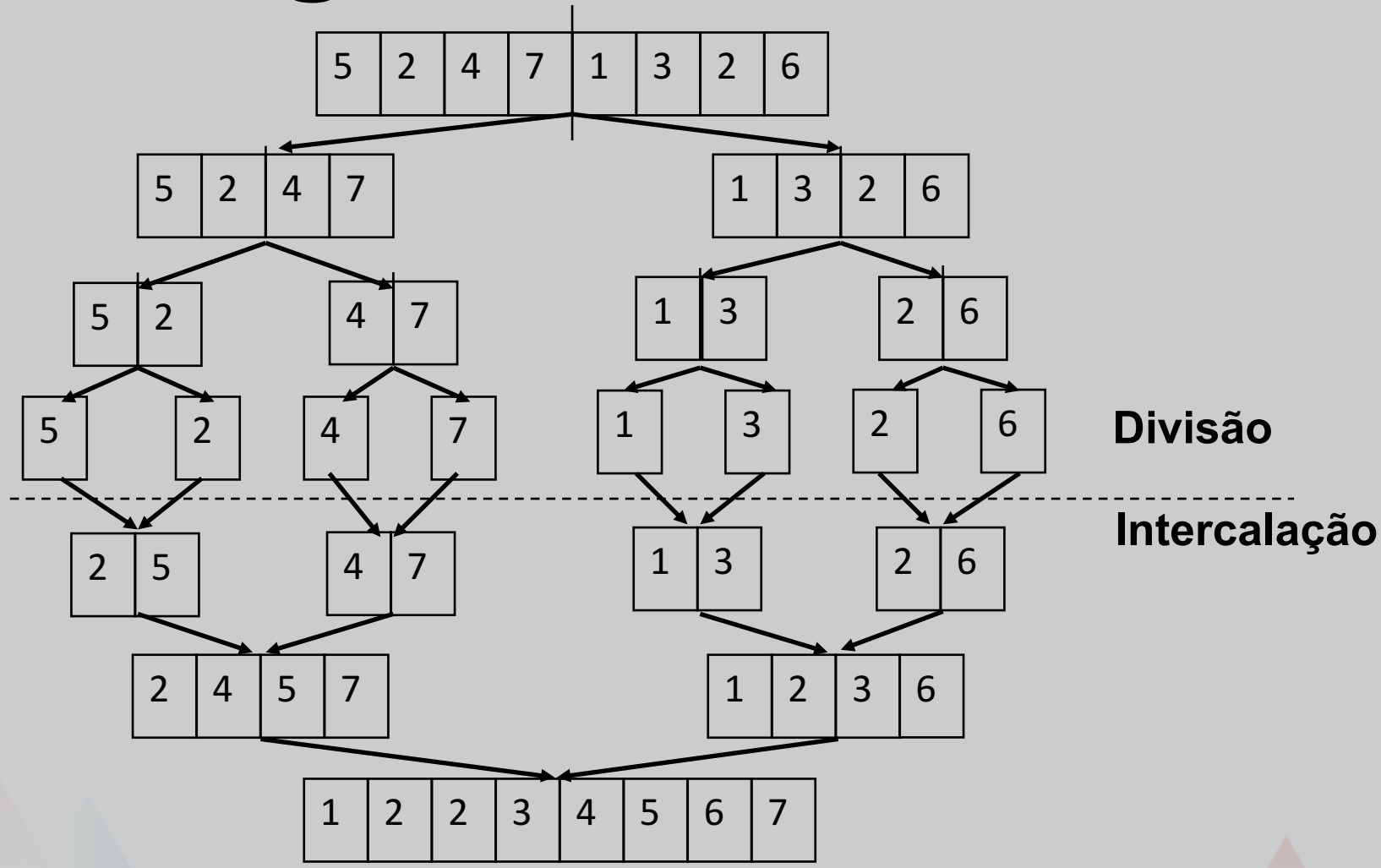
Merge sort


5 2 4 7 1 3 2 6
5 2 4 7
1 3 2 6
5 2
4 7
1 3
2 6
5
2
4
7
1
3
2
6
2 5
4 7
1 3
2 6
Divisão
Intercalação

Merge sort


5 2 4 7 1 3 2 6
5 2 4 7
1 3 2 6
5 2
4 7
1 3
2 6
5
2
4
7
1
3
2
6
Divisão
Intercalação
2 5
4 7
1 3
2 6
2 4 5 7
1 2 3 6

Merge sort


5 2 4 7 1 3 2 6
5 2 4 7
1 3 2 6
5 2
4 7
1 3
2 6
5
2
4
7
1
3
2
6
Divisão
Intercalação
2 5
4 7
1 3
2 6
2 4 5 7
1 2 3 6
1 2 2 3 4 5 6 7

Merge sort

Implementação em Python

```
def merge_sort(v, ini, fim):
    if ini < fim:
        meio = (ini + fim) // 2
        merge_sort(v, ini, meio)
        merge_sort(v, meio+1,
fim)
        intercala(v, ini, meio,
fim)
```

```
def intercala(v, ini, meio, fim):
    L = v[ini:meio+1]
    R = v[meio+1:fim+1]
    L.append(999) #sentinela
    R.append(999) #sentinela
    i = 0
    j = 0
    for k in range(ini, fim+1):
        if L[i] <= R[j]:
            v[k] = L[i]
            i += 1
        else:
            v[k] = R[j]
            j += 1
```

Quicksort

Algoritmo também baseado na estratégia dividir para conquistar.

Ideia básica:

- **Dividir o vetor em dois vetores menores que serão ordenados independentemente e combinados para produzir o resultado final.**

Quicksort: primeiro passo

Escolha de um elemento pivô x, colocando-o em sua posição correta

- **Ordenar de forma que os elementos à esquerda do pivô são menores ou iguais a ele e os elementos à direita são maiores ou iguais a ele**
  - **Percorrer o vetor v da esquerda para direita até v[i] >= x; e da direita para esquerda até que v[j] <= x.**
    - **Trocar v[i] com v[j], incrementar i, decrementar j**
  - **Quando i e j se cruzarem, a iteração finaliza, de forma que v[0]...v[j] são menores ou iguais a x e v[i]...v[n-1] são maiores ou iguais a x**

Quick sort: segundo passo

Ordenar subvetores à esquerda e à direita do elemento pivô.

Quicksort: exemplo

25 57 35 37 12 86 92 33

Quicksort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 ponteiros inicializados

i → ← j

Quicksort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 ponteiros inicializados

i → ← j

25 57 35 37 12 86 92 33 procura-se i>=pivô

i j

Quicksort: exemplo

pivô=v[(0+7)/2]=37

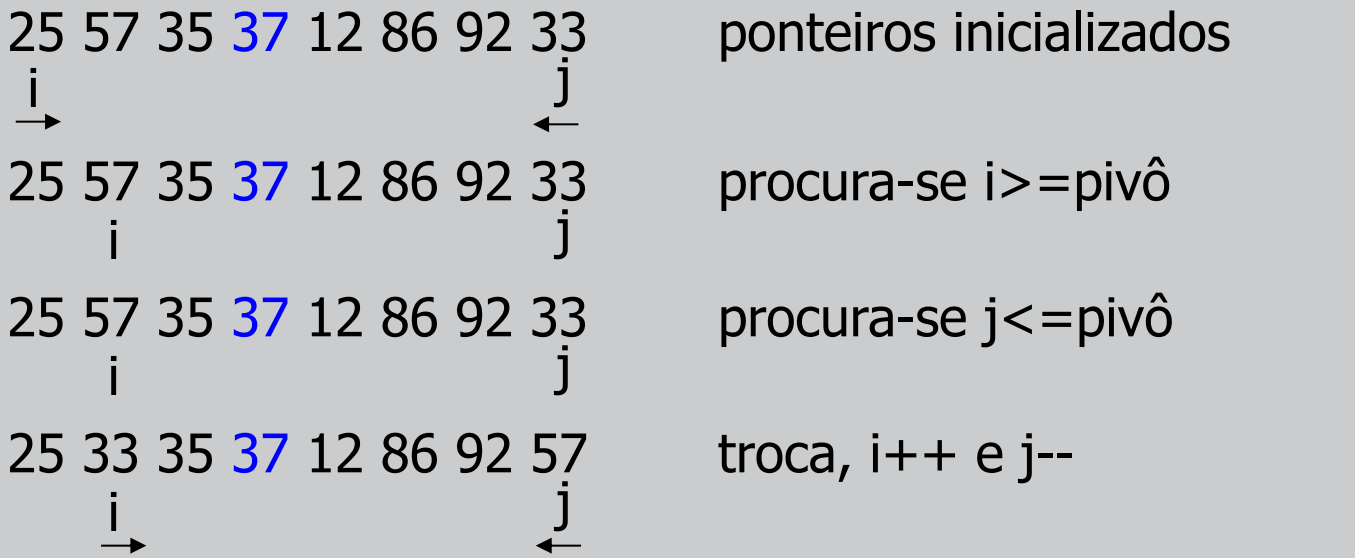
25 57 35 37 12 86 92 33 ponteiros inicializados
i → ← j

25 57 35 37 12 86 92 33 procura-se i>=pivô
i j

25 57 35 37 12 86 92 33 procura-se j<=pivô
i j

Quicksort: exemplo

pivô=v[(0+7)/2]=37



Quicksort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 i → ← j	ponteiros inicializados
25 57 35 37 12 86 92 33 i j	procura-se i>=pivô
25 57 35 37 12 86 92 33 i j	procura-se j<=pivô
25 33 35 37 12 86 92 57 i j	troca, i++ e j--
25 33 35 37 12 86 92 57 i j	procura-se i>=pivô

Quicksort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 i → ← j	ponteiros inicializados
25 57 35 37 12 86 92 33 i j	procura-se i>=pivô
25 57 35 37 12 86 92 33 i j	procura-se j<=pivô
25 33 35 37 12 86 92 57 i j	troca, i++ e j--
25 33 35 37 12 86 92 57 i j	procura-se i>=pivô
25 33 35 37 12 86 92 57 i j	procura-se j<=pivô

Quicksort: exemplo

pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 — ponteiros inicializados
i → ← j

25 57 35 37 12 86 92 33 — procura-se i>=pivô
i j

25 57 35 37 12 86 92 33 — procura-se j<=pivô
i j

25 33 35 37 12 86 92 57 — troca, i++ e j--
i j

25 33 35 37 12 86 92 57 — procura-se i>=pivô
i j

25 33 35 37 12 86 92 57 — procura-se j<=pivô
i j

25 33 35 12 37 86 92 57 — troca, i++ e j--
i → ← j

Quicksort: exemplo

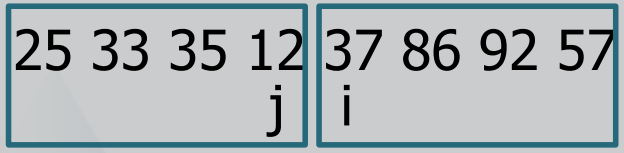
pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57
j i

como i e j se cruzaram, fim do processo

Todos à esquerda do pivô são menores ou iguais a ele
→ v[0]...v[j]<=pivô

Todos à direita do pivô são maiores ou iguais a ele
→ v[i]...v[n-1]>=pivô



Nova iteração com subvetores:
v[0]...v[j] e v[i]...v[n-1]

Quicksort: exercício

Fazer as ordenações dos subvetores, repetindo o processo

25 33 35 12 37 86 92 57

Quicksort

Implementação em Python

```
def quick_sort(v, ini, fim):
    meio = (ini + fim) // 2
    pivo = v[meio]
    i = ini
    j = fim
    while i < j:
        while v[i] < pivo:
            i += 1
        while v[j] > pivo:
            j -= 1
        if i <= j:
            v[i], v[j] = v[j], v[i]
        i += 1
        j -= 1
    if j > ini:
        quick_sort(v, ini, j)
    if i < fim:
        quick_sort(v, i, fim)
```

ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES II

Algoritmos clássicos de ordenação II