SCC-601– Introdução à Ciência da Computação II

Ordenação e Complexidade – Parte 6

Lucas Antiqueira

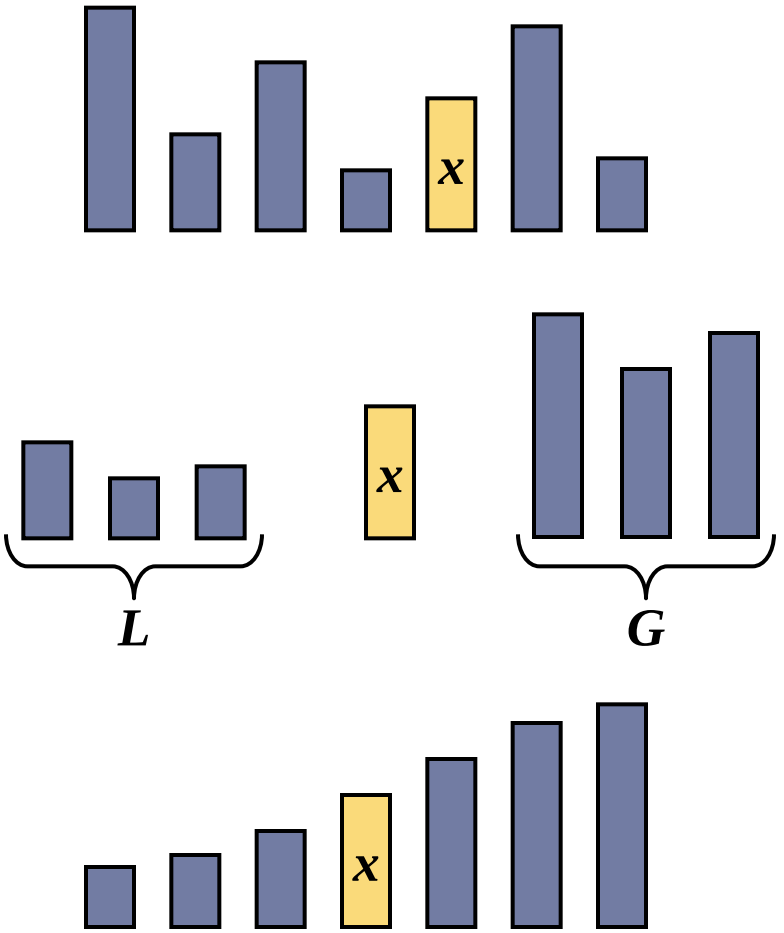
QuickSort

QuickSort

- Melhoramento do BubbleSort
  - Idéia: Troca entre elementos distantes são mais efetivas
- Assim como o MergeSort, utiliza a bordagem dividir para conquistar

QuickSort

- Divisão: selecione um elemento x (chamado pivô) e divida v em:
  - L elementos menores ou iguais a x
  - G elementos maiores ou iguais a x
- Recursão: ordenar L e G



QuickSort

- Dado um elemento pivô x
  - Percorrer o vetor v da esquerda para a direita até v[i] >= x; e da direita para a esquerda até v[j] <= x
    - Troca v[i] com v[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

- A seguir:
  - Ordenar sub-vetores à esquerda e à direita do elemento pivô

QuickSort

25 57 35 37 12 86 92 33 ⟶ Vetor ***v*** a ser ordenado

QuickSort

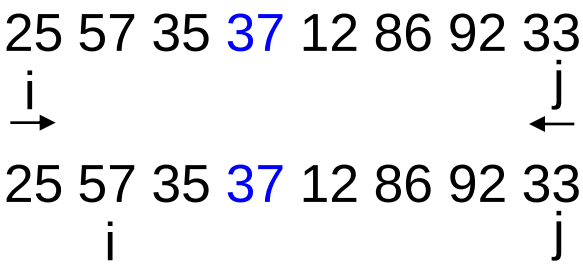
pivô=v[(0+7)/2]=37

25 57 35 37 12 86 92 33 ponteiros inicializados

i → j ←

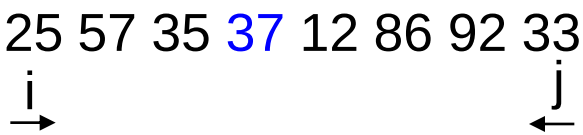
QuickSort

pivô=v[(0+7)/2]=37

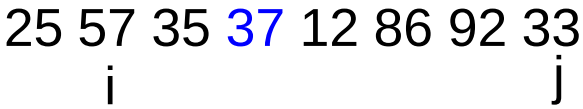


QuickSort

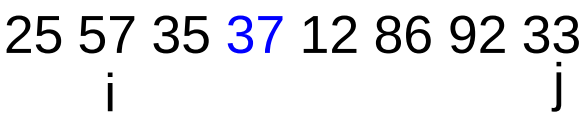
pivô=v[(0+7)/2]=37



ponteiros inicializados



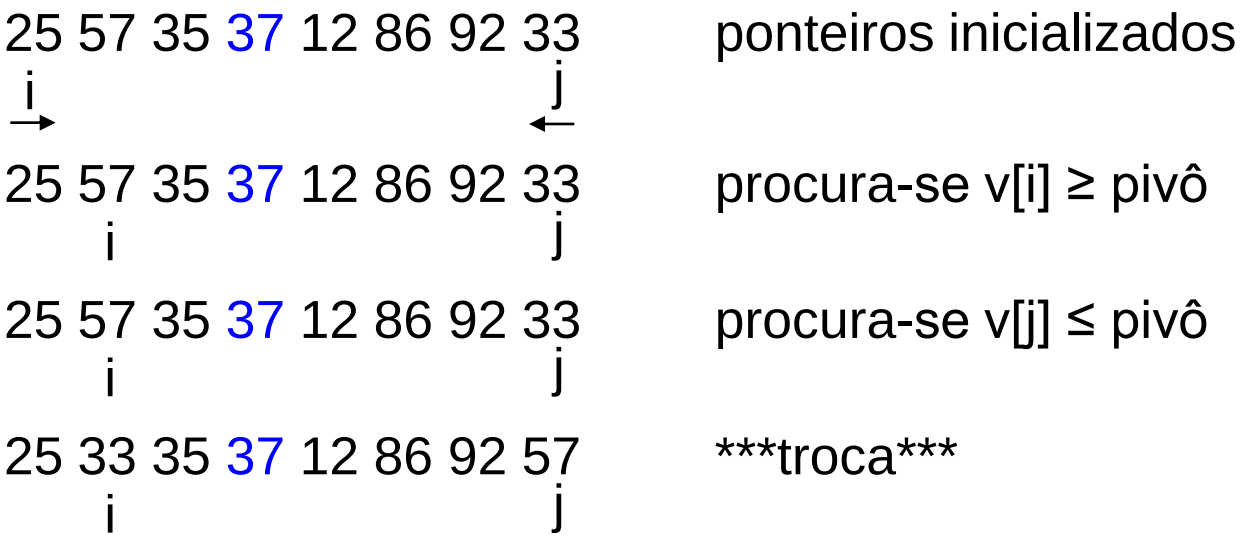
procura-se v[i] ≥ pivô



procura-se v[j] ≤ pivô

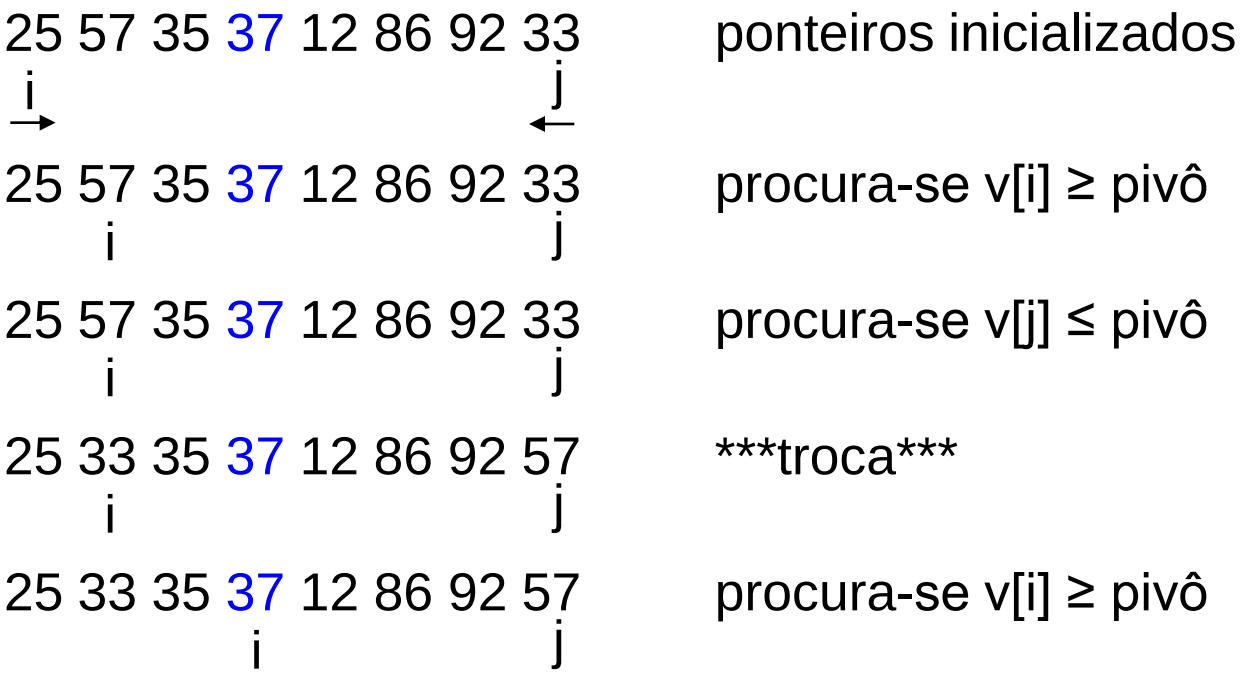
QuickSort

pivô=v[(0+7)/2]=37



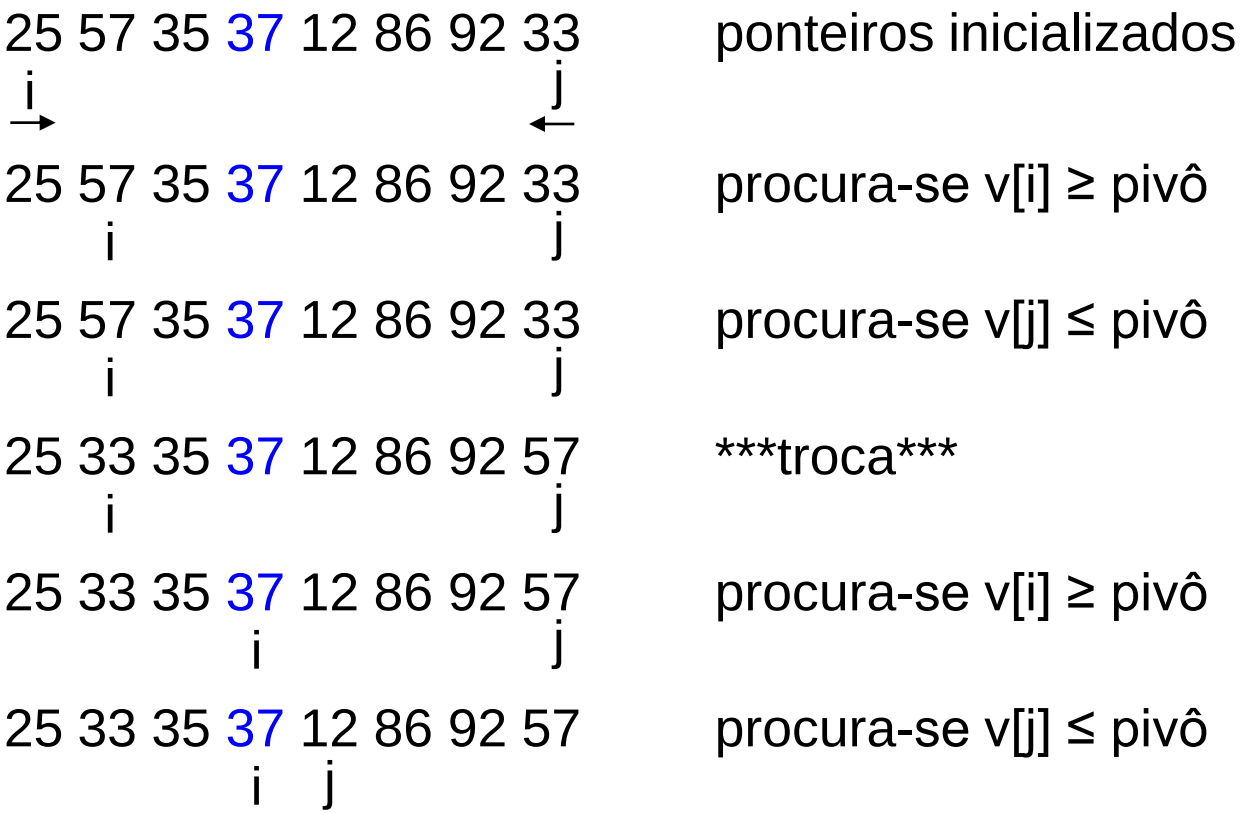
QuickSort

pivô=v[(0+7)/2]=37



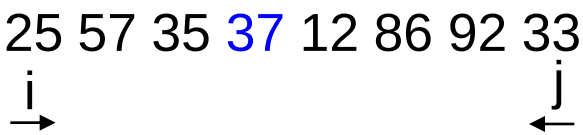
QuickSort

pivô=v[(0+7)/2]=37

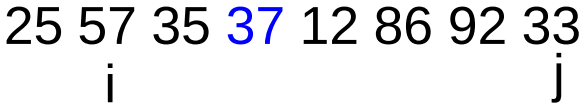


QuickSort

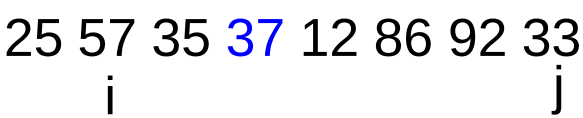
pivô=v[(0+7)/2]=37



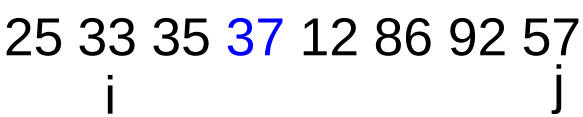
ponteiros inicializados



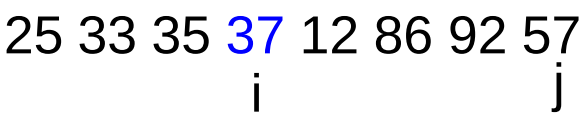
procura-se v[i] ≥ pivô



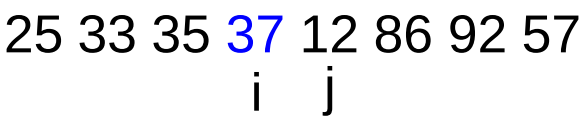
procura-se v[j] ≤ pivô



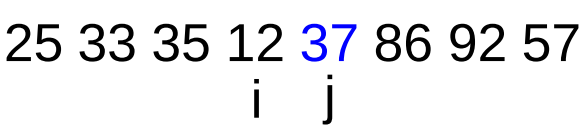
troca



procura-se v[i] ≥ pivô



procura-se v[j] ≤ pivô



troca

QuickSort

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57 procura-se v[i] ≥ pivô
i j

QuickSort

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57
i j

procura-se v[i] ≥ pivô

25 33 35 12 37 86 92 57
j i

procura-se v[j] ≤ pivô

→ i e j se cruzaram, fim do processo

QuickSort

pivô=v[(0+7)/2]=37

25 33 35 12 37 86 92 57 — procura-se v[i] ≥ pivô

i j

25 33 35 12 37 86 92 57 — procura-se v[j] ≤ pivô

j i → i e j se cruzaram, fim do processo

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

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

QuickSort

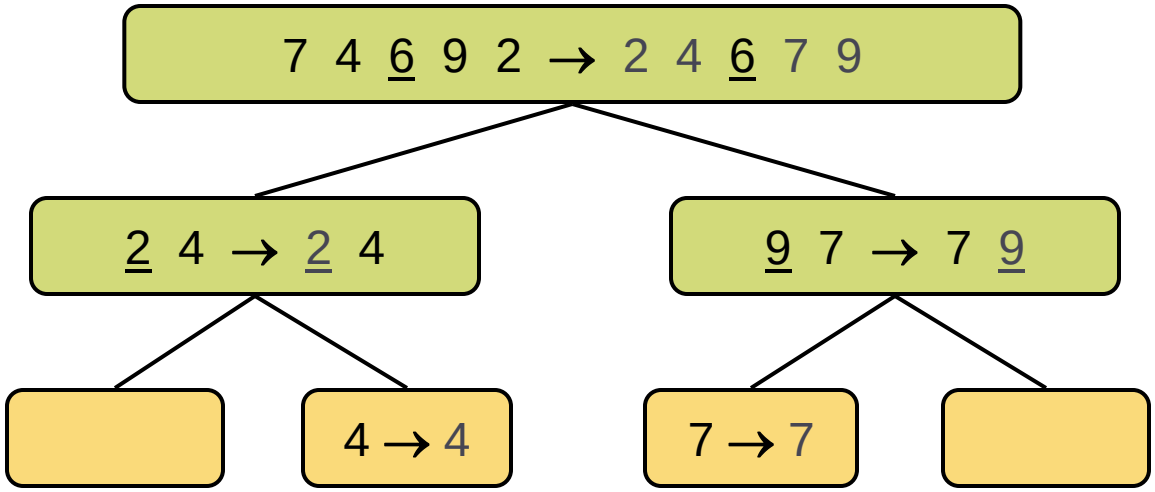
- Exercício: Ordene os subvetores, repetindo o processo para cada um deles.

25 33 35 12 (37) 86 92 57

O processo é aplicado até que toda partição contenha apenas um (ou nenhum) item.

QuickSort

- Uma execução do QuickSort pode ser representada por uma árvore binária
  - Cada nó representa uma chamada recursiva e exibe:
    1. A sequência não ordenada e seu pivô
    2. A sequência ordenada ao final da chamada recursiva
  - A raiz representa a chamada inicial a QuickSort
  - As folhas tem 0 ou 1 elementos



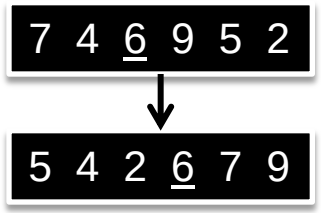
```
void quicksort(int v[], int inf, int sup) {
    int aux, meio, pivo, i, j;

    if (inf < sup) {
        meio = (inf + sup) / 2;
        pivo = v[meio];
        v[meio] = v[sup];
        v[sup] = pivo;
        i = inf - 1;
        j = sup;
        do {
            do {
                i++;
            } while (v[i] < pivo);
            do {
                j--;
            } while ((v[j] > pivo) && (j > inf));
            if (i < j) {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        } while (i < j);
        v[sup] = v[i];
        v[i] = pivo;
        quicksort(v, inf, i - 1);
        quicksort(v, i + 1, sup);
    }
}
```

```
void quicksort(int v[], int inf, int sup) {
    int aux, meio, pivo, i, j;

    if (inf < sup) {
        meio = (inf + sup) / 2;
        pivo = v[meio];
        v[meio] = v[sup];
        v[sup] = pivo;
        i = inf - 1;
        j = sup;
        do {
            do {
                i++;
            } while (v[i] < pivo);
            do {
                j--;
            } while ((v[j] > pivo) && (j > inf));
            if (i < j) {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        } while (i < j);
        v[sup] = v[i];
        v[i] = pivo;
        quicksort(v, inf, i - 1);
        quicksort(v, i + 1, sup);
    }
}
```

Troca pivô com a última posição.

Por que?

```
void quicksort(int v[], int inf, int sup) {
    int aux, meio, pivo, i, j;

    if (inf < sup) {
        meio = (inf + sup) / 2;
        pivo = v[meio];
        v[meio] = v[sup];
        v[sup] = pivo;
        i = inf - 1;
        j = sup;
        do {
            do {
                i++;
            } while (v[i] < pivo);
            do {
                j--;
            } while ((v[j] > pivo) && (j > inf));
            if (i < j) {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        } while (i < j);
        v[sup] = v[i];
        v[i] = pivo;
        quicksort(v, inf, i - 1);
        quicksort(v, i + 1, sup);
    }
}
```



Movimento do pivô

```
void quicksort(int v[], int inf, int sup) {
    int aux, meio, pivo, i, j;

    if (inf < sup) {
        meio = (inf + sup) / 2;
        pivo = v[meio];
        v[meio] = v[sup];
        v[sup] = pivo;
        i = inf - 1;
        j = sup;
        do {
            do {
                i++;
            } while (v[i] < pivo);
            do {
                j--;
            } while ((v[j] > pivo) && (j > inf));
            if (i < j) {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        } while (i < j);
        v[sup] = v[i];
        v[i] = pivo;
        quicksort(v, inf, i - 1);
        quicksort(v, i + 1, sup);
    }
}
```

Como o pivô pode mover-se, joga-se o pivô na última posição para, no final, colocá-lo na posição correta

```
void quicksort(int v[], int inf, int sup) {
    int aux, meio, pivo, i, j;

    if (inf < sup) {
        meio = (inf + sup) / 2;
        pivo = v[meio];
        v[meio] = v[sup];
        v[sup] = pivo;
        i = inf - 1;
        j = sup;
        do {
            do {
                i++;
            } while (v[i] < pivo);
            do {
                j--;
            } while ((v[j] > pivo) && (j > inf));
            if (i < j) {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        } while (i < j);
        v[sup] = v[i];
        v[i] = pivo;
        quicksort(v, inf, i - 1);
        quicksort(v, i + 1, sup);
    }
}
```

Encontra posições *i* e *j* para troca

```
void quicksort(int v[], int inf, int sup) {
    int aux, meio, pivo, i, j;

    if (inf < sup) {
        meio = (inf + sup) / 2;
        pivo = v[meio];
        v[meio] = v[sup];
        v[sup] = pivo;
        i = inf - 1;
        j = sup;
        do {
            do {
                i++;
            } while (v[i] < pivo);
            do {
                j--;
            } while ((v[j] > pivo) && (j > inf));
            if (i < j) {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        } while (i < j);
        v[sup] = v[i];
        v[i] = pivo;
        quicksort(v, inf, i - 1);
        quicksort(v, i + 1, sup);
    }
}
```

Troca quando $i < j$

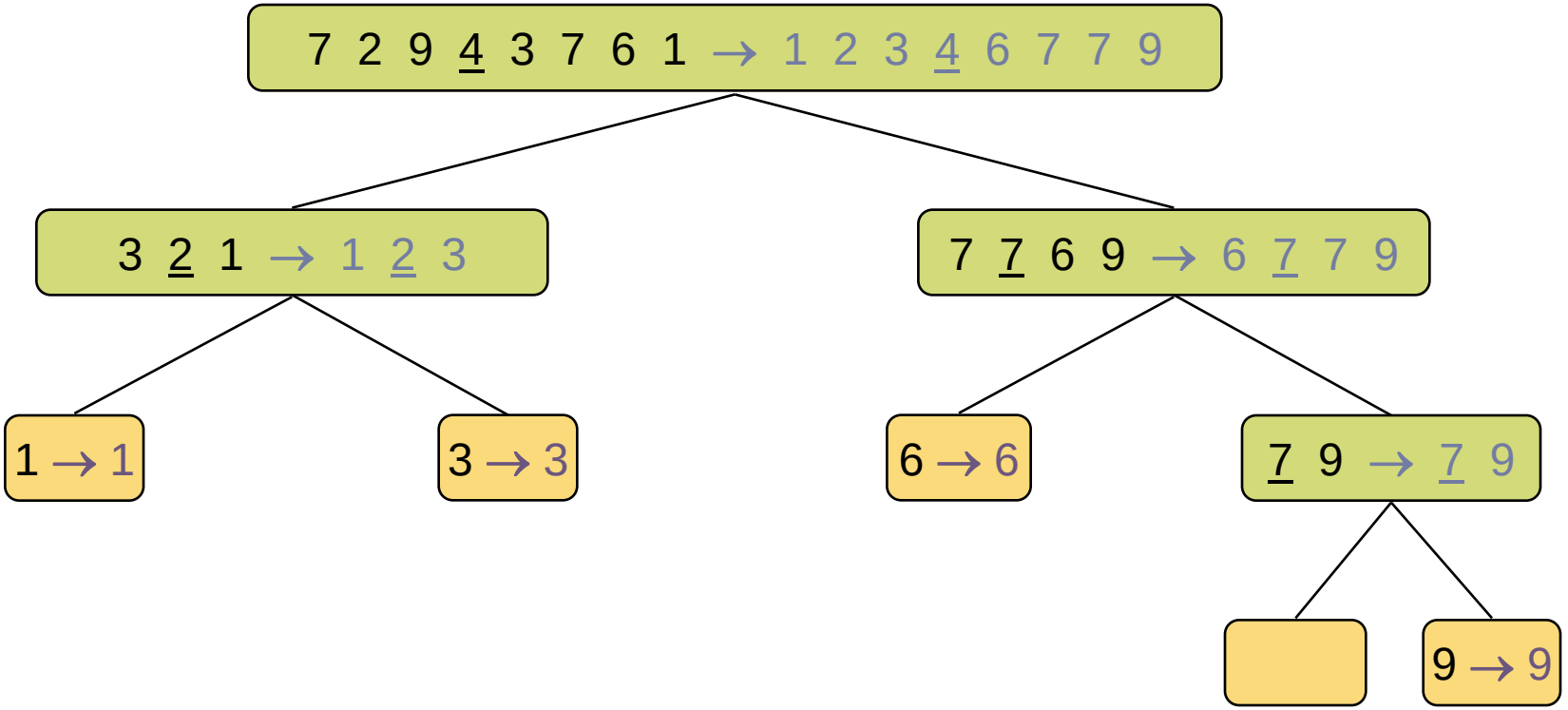
```
void quicksort(int v[], int inf, int sup) {
    int aux, meio, pivo, i, j;

    if (inf < sup) {
        meio = (inf + sup) / 2;
        pivo = v[meio];
        v[meio] = v[sup];
        v[sup] = pivo;
        i = inf - 1;
        j = sup;
        do {
            do {
                i++;
            } while (v[i] < pivo);
            do {
                j--;
            } while ((v[j] > pivo) && (j > inf));
            if (i < j) {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        } while (i < j);
        v[sup] = v[i];
        v[i] = pivo;
        quicksort(v, inf, i - 1);
        quicksort(v, i + 1, sup);
    }
}
```

Quando *i* cruzar com *j*, termina o loop

```
void quicksort(int v[], int inf, int sup) {
    int aux, meio, pivo, i, j;

    if (inf < sup) {
        meio = (inf + sup) / 2;
        pivo = v[meio];
        v[meio] = v[sup];
        v[sup] = pivo;
        i = inf - 1;
        j = sup;
        do {
            do {
                i++;
            } while (v[i] < pivo);
            do {
                j--;
            } while ((v[j] > pivo) && (j > inf));
            if (i < j) {
                aux = v[i];
                v[i] = v[j];
                v[j] = aux;
            }
        } while (i < j);
        v[sup] = v[i];
        v[i] = pivo;
        quicksort(v, inf, i - 1);
        quicksort(v, i + 1, sup);
    }
}
```

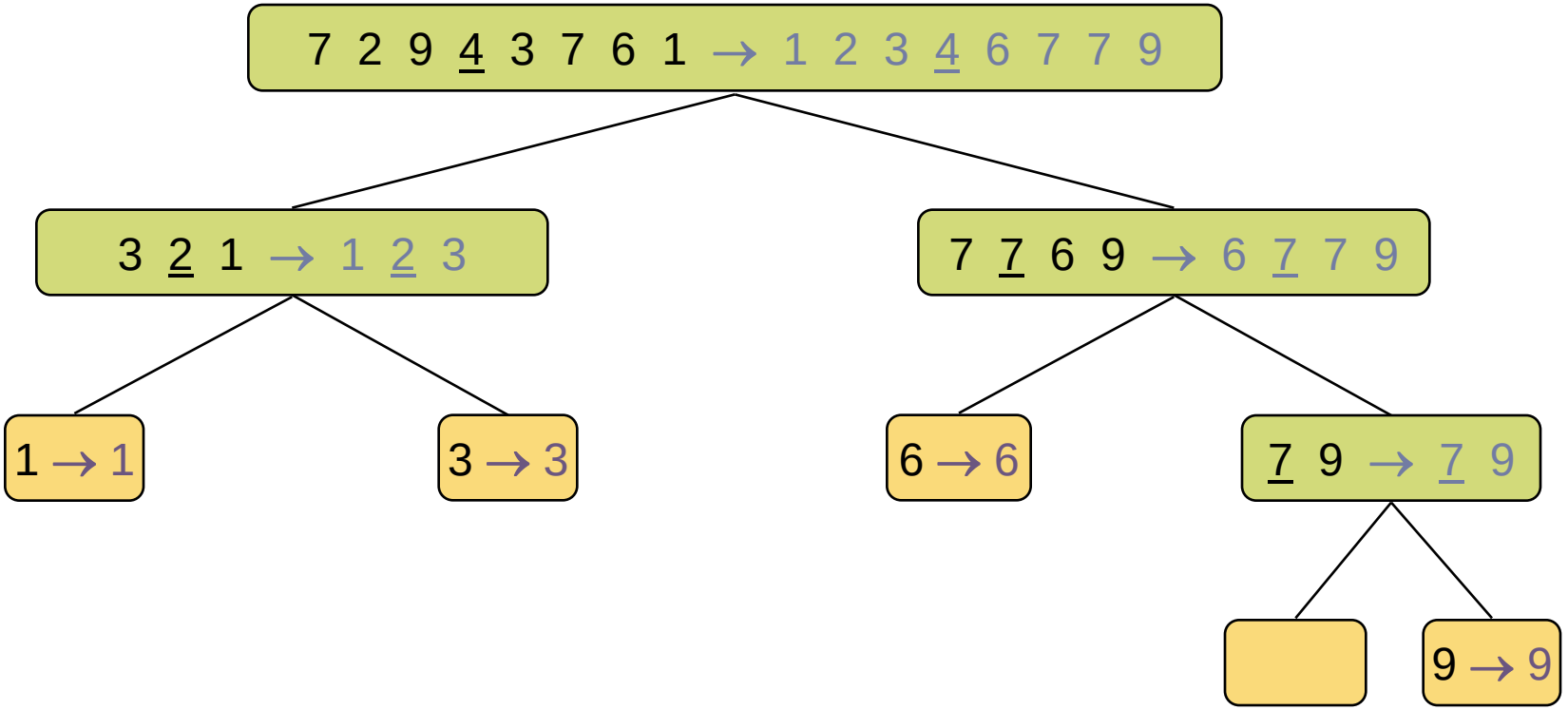
Resolve os subproblemas recursivamente

QuickSort

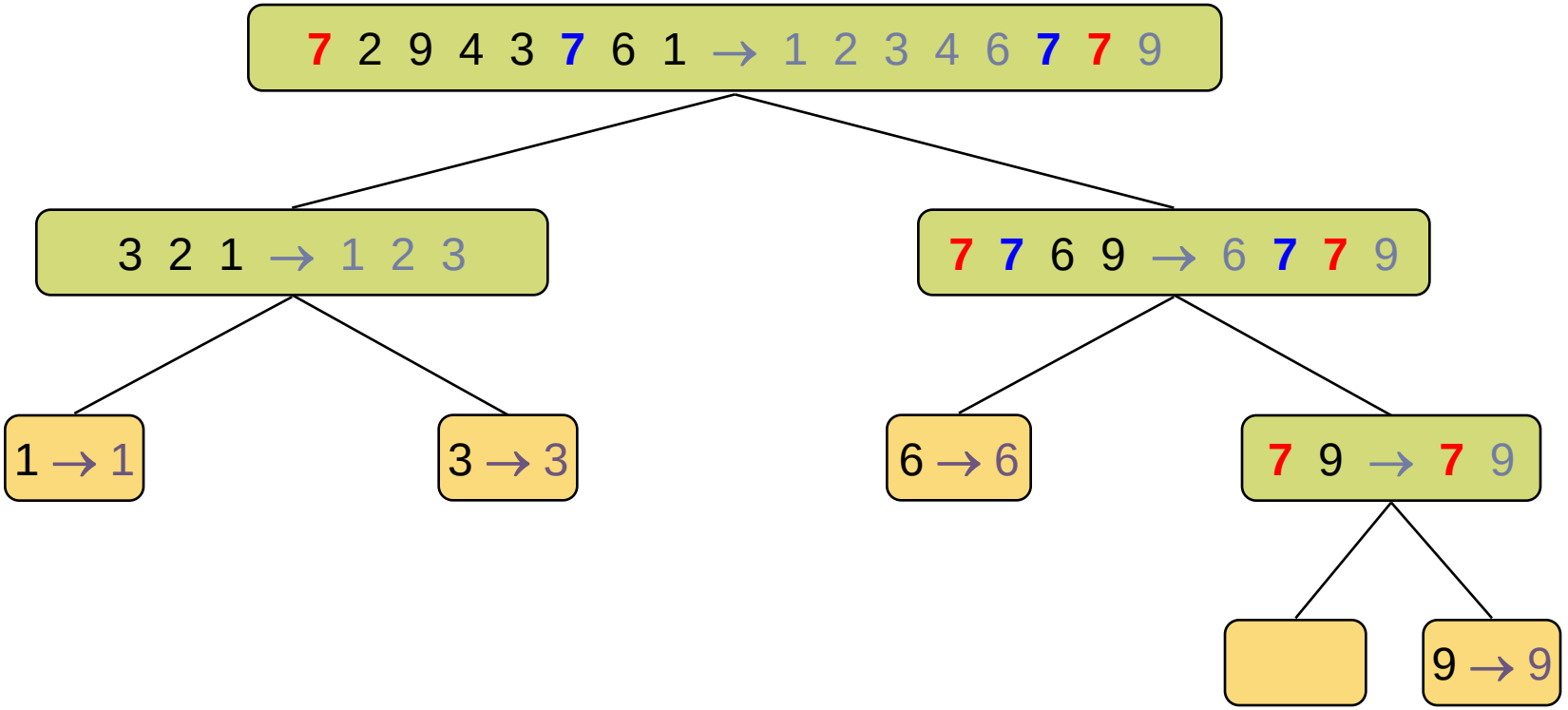


Percorra o algoritmo para esse exemplo.

QuickSort é estável?



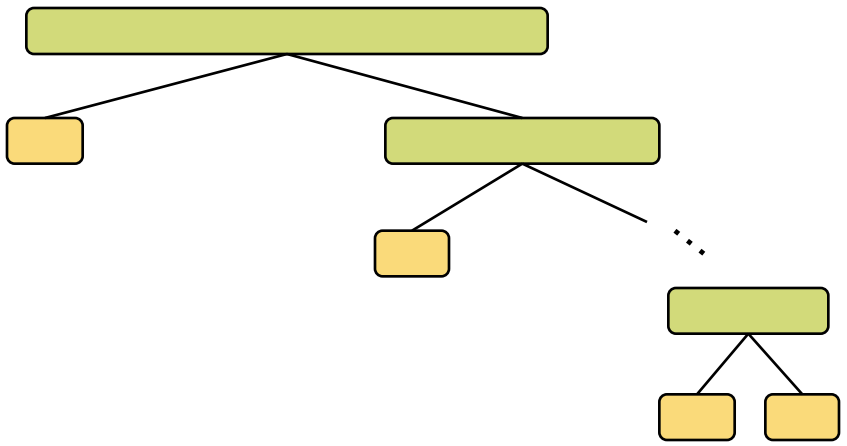
QuickSort é estável? Não



QuickSort

- O pior caso acontece quando o pivô é o elemento mínimo ou máximo do vetor
- L ou G tem tamanho $n - 1$ e o outro tem tamanho 0
- O tempo é proporcional à soma:
  $n + (n - 1) + \ldots + 2 + 1$
- Portanto, o pior caso do QuickSort é $O(n^2)$

Prof.	Tempo
0	n
1	$n - 1$
…	…
$n - 1$	1



QuickSort

- No melhor caso, o pivô é exatamente a mediana do vetor
- L e G contém aproximadamente $n/2$ elementos
- O comportamento assintótico, nesse caso, é o mesmo do MergeSort, pois a árvore binária de recursão é análoga (ou seja, é o mais completa possível)
- Portanto, o melhor caso do QuickSort é $O(n \log_2 n)$

QuickSort

- No melhor caso, o pivô é exatamente a mediana do vetor
- L e G contém aproximadamente $n/2$ elementos
- O comportamento assintótico, nesse caso, é o mesmo do MergeSort, pois a árvore binária de recursão é análoga (ou seja, é o mais completa possível)
- Portanto, o melhor caso do QuickSort é $O(n \log_2 n)$

- Caso médio é também $O(n \log_2 n)$
  - Veja, por exemplo, http://www.im.pwr.wroc.pl/~cichon/Math/QSortAvg.pdf

- Complexidade de espaço $O(n)$

QuickSort

- Um dos algoritmos mais rápidos para uma variedade de situações, sendo frequentemente utilizado
- Como evitar o pior caso?

QuickSort

- Um dos algoritmos mais rápidos para uma variedade de situações, sendo frequentemente utilizado

- Como evitar o pior caso? *Depende da escolha do pivô*

  1. Escolher 3 elementos quaisquer do vetor e usar a mediana deles como pivô
  2. Escolha aleatória do pivô, etc