



Ordenação: QuickSort

Prof. Túlio Toffolo

<http://www.toffolo.com.br>

BCC202 – Aula 16

Algoritmos e Estruturas de Dados I

QuickSort

- Proposto por Hoare em 1960 e publicado em 1962.
- É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.
- Provavelmente é o mais utilizado.

- A idéia básica é dividir o problema de ordenar um conjunto com n itens em dois sub-problemas menores.
- Os problemas menores são ordenados independentemente.
- Os resultados são combinados para produzir a solução final.

- A parte mais delicada do método é o processo de partição.
- O vetor $v[\text{esq}..\text{dir}]$ é rearranjado por meio da escolha arbitrária de um pivô x .
- O vetor v é particionado em duas partes:
 - Parte esquerda: $\text{chaves} \leq x$.
 - Parte direita: $\text{chaves} \geq x$.

- Algoritmo para o particionamento:
 - 1. Escolha arbitrariamente um pivô x .
 - 2. Percorra o vetor a partir da esquerda até que $v[i] \geq x$.
 - 3. Percorra o vetor a partir da direita até que $v[j] \leq x$.
 - 4. Troque $v[i]$ com $v[j]$.
 - 5. Continue este processo até os apontadores i e j se cruzarem.

QuickSort – Após a Partição



Ao final, do algoritmo de partição:

- Vetor $v[\text{esq}..\text{dir}]$ está particionado de tal forma que:
 - Os itens em $v[\text{esq}], v[\text{esq} + 1], \dots, v[j]$ são menores ou iguais a x ;
 - Os itens em $v[i], v[i + 1], \dots, v[\text{dir}]$ são maiores ou iguais a x .

QuickSort – Exemplo

- O pivô x é escolhido como sendo:
 - O elemento central: $v[(i + j) / 2]$.
- Exemplo:

3	6	4	5	1	7	2
---	---	---	---	---	---	---

QuickSort – Exemplo

Primeira
partição



3	6	4	5	1	7	2
---	---	---	---	---	---	---

3	2	4	1	5	7	6
---	---	---	---	---	---	---

Segunda
partição



1	2	4	3
---	---	---	---

Caso especial: pivô já na
posição correta

...

terceira
partição



3	4
---	---

Continua...


QuickSort – Exemplo

quarta
partição

quinta
partição



1	2	3	4		5	7	6
---	---	---	---	--	---	---	---



5	6	7
---	---	---

5	6
---	---

Final

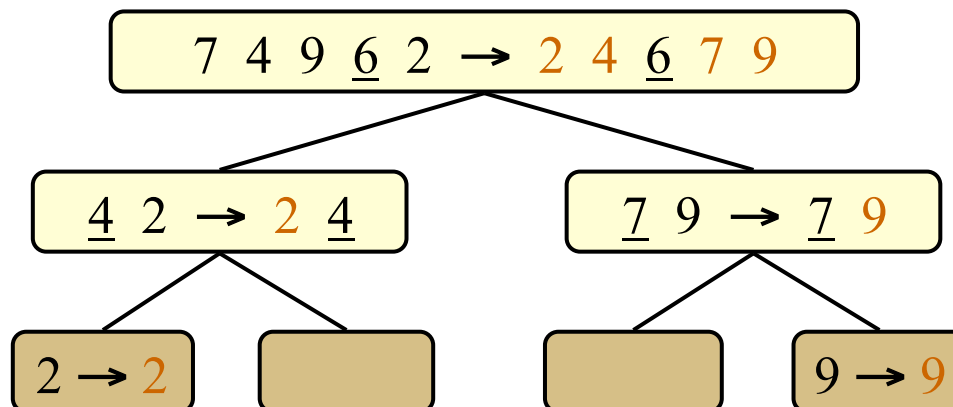
1	2	3	4	5	6	7
---	---	---	---	---	---	---

QUICKSORT

EXEMPLO DE EXECUÇÃO

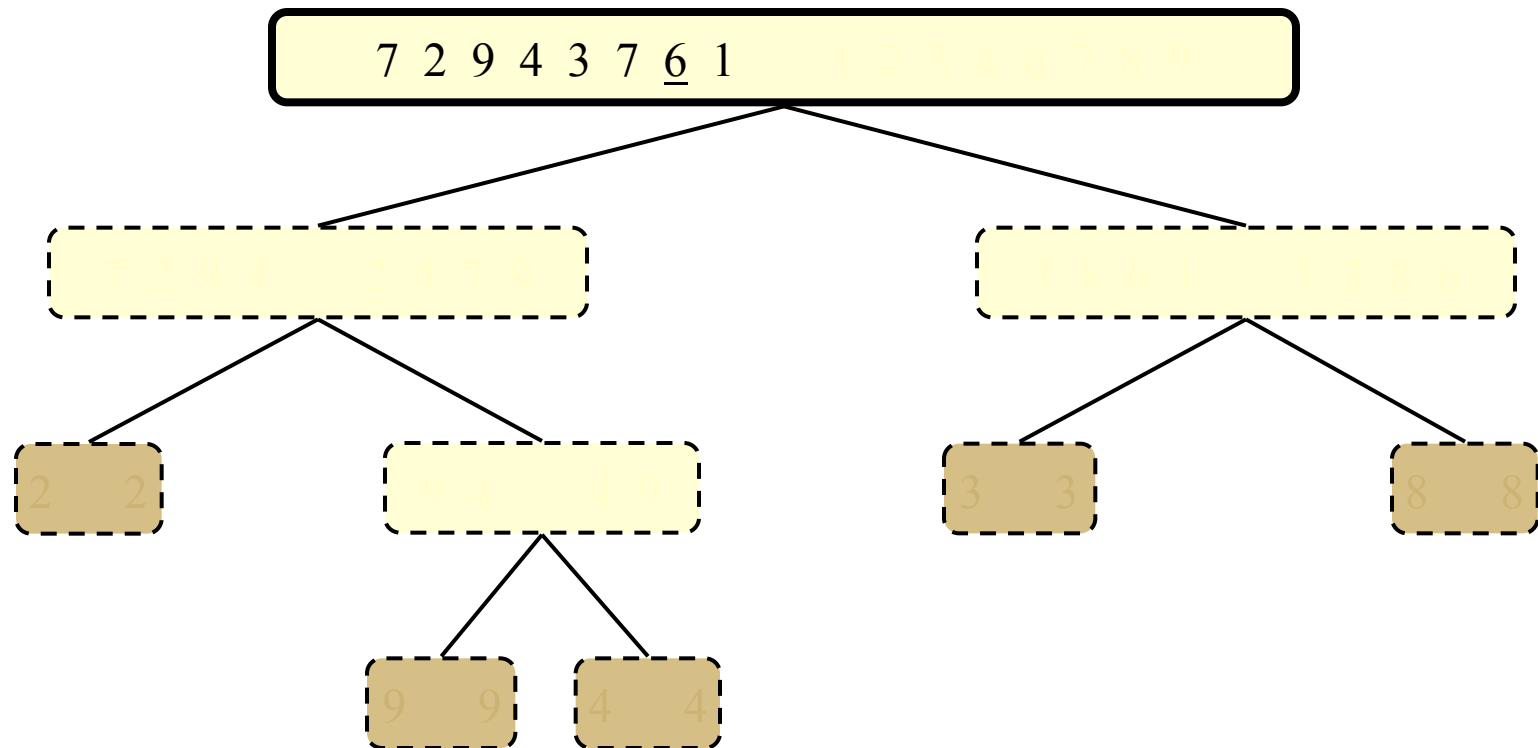
QuickSort – Execução

- A execução do QuickSort pode ser facilmente descrita por uma árvore binária
 - Cada nó representa uma chamada recursiva do QuickSort
 - O nó raiz é a chamada inicial
 - Os nós folhas são vetores de 0 ou 1 número (casos bases)



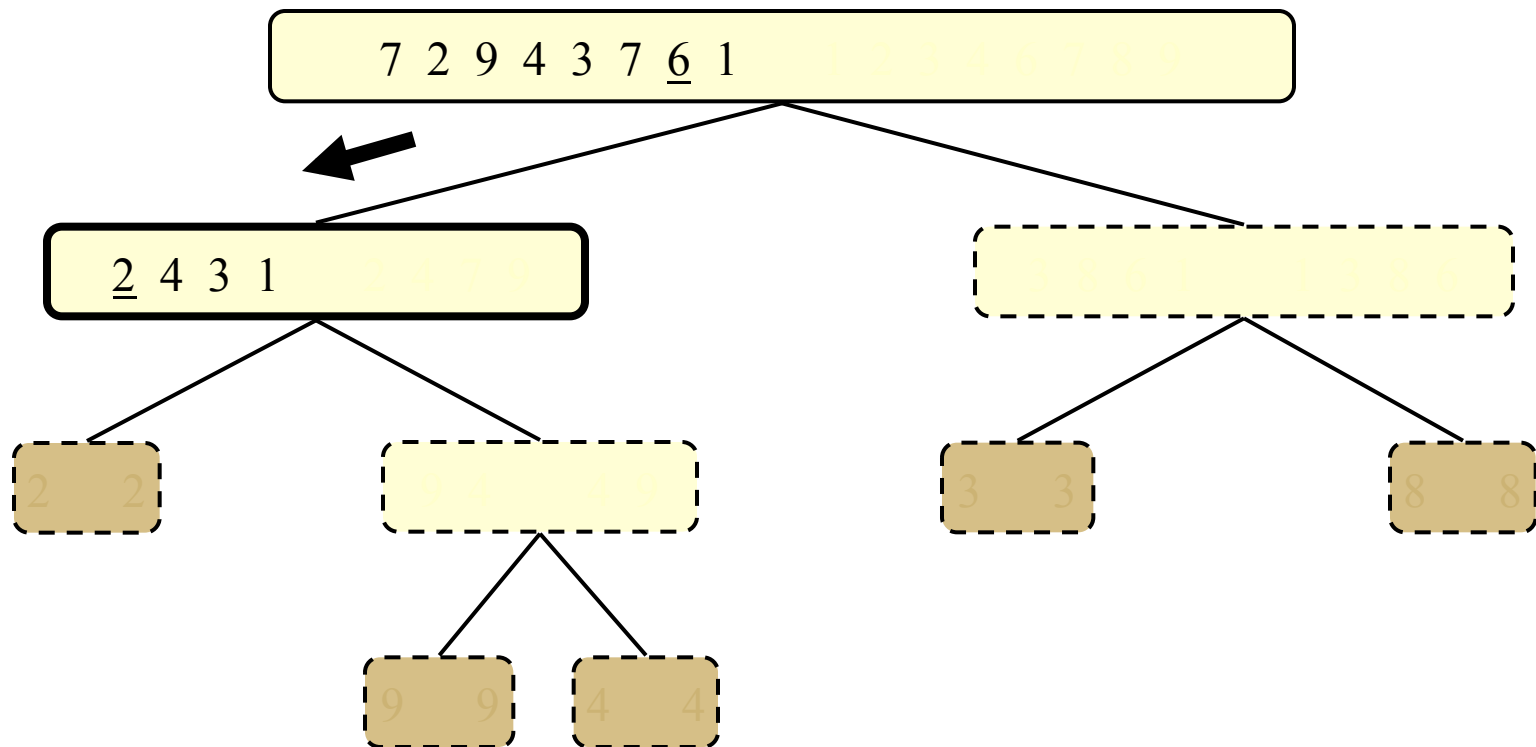
QuickSort: Exemplo de Execução

- Seleção do pivô



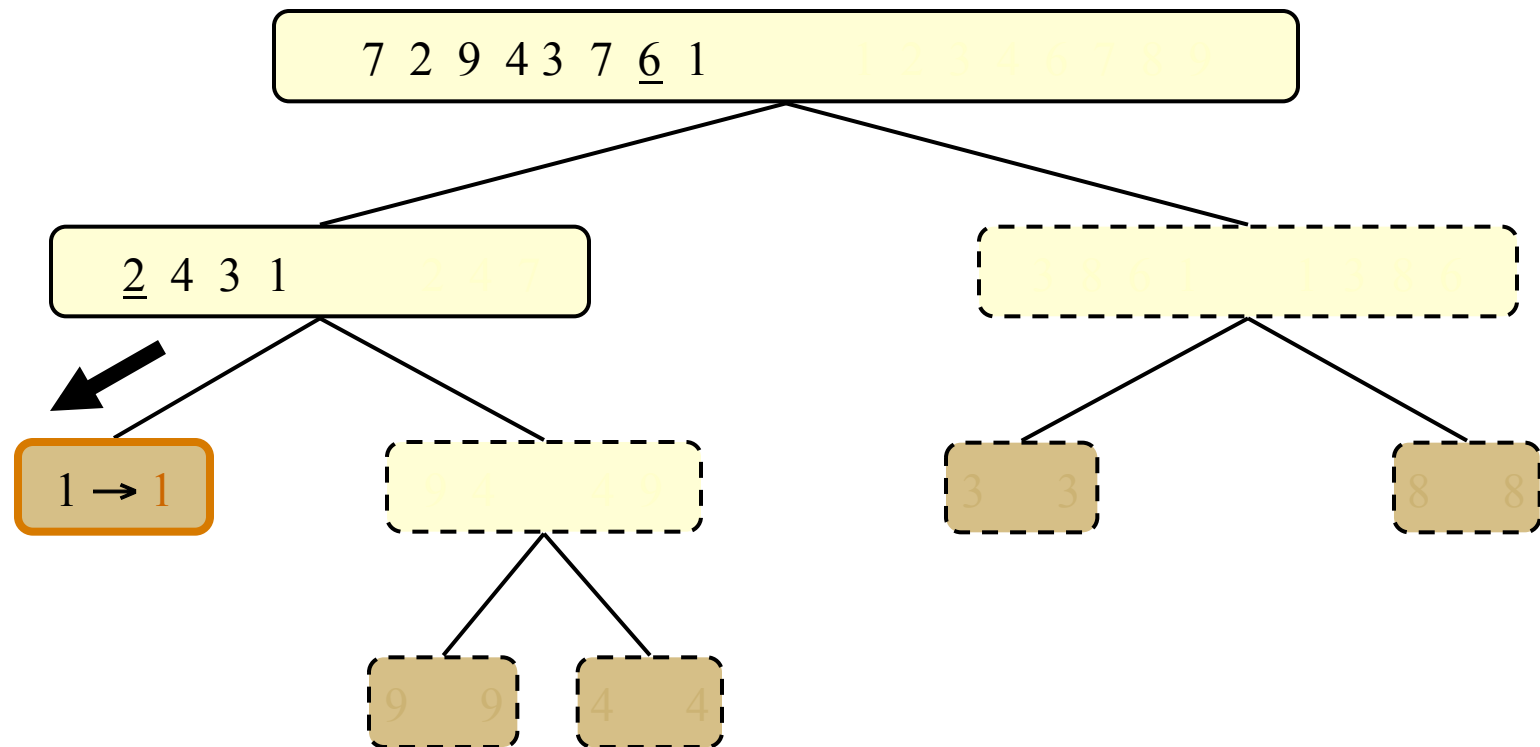
QuickSort: Exemplo de Execução (cont.)

- Partição, chamada recursiva e seleção do pivô



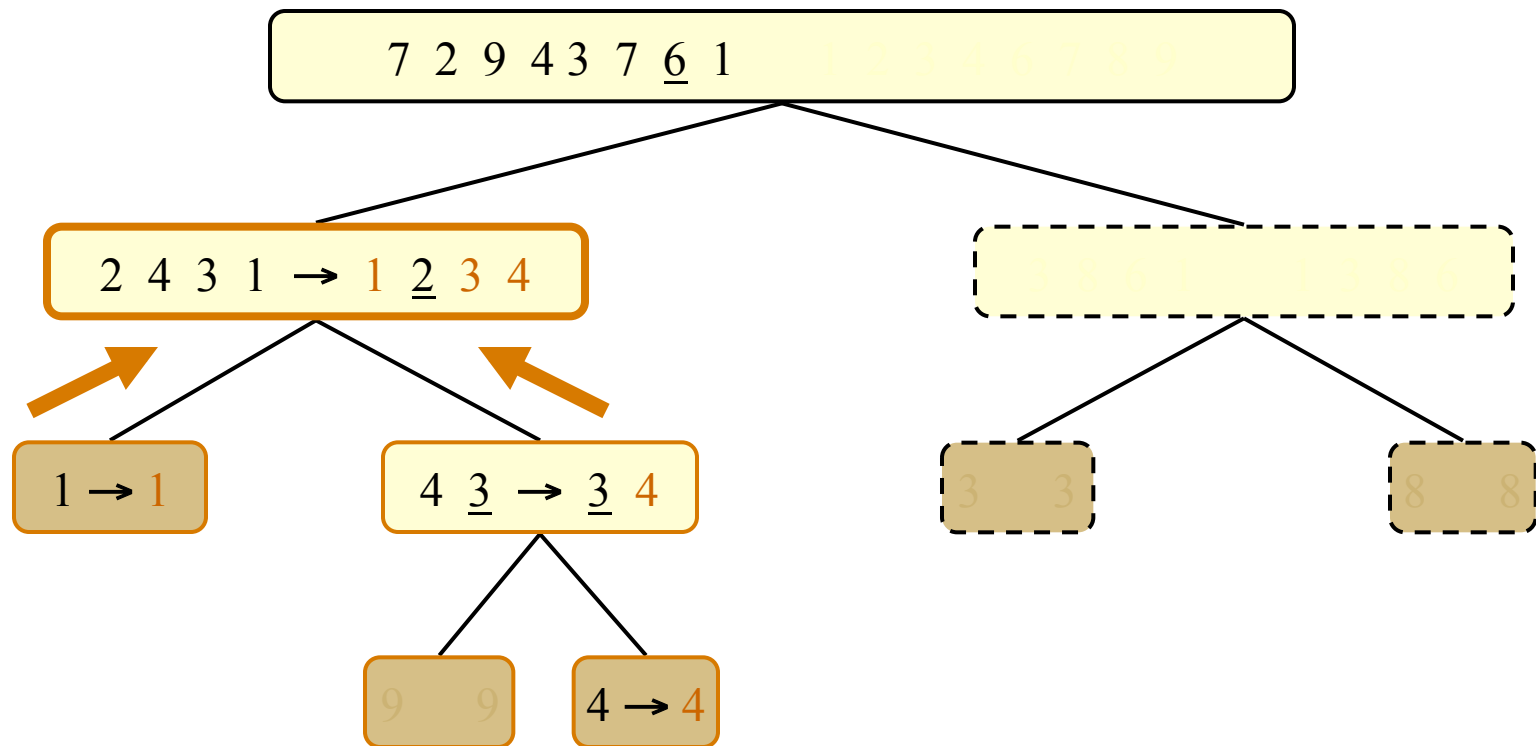
QuickSort: Exemplo de Execução (cont.)

- Partição, chamada recursiva e caso base



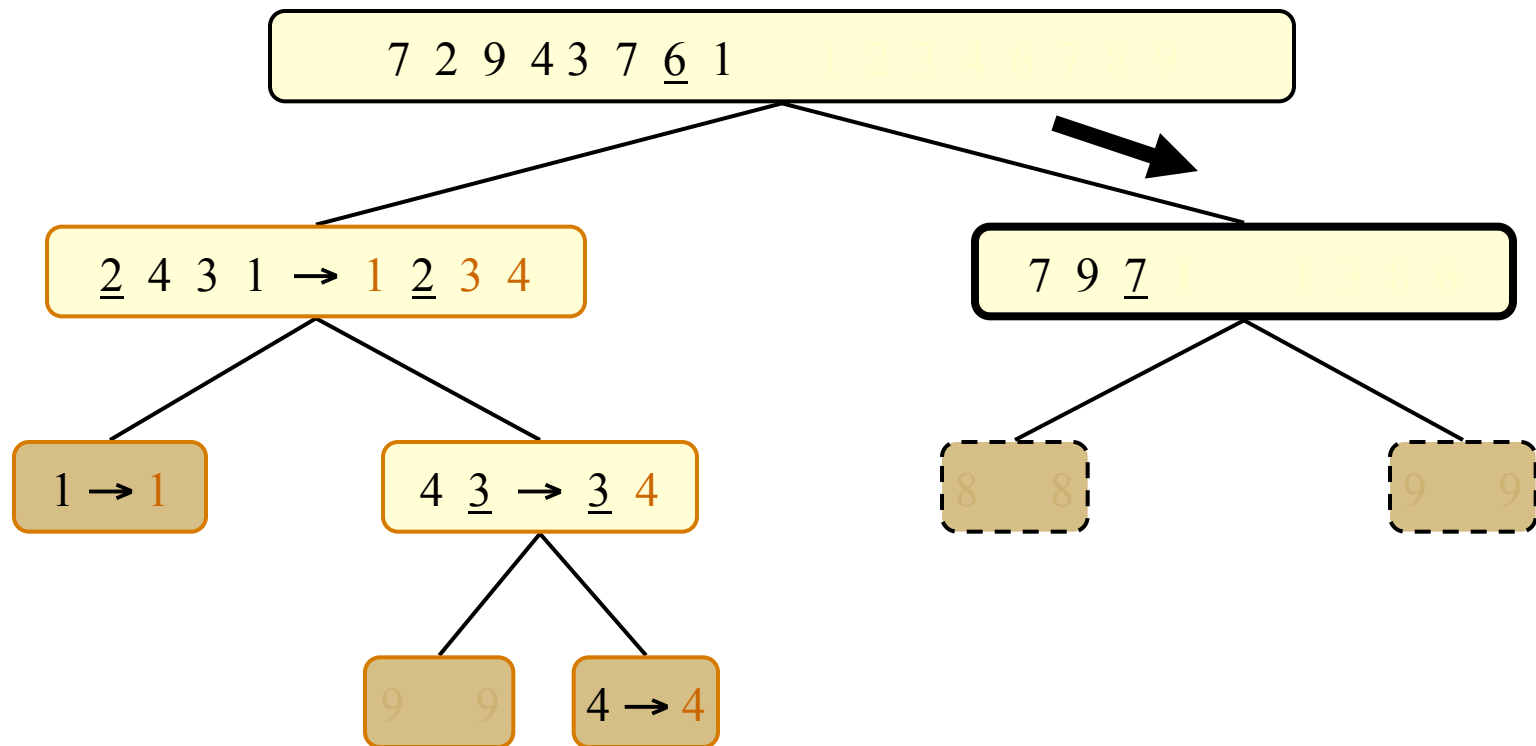
QuickSort: Exemplo de Execução (cont.)

- Chamada recursiva, ..., caso base e união



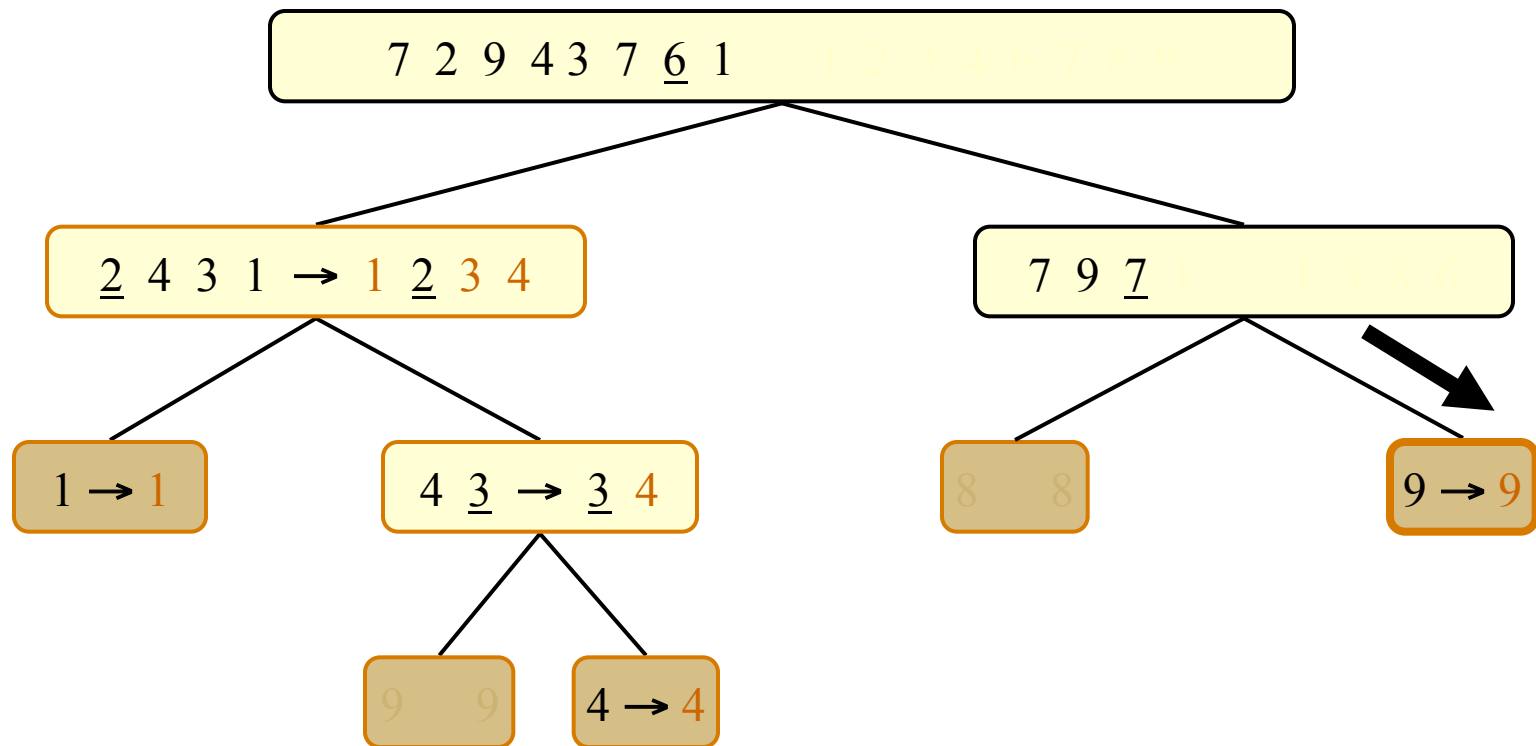
QuickSort: Exemplo de Execução (cont.)

- Chamada recursiva e seleção do pivô



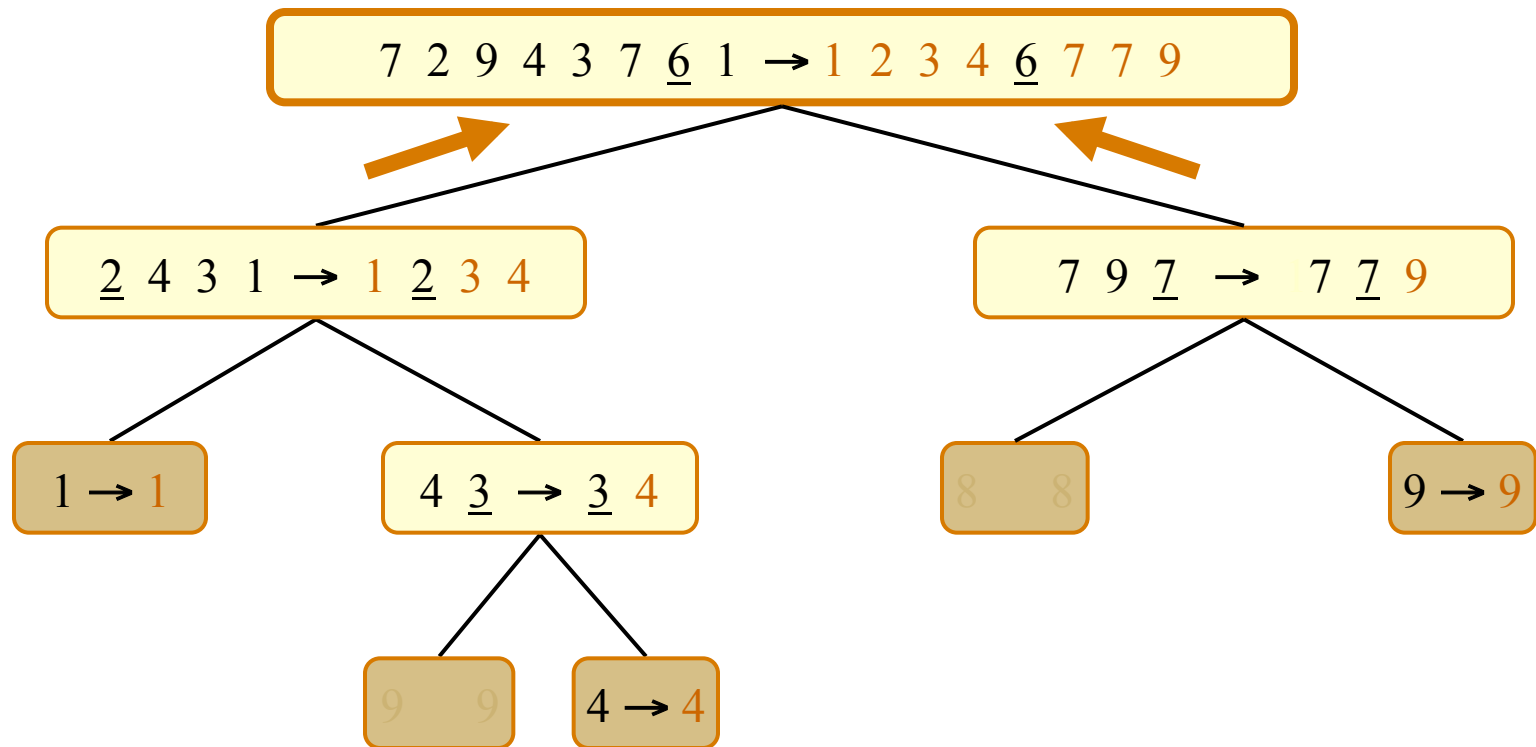
QuickSort: Exemplo de Execução (cont.)

- Partição, ..., chamada recursiva, caso base



QuickSort: Exemplo de Execução (cont.)

- União



QUICKSORT **IMPLEMENTAÇÃO RECURSIVA**

QuickSort - Função

```
void QuickSort(TItem *v, int n) {  
    QuickSort_ordena(v, 0, n-1);  
}
```

```
void QuickSort_ordena(TItem *v, int esq, int dir) {  
    int i, j;  
    QuickSort_particao(v, esq, dir, &i, &j);  
    if (esq < j) QuickSort_ordena(v, esq, j);  
    if (i < dir) QuickSort_ordena(v, i, dir);  
}
```

QuickSort - Partição

```
void QuickSort_particao(TItem *v, int esq, int dir,
                      int *i, int *j) {
    TItem pivo, aux;
    *i = esq; *j = dir;
    pivo = v[(*i + *j)/2]; /* obtem o pivo x */
    do {
        while (pivo.chave > v[*i].chave) (*i)++;
        while (pivo.chave < v[*j].chave) (*j)--;
        if (*i <= *j) {
            aux = v[*i];
            v[*i] = v[*j];
            v[*j] = aux;
            (*i)++; (*j)--;
        }
    } while (*i <= *j);
}
```

QuickSort



- O anel interno da função QuickSort_particao é extremamente simples.
- Razão pela qual o algoritmo QuickSort é tão rápido.

QUICKSORT

ANÁLISE DO ALGORITMO

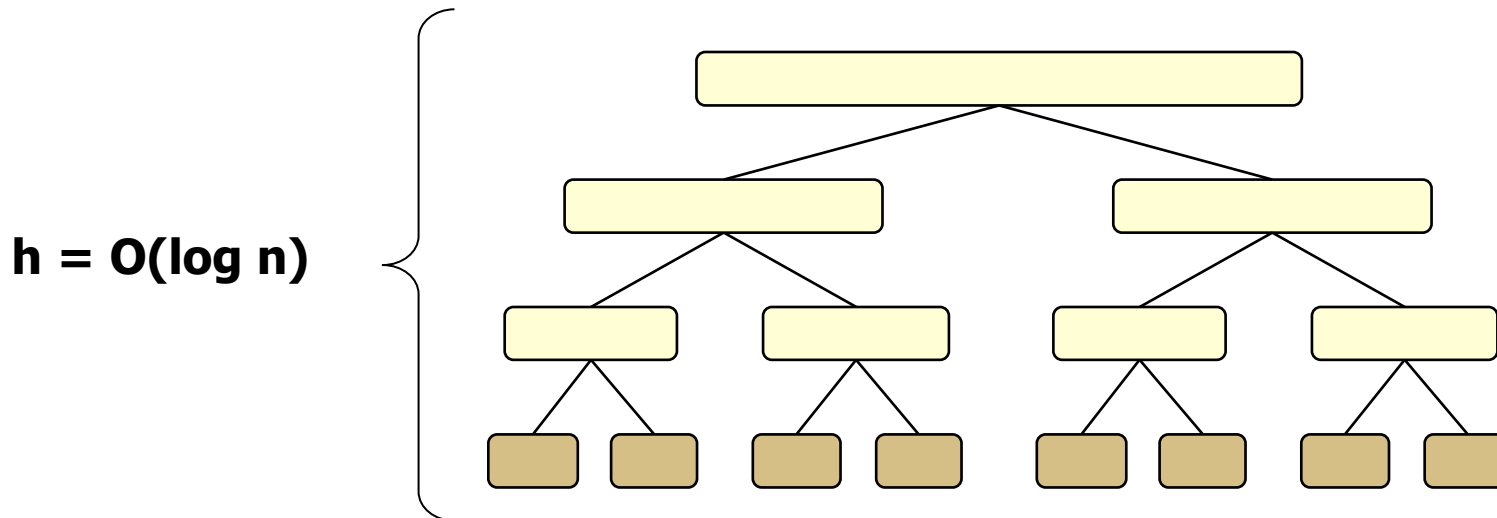
Características:

- Qual o pior caso para o QuickSort?
 - Por que?
 - Qual sua **ordem de complexidade**?
 - Qual o **melhor caso**?
 - O algoritmo é **estável**?

Análise do QuickSort

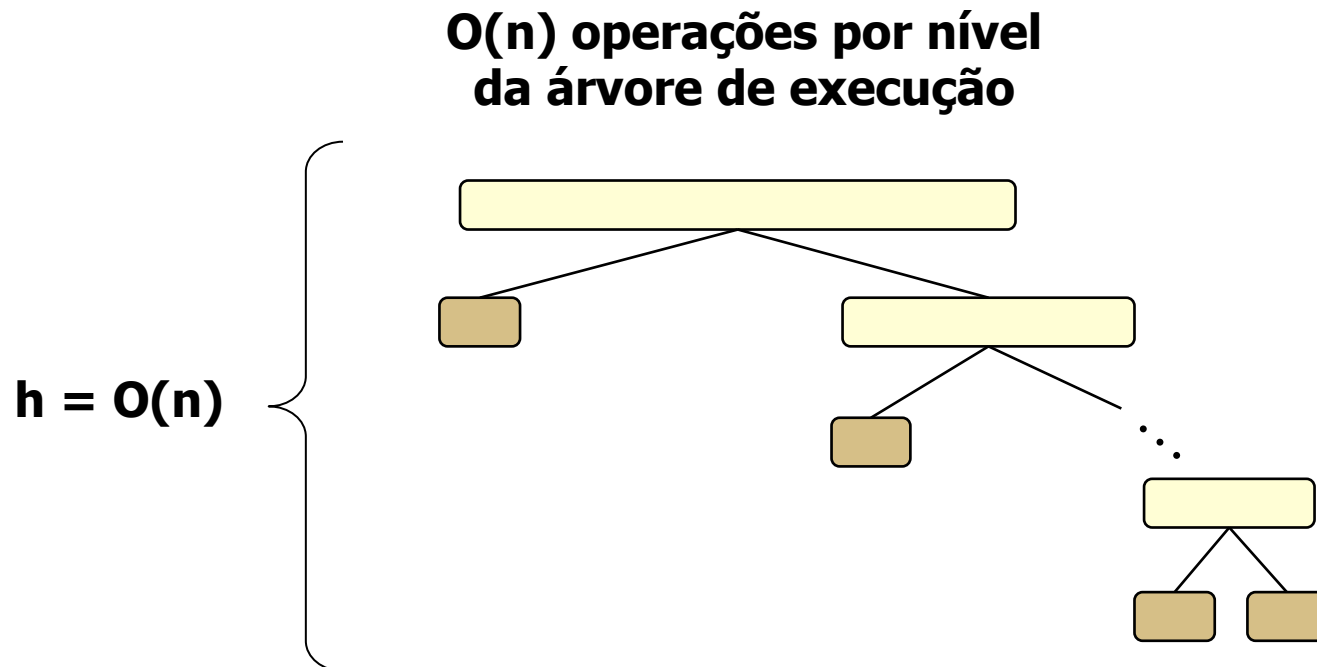
- Melhor caso: $C(n) = 2C(n/2) + n = n \log n$
 - Ocorre quando o problema é sempre dividido em subproblemas de igual tamanho após a partição.

**$O(n)$ operações por nível
da árvore de execução**



Análise do QuickSort

- Pior caso: $C(n) = O(n^2)$
 - O pior caso ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos de um arquivo já ordenado.



Análise do QuickSort

- Pior caso: $C(n) = O(n^2)$
 - O pior caso pode ser evitado empregando pequenas modificações no algoritmo.
 - Para isso basta escolher três itens quaisquer do vetor e usar a mediana dos três como pivô.

Análise do QuickSort

- Caso médio de acordo com Sedgewick e Flajolet (1996, p. 17):
 - $C(n) \approx 1,386n \log n - 0,846n$,
 - Isso significa que em média o tempo de execução do QuickSort é cerca de $O(n \log n)$.

QUICKSORT

VANTAGENS/DESVANTAGENS

- Vantagens:
 - É extremamente eficiente para ordenar arquivos de dados.
 - Necessita de apenas uma pequena pilha como memória auxiliar.
 - Requer $O(n \log n)$ comparações em média (caso médio) para ordenar n itens.

- Desvantagens:
 - Tem um pior caso $O(n^2)$ comparações.
 - Sua implementação é delicada e difícil:
 - Um pequeno engano pode levar a efeitos inesperados para algumas entradas de dados.
 - O método não é estável.

QUICKSORT

IMPLEMENTAÇÃO

NÃO-RECURSIVA

QuickSort Não Recursivo



```
void QuickSort_iter(TItem *v, int n) {  
    TPilha pilha_dir, pilha_esq;  
    int esq, dir, i, j;  
  
    TPilha_Inicia(&pilha_dir);  
    TPilha_Inicia(&pilha_esq);  
    esq = 0;  
    dir = n-1;  
  
    TPilha_Push(&pilha_dir, dir);  
    TPilha_Push(&pilha_esq, esq);  
  
    // ... continua
```

QuickSort Não Recursivo

```
// ... continuação:
```

```
do {  
    if (dir > esq) {  
        QuickSort_particao(v, esq, dir, &i, &j);  
        TPilha_Push(&pilha_dir, j);  
        TPilha_Push(&pilha_esq, esq);  
        esq = i;  
    }  
    else {  
        TPilha_Pop(&pilha_dir, &dir);  
        TPilha_Pop(&pilha_esq, &esq);  
    }  
} while (!TPilha_EhVazia(&pilha_dir));  
}
```

Pilha de Recursão x Pilha no Explícita



- O que é colocado em cada uma das pilhas?
- Que intervalo do vetor é empilhado em cada caso?

- Vantagens da versão não-recursiva:
 - É extremamente eficiente para ordenar arquivos de dados.
 - Necessita de apenas uma pequena pilha como memória auxiliar.
- Desvantagens da versão não-recursiva:
 - Sua implementação é delicada e difícil:

QUICKSORT

MELHORIAS

QuickSort – Melhorias no Algoritmo

- Pivô – Mediana de três
 - Não empilhar quando tem apenas um item
- Melhor ainda: usar algoritmo de inserção (InsertSort) para vetores pequenos (menos de 10 ou 20 elementos)
- Escolha correta do lado a ser empilhado
- Resultado:
 - **Melhoria no tempo de execução de 25% a 30%**



Perguntas?

QUICKSORT
EXERCÍCIO

Exercício

- Dada a sequência de números:

3 4 9 2 5 1 8 6 3

Ordene em ordem crescente utilizando o algoritmo de ordenação **QuickSort**, apresentando a sequência dos números a cada passo.