Algoritmo e Estrutura de Dados II
CTCO02

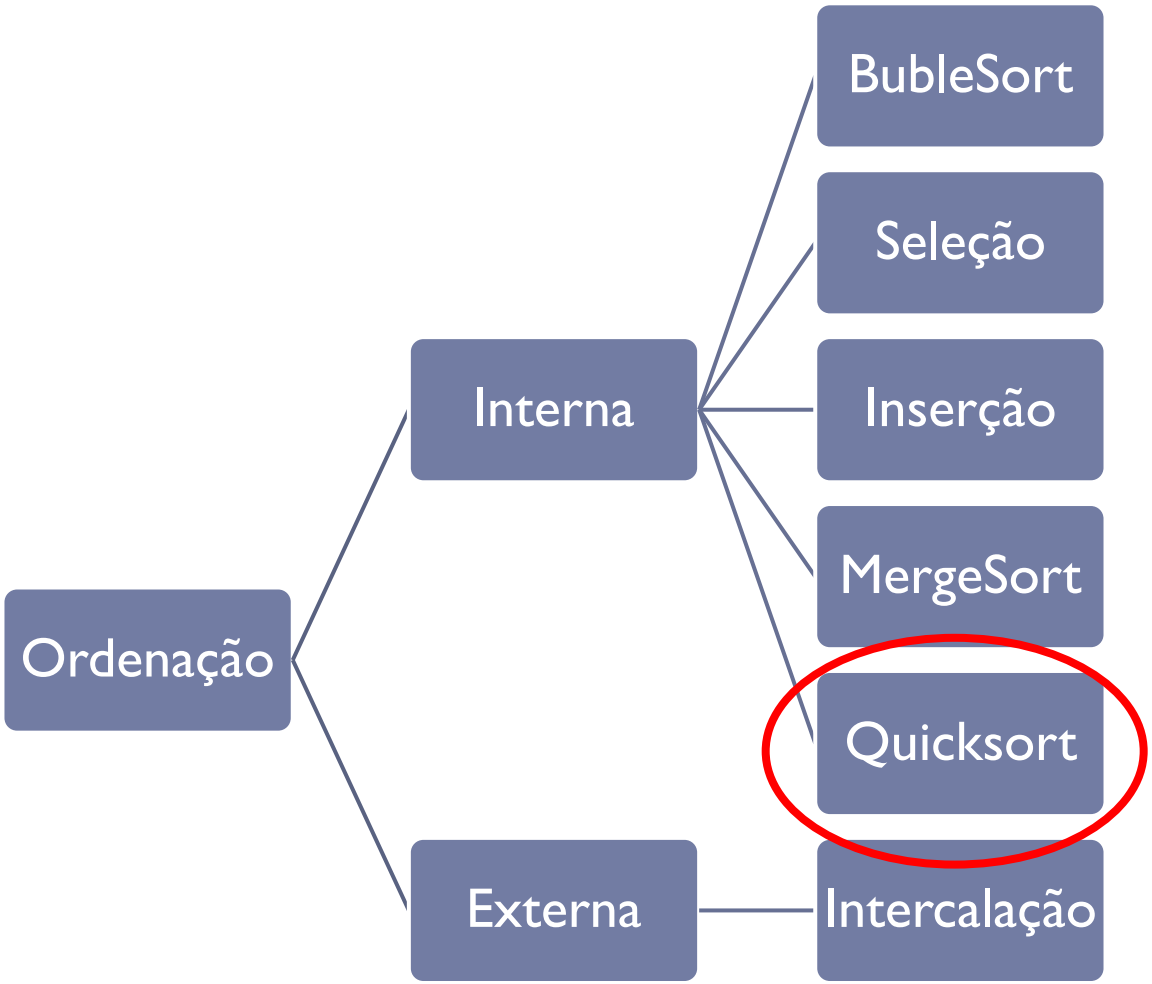
Ordenação
QuickSort

Vanessa Souza

Universidade Federal de Itajubá – Instituto de Matemática e Computação

Ordenação

Classificação dos Métodos de Ordenação



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.
- Também utiliza a estratégia : Dividir para Conquistar

QuickSort

- O algoritmo baseia-se em escolher um elemento (pivô) e dividir o vetor desordenado em duas partes: a parte da esquerda, com elementos menores do que o pivô, e a parte da direita, com elementos maiores do que o pivô.

- Ao final de cada passada do algoritmo, o pivô estará na sua posição no vetor (ordenado).

- O problema se reduz então em ordenar os elementos à esquerda e à direita do pivô.

QuickSort

- Estratégia do algoritmo
  - Escolhe um pivô
  - Particiona o vetor com base no pivô
    - Menores a esquerda
    - Maiores a direita
  - A posição final do pivô no vetor será base para uma nova partição

QuickSort

Algorithm 1 QuickSort

```
procedure QUICKSORT(V, inicio, fim)                ▷ inicio e fim sao indices do vetor
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure

procedure PARTICIONA(V, inicio, fim)
    pivo ← V[inicio]
    pos ← inicio                                   ▷ guarda a posicao final do pivo no vetor
    for (i = inicio + 1; i <= fim; i++) do
        if (V[i] < pivo) then
            pos ← pos + 1
            if (i ≠ pos) then
                troca V[i] com V[pos]
            end if
        end if
    end for
    troca V[inicio] com V[pos]
    Retorna pos
end procedure
```

QuickSort

- QuickSort(vet, 0, 4)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

0	1	2	3	4
3	5	4	1	2

QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)

0	1	2	3	4
3	5	4	1	2

Particiona(vet, 0, 4)

Objetivo: Colocar o pivô (vet[inicio]) na posição correta dele no vetor.
Qual é essa posição?

QuickSort

- QuickSort(vet, 0, 4)

Particiona(vet, 0, 4)

Pivô = 3
pos = 0
aux = 1

0	1	2	3	4
3	5	4	1	2

Pivô = 3
pos = 0

A variável pos guarda a posição em que o pivô deverá ficar.

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 0
aux = 1

0	1	2	3	4
3	5	4	1	2

Compara o pivô com os demais elementos do vetor.
Se o elemento for maior que o pivô **não faz nada**.

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 0
aux = 2

0	1	2	3	4
3	5	4	1	2

Compara o pivô com os demais elementos do vetor.
Se o elemento for maior que o pivô **não faz nada**.

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 0
aux = 3

Compara o pivô com os demais elementos do vetor.
Se o elemento for menor que o pivô:
1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 1
aux = 3

0	1	2	3	4
3	5	4	1	2

Compara o pivô com os demais elementos do vetor.
Se o elemento for menor que o pivô:

1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 1
aux = 3

Compara o pivô com os demais elementos do vetor.
Se o elemento for menor que o pivô:
1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Semanticamente, pos ≠ aux significa que já houve um elemento maior que o pivô antes de encontrar um menor que ele. Por isso realiza a troca dos elementos, de forma a deixar os elementos menores que o pivô à esquerda dele e os maiores à direita.

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 1
aux = 4

0	1	2	3	4
3	1	4	5	2

Compara o pivô com os demais elementos do vetor.
Se o elemento for menor que o pivô:

1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 2
aux = 4

0	1	2	3	4
3	1	4	5	2

Compara o pivô com os demais elementos do vetor.
Se o elemento for menor que o pivô:

1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 2
aux = 4

Compara o pivô com os demais elementos do vetor.
Se o elemento for menor que o pivô:

1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 2
aux = 5

Compara o pivô com os demais elementos do vetor.
Quando chega ao final do vetor, troca
vet[pos] com vet[inicio]

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]

Particiona(vet, 0, 4)

Pivô = 3
pos = 2
aux = 5

0	1	2	3	4
2	1	3	5	4

Compara o pivô com os demais elementos do vetor.
Quando chega ao final do vetor, troca
vet[pos] com vet[inicio]

Semanticamente, é essa troca que coloca o pivô no local correto dele.

Particiona(vet, 0, 4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]
  - PosPivo = 2

Particiona(vet, 0, 4)

Pivô = 3
pos = 2
aux = 5

0	1	2	3	4
2	1	3	5	4

Após a troca, a função Particiona finaliza, retornando o valor de pos.

~~*Particiona(vet, 0, 4)*~~
QuickSort(vet, 0, 4)

QuickSort

2	1	3	5	4

- ✓ O princípio de ordenação do Quick é ordenar o pivô!
- ✓ O valor 3 já está ordenado.
- ✓ Necessário ordenar à esquerda do pivô e à direita do pivô.

O problema se reduz a ordenar os outros vetores

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Como?

QuickSort

- QuickSort(vet, 0, 4)
  - Pivô = vet[inicio]
  - PosPivo = 2

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 0
fim = 4
posPivo = 2

0	1	2	3	4
2	1	3	5	4

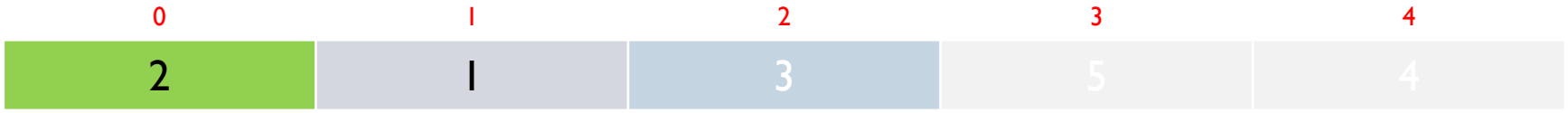
Nova chamada QuickSort

~~*Particiona(vet, 0, 4)*~~
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 1)
  - Pivô = vet[inicio]

Inicio = 0
fim = 1
posPivo = ?



QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

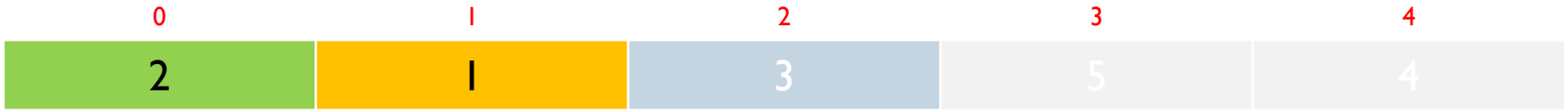
QuickSort

- QuickSort(vet, 0, 1)
  - Pivô = vet[inicio]

Particiona(vet, 0, 1)

Pivô = 2
pos = 1
aux = 1

Inicio = 0
fim = 1
posPivo = ?



Compara o pivô com os demais elementos do vetor.
Se o elemento for menor que o pivô:

1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Particiona(vet, 0, 1)
QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

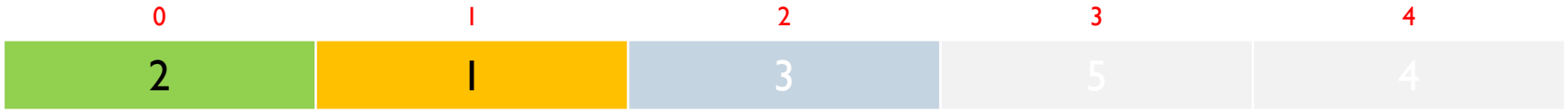
QuickSort

- QuickSort(vet, 0, 1)
  - Pivô = vet[inicio]

Particiona(vet, 0, 1)

Pivô = 2
pos = 1
aux = 1

Inicio = 0
fim = 1
posPivo = ?



Compara o pivô com os demais elementos do vetor.
Quando chega ao final do vetor, troca
vet[pos] com vet[inicio]

Particiona(vet, 0, 1)
QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

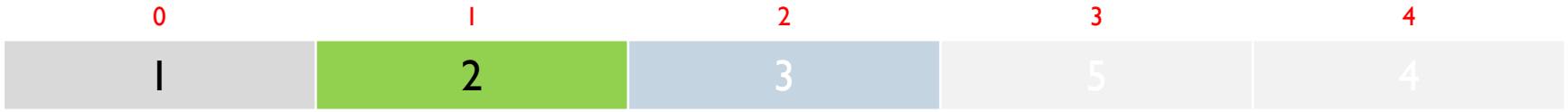
QuickSort

- QuickSort(vet, 0, 1)
  - Pivô = vet[inicio]

Particiona(vet, 0, 1)

Pivô = 2
pos = 1
aux = 2

Inicio = 0
fim = 1
posPivo = ?



Compara o pivô com os demais elementos do vetor.
Quando chega ao final do vetor, troca
vet[pos] com vet[inicio]

Particiona(vet, 0, 1)
QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

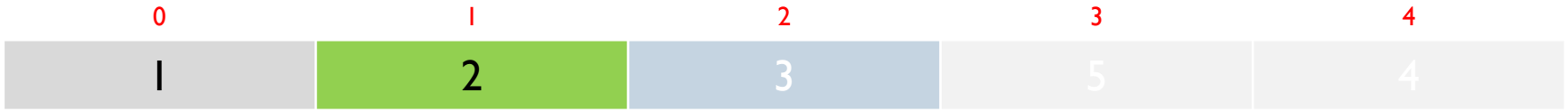
QuickSort

- QuickSort(vet, 0, 1)
  - Pivô = vet[inicio]
  - posPivo = 1

Particiona(vet, 0, 1)

Pivô = 2
pos = 1
aux = 2

Inicio = 0
fim = 1
posPivo = 1



Após a troca, a função Particiona finaliza, retornando o valor de pos.

~~*Particiona(vet, 0, 1)*~~
QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 1)
  - Pivô = vet[inicio]
  - posPivo = 1

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 0
fim = 1
posPivo = 1

0	1	2	3	4
1	2	3	5	4

Nova chamada QuickSort

~~*Particiona(vet, 0, 1)*~~
QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 0)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 0
fim = 0
posPivo =

0	1	2	3	4
1	2	3	5	4

Semanticamente, a posição 0 já pode ser considerada ordenada, porque é um vetor unitário

QuickSort(vet, 0,0)
QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 1)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 0
fim = 1
posPivo = 1

0	1	2	3	4
1	2	3	5	4

Nova chamada QuickSort

~~*QuickSort(vet, 0,0)*~~
QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 2, 1)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 0
fim = 1
posPivo = 1

0	1	2	3	4
1	2	3	5	4

Nova chamada QuickSort

Semanticamente significa que não há mais elementos no vetor para ordenar

QuickSort(vet, 2, 1)
QuickSort(vet, 0, 1)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 2, 1)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 0
fim = 1
posPivo = 1

0	1	2	3	4
1	2	3	5	4

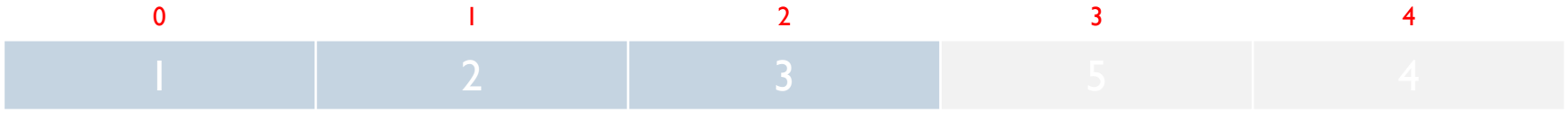
~~*QuickSort(vet, 2, 1)*~~
~~*QuickSort(vet, 0, 1)*~~
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 0
fim = 4
posPivo = 2



Nova chamada QuickSort

QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

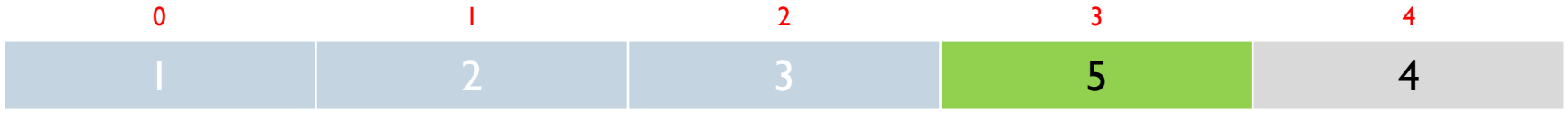
QuickSort

- QuickSort(vet, 3, 4)
  - pivô = vet[inicio]

Particiona(vet, 3, 4)

Pivô = 3
pos = 3
aux = 4

Inicio = 3
fim = 4
posPivo = ?



Compara o pivô com os demais elementos do vetor

Particiona(vet, 3, 4)
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

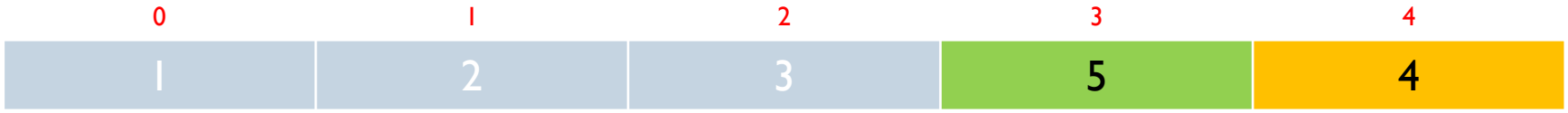
QuickSort

- QuickSort(vet, 3, 4)
  - pivô = vet[inicio]

Particiona(vet, 3, 4)

Pivô = 3
pos = 3
aux = 4

Inicio = 3
fim = 4
posPivo = ?



Compara o pivô com os demais elementos do vetor
Se o elemento for menor que o pivô:
1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Particiona(vet, 3, 4)
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

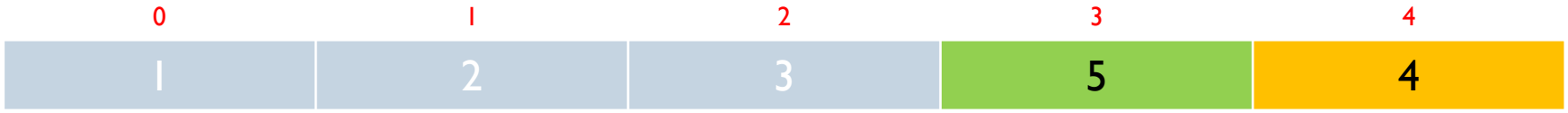
QuickSort

- QuickSort(vet, 3, 4)
  - pivô = vet[inicio]

Particiona(vet, 3, 4)

Pivô = 3
pos = 4
aux = 4

Inicio = 3
fim = 4
posPivo = ?



Compara o pivô com os demais elementos do vetor
Se o elemento for menor que o pivô:
1. Incrementa pos → pos++
2. Se pos ≠ aux, troca os elementos

Particiona(vet, 3, 4)
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

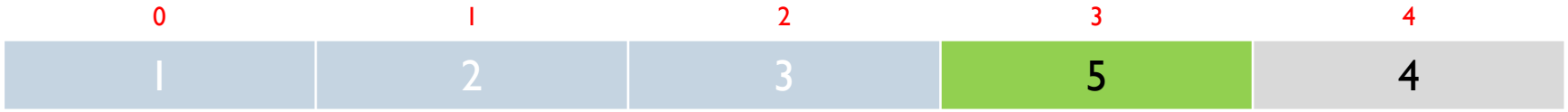
QuickSort

- QuickSort(vet, 3, 4)
  - pivô = vet[inicio]

Particiona(vet, 3, 4)

Pivô = 3
pos = 4
aux = 5

Inicio = 3
fim = 4
posPivo = ?



Compara o pivô com os demais elementos do vetor
Quando chega ao final do vetor, troca
vet[pos] com vet[inicio]

Particiona(vet, 3, 4)
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

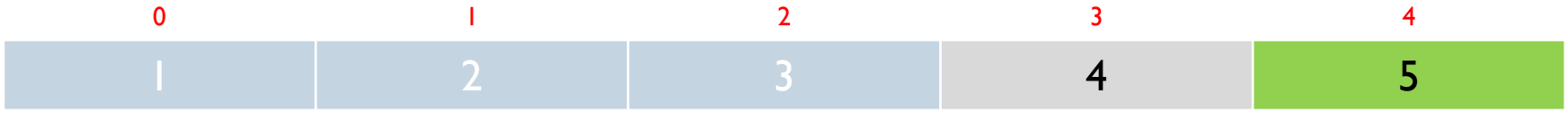
QuickSort

- QuickSort(vet, 3, 4)
  - pivô = vet[inicio]

Particiona(vet, 3, 4)

Pivô = 3
pos = 4
aux = 5

Inicio = 3
fim = 4
posPivo = ?



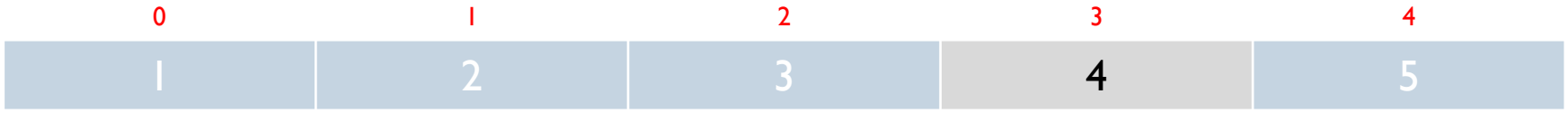
Compara o pivô com os demais elementos do vetor
Quando chega ao final do vetor, troca
vet[pos] com vet[inicio]

Particiona(vet, 3, 4)
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

QuickSort

Inicio = 3
fim = 4
posPivo = 4

- QuickSort(vet, 3, 4)
  - pivô = vet[inicio]
  - posPivo = 4



Após a troca, a função Particiona finaliza, retornando o valor de pos.

~~*Particiona(vet, 3, 4)*~~
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 3, 4)
  - pivô = vet[inicio]
  - posPivo = 4

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 3
fim = 4
posPivo = 4

0	1	2	3	4
1	2	3	4	5

Nova chamada QuickSort

QuickSort(vet, 3,3)
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 3, 3)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 3
fim = 4
posPivo = 4

0	1	2	3	4
1	2	3	4	5

Nova chamada QuickSort

~~*QuickSort(vet, 3,3)*~~
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 3, 3)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 3
fim = 4
posPivo = 4

0	1	2	3	4
1	2	3	4	5

Semanticamente, a posição 3 já pode ser considerada ordenada, porque é um vetor unitário

~~*QuickSort(vet, 3,3)*~~
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 5, 4)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 3
fim = 4
posPivo = 4

0	1	2	3	4
1	2	3	4	5

Nova chamada QuickSort

Semanticamente significa que não há mais elementos no vetor para ordenar

QuickSort(vet, 5,4)
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 5, 4)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 3
fim = 4
posPivo = 4

0	1	2	3	4
1	2	3	4	5

Nova chamada QuickSort

Semanticamente significa que não há mais elementos no vetor para ordenar

~~*QuickSort(vet, 5,4)*~~
QuickSort(vet, 3,4)
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 3, 4)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 3
fim = 4
posPivo = 4

0	1	2	3	4
1	2	3	4	5

Fim da Chamada vet, 3, 4

~~*QuickSort(vet, 3,4)*~~
QuickSort(vet, 0, 4)

QuickSort

- QuickSort(vet, 0, 4)

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

Inicio = 0
fim = 4
posPivo = 2

0	1	2	3	4
1	2	3	4	5

Fim da Chamada vet, 0, 4

~~*QuickSort(vet, 0, 4)*~~

QuickSort

```
procedure QUICKSORT(V, inicio, fim)
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure
```

0	1	2	3	4
1	2	3	4	5

Pilha vazia!
Vetor Ordenado!

QuickSort

Algorithm 1 QuickSort

```
procedure QUICKSORT(V, inicio, fim)                ▷ inicio e fim sao indices do vetor
    if inicio < fim then
        pivo ← Particiona(vet, inicio, fim)
        QuickSort(V, inicio, pivo-1)
        QuickSort(V, pivo+1, fim)
    end if
end procedure

procedure PARTICIONA(V, inicio, fim)
    pivo ← V[inicio]
    pos ← inicio                                   ▷ guarda a posicao final do pivo no vetor
    for (i = inicio + 1; i <= fim; i++) do
        if (V[i] < pivo) then
            pos ← pos + 1
            if (i ≠ pos) then
                troca V[i] com V[pos]
            end if
        end if
    end for
    troca V[inicio] com V[pos]
    Retorna pos
end procedure
```

QuickSort

- Exercício
  - Fazer o teste de mesa com o seguinte vetor:

3	7	1	18	8	6

Quicksort

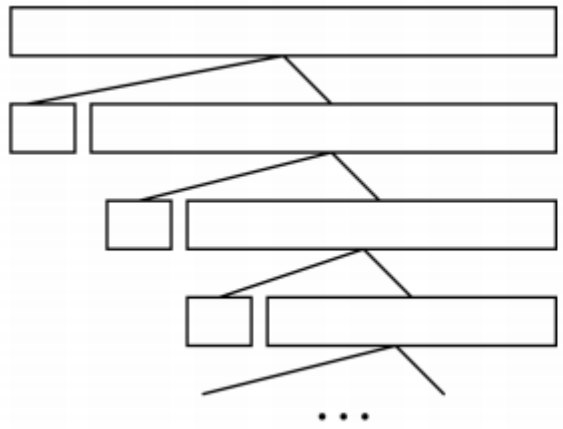
- A eficiência do algoritmo QuickSort depende do algoritmo de partição.

- Na melhor situação, cada passo de particionamento divide um problema de tamanho n em dois problemas de tamanho (aproximadamente) n/2.
  - Neste caso teremos $\log_2(n)+1$ níveis na árvore de recursão. Portanto, o tempo de execução do algoritmo QuickSort é $O(n \log_2 n)$.

Note, no entanto, que este algoritmo é melhor do que o *MergeSort*, pois não requer espaço adicional.

Quicksort

- Mas, o desempenho do algoritmo QuickSort depende da escolha do pivô.
- Imagine, por exemplo, no pior caso, que o pivô escolhido é sempre o menor elemento.
- Neste caso, a árvore de recursão terá a seguinte forma:



Neste caso, em vez de $\log_2 n$ níveis, vão existir n níveis na árvore de recursão e, portanto, a complexidade do algoritmo será $O(n^2)$.

QuickSort

1	2	3	4	5

Particiona o vetor
Início = 0
Fim = 4

- Solução???

Quicksort

- Boas estratégias são:
  - Escolher o pivô aleatoriamente
  - Escolher o elemento na posição central do vetor.

- Isto evita que, no caso do vetor já estar ordenado (ou quase ordenado) a árvore de recursão seja como a do pior caso.

- Na média, o algoritmo QuickSort tem bom desempenho, por isso é tão utilizado.

Comparação de Complexidade

Número de Comparações

- Na maioria dos métodos de ordenação, o fator relevante que determina seu tempo de execução é o número de comparações realizadas.

Algoritmo	Complexidade Assintótica
Bolha	$O(n^2)$
Bolha Inteligente	$O(n^2)$
Seleção	$O(n^2)$
Inserção	$O(n^2)$
MergeSort	$O(n \log_2 n)$
QuickSort	$O(n \log_2 n)$

Tempo de Execução

- Calculados para um vetor de tamanho 10.000 com elementos distribuídos aleatoriamente.

Algoritmo	Tempo (segundos)
Bolha	625
Bolha Inteligente	453
Seleção	188
Inserção	125
MergeSort	< 1
QuickSort	< 1

Tempo de Execução

- Calculados para um vetor de tamanho 10.000 com elementos **ordenados em ordem crescente**.

Algoritmo	Tempo (segundos)
Bolha	375
Bolha Inteligente	188
Seleção	172
Inserção	< 1
MergeSort	<1
QuickSort	1250

Tempo de Execução

- Calculados para um vetor de tamanho 10.000 com elementos **ordenados em ordem decrescente**.

Algoritmo	Tempo (segundos)
Bolha	625
Bolha Inteligente	422
Seleção	235
Inserção	234
MergeSort	<1
QuickSort	1560

Tempo de Execução

- Comparação para as diferentes entradas

*Tempo em Segundos

Algoritmo	Aleatório	Ordem Crescente	Ordem Decrescente
Bolha	625	375	625
Bolha Inteligente	453	188	422
Seleção	188	172	235
Inserção	125	< 1	234
MergeSort	< 1	<1	<1
QuickSort	< 1	1250	1560

Comparação entre os algoritmos

- https://youtu.be/ZZuD6iUe3Pc?feature=shared

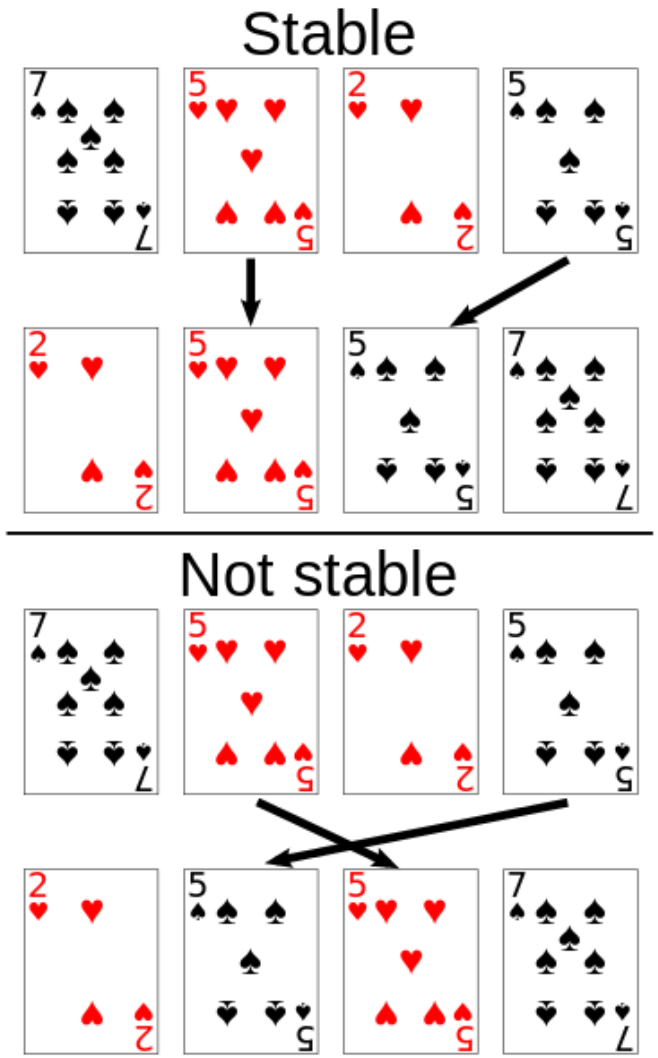
Estabilidade

Estabilidade x Instabilidade

- Um método de ordenação é denominado estável se a ordem relativa dos elementos que exibam a mesma chave permanecer inalterada ao longo de todo o processo de ordenação; caso contrário, ele é denominado instável.

- Em geral, a estabilidade da ordenação é desejável, especialmente quando os elementos já estiverem ordenados em relação a uma ou mais chaves secundárias.

Estabilidade x Instabilidade



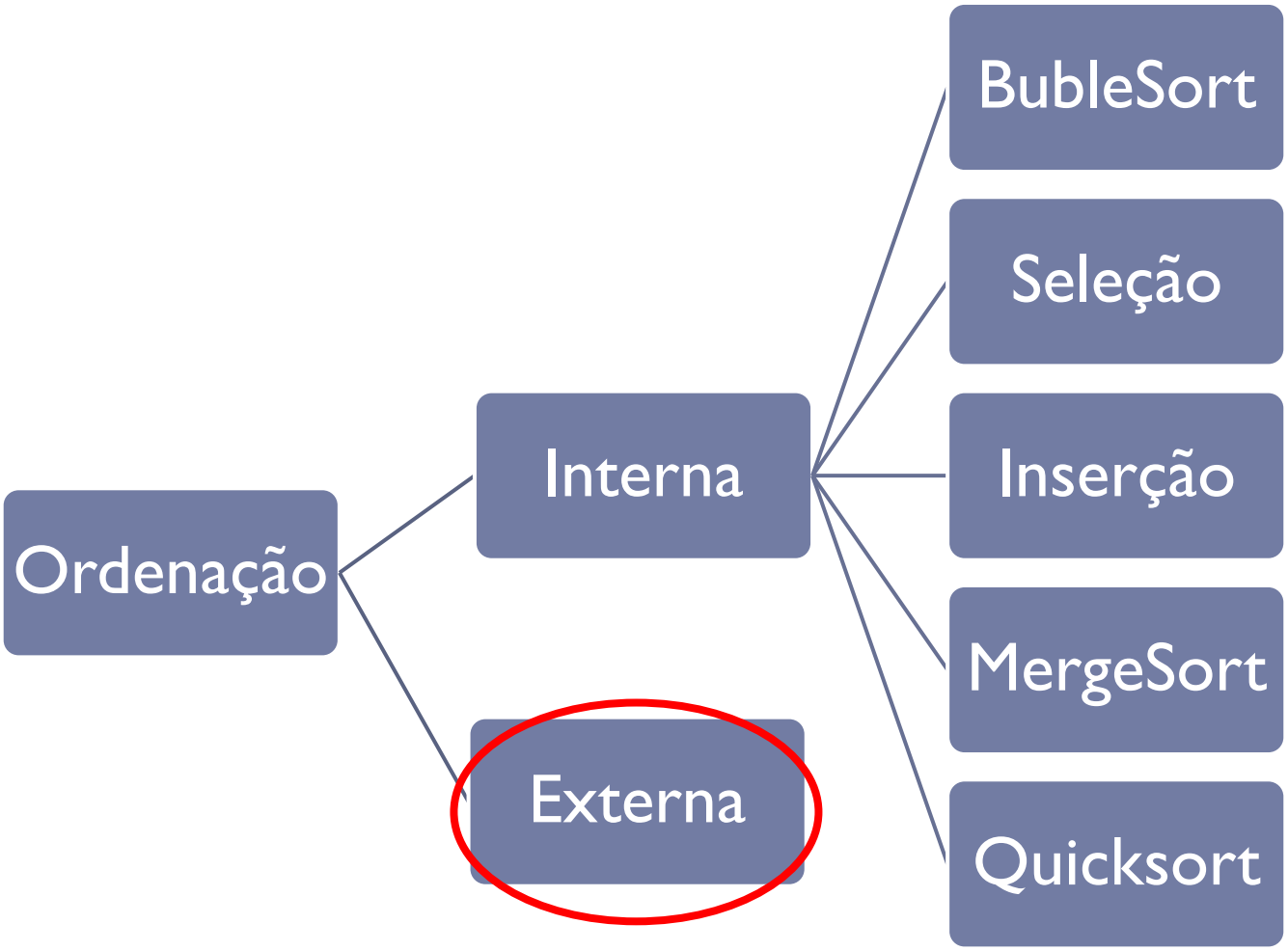
Estabilidade x Instabilidade

- 5 3 2 2 4 1

Algoritmo	Complexidade Assintótica	Estabilidade
Bolha	$O(n^2)$	SIM
Bolha Inteligente	$O(n^2)$	SIM
Seleção	$O(n^2)$	NÃO
Inserção	$O(n^2)$	SIM
MergeSort	$O(n \log_2 n)$	NÃO
QuickSort	$O(n \log_2 n)$	NÃO

Ordenação Externa

Classificação dos Métodos de Ordenação



Ordenação Externa

- A ordenação externa envolve arquivos compostos por um número de registros que é maior do que a memória interna.

- Utiliza métodos de ordenação muito diferentes da ordenação interna.
  - As estruturas de dados devem levar em conta o fato de que os dados estão armazenados em unidades de memória externa, relativamente muito mais lentas do que a memória principal.

Ordenação Externa

- Alto custo computacional.

- O custo principal da ordenação externa está relacionado com o custo de transferir dados entre a memória interna e a memória externa.
  - Minimizar o número de vezes que cada item é transferido da memória interna para a externa.
  - Otimizar entrada/saída/processamento de dados
  - Tecnologia utilizada
    - Fita
    - HD
    - ...

Ordenação Externa

- Estratégia Básica:
  - Quebrar o arquivo em blocos do tamanho da memória interna disponível.
  - Ordenar o bloco menor
  - Intercalar blocos ordenados

Ordenação Externa

- MergeSort Externo
- QuickSort Externo

QuickSort

- Exercício
  - Fazer o teste de mesa com o seguinte vetor:

22	33	55	77	99	11	44	66	88