Algoritmo e Estrutura de Dados II
COM-112

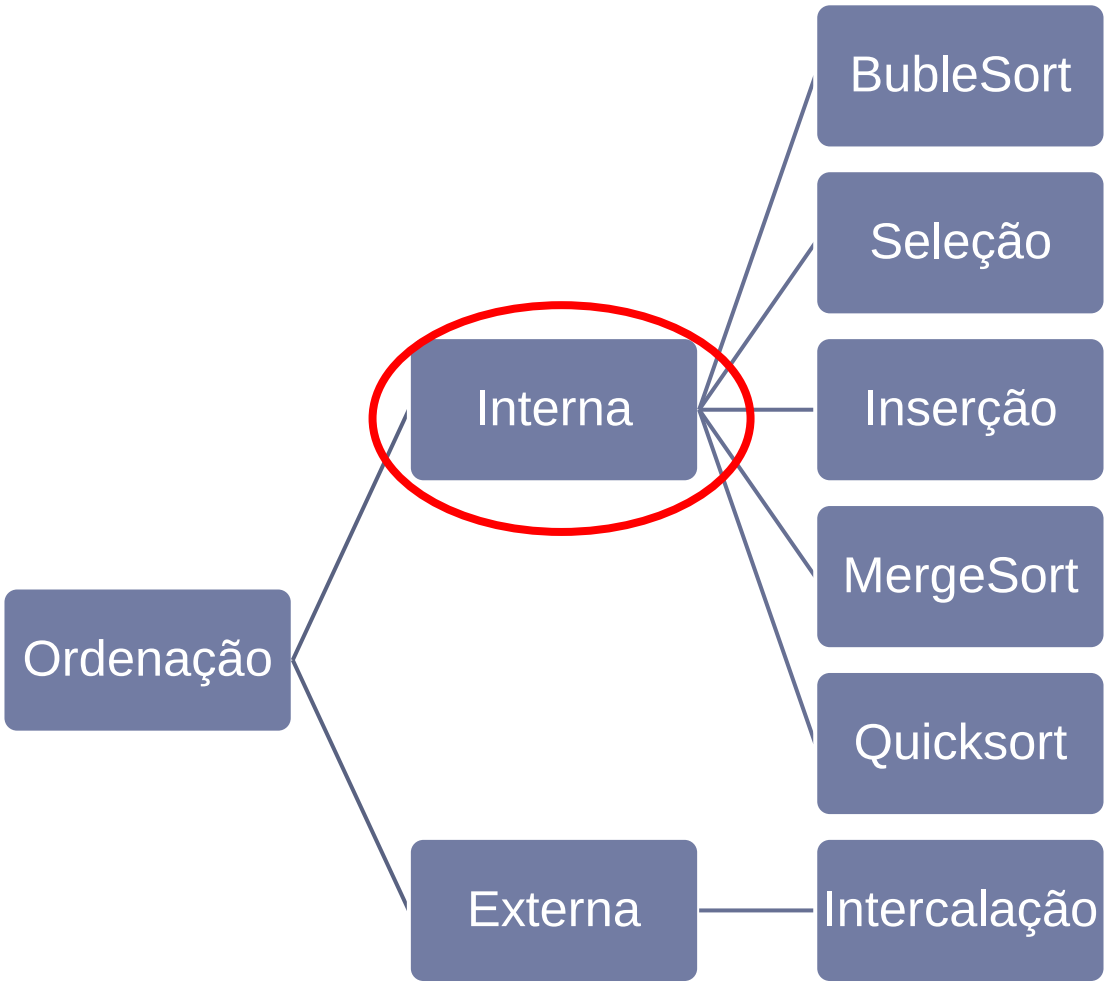
Aula 5

Vanessa Souza

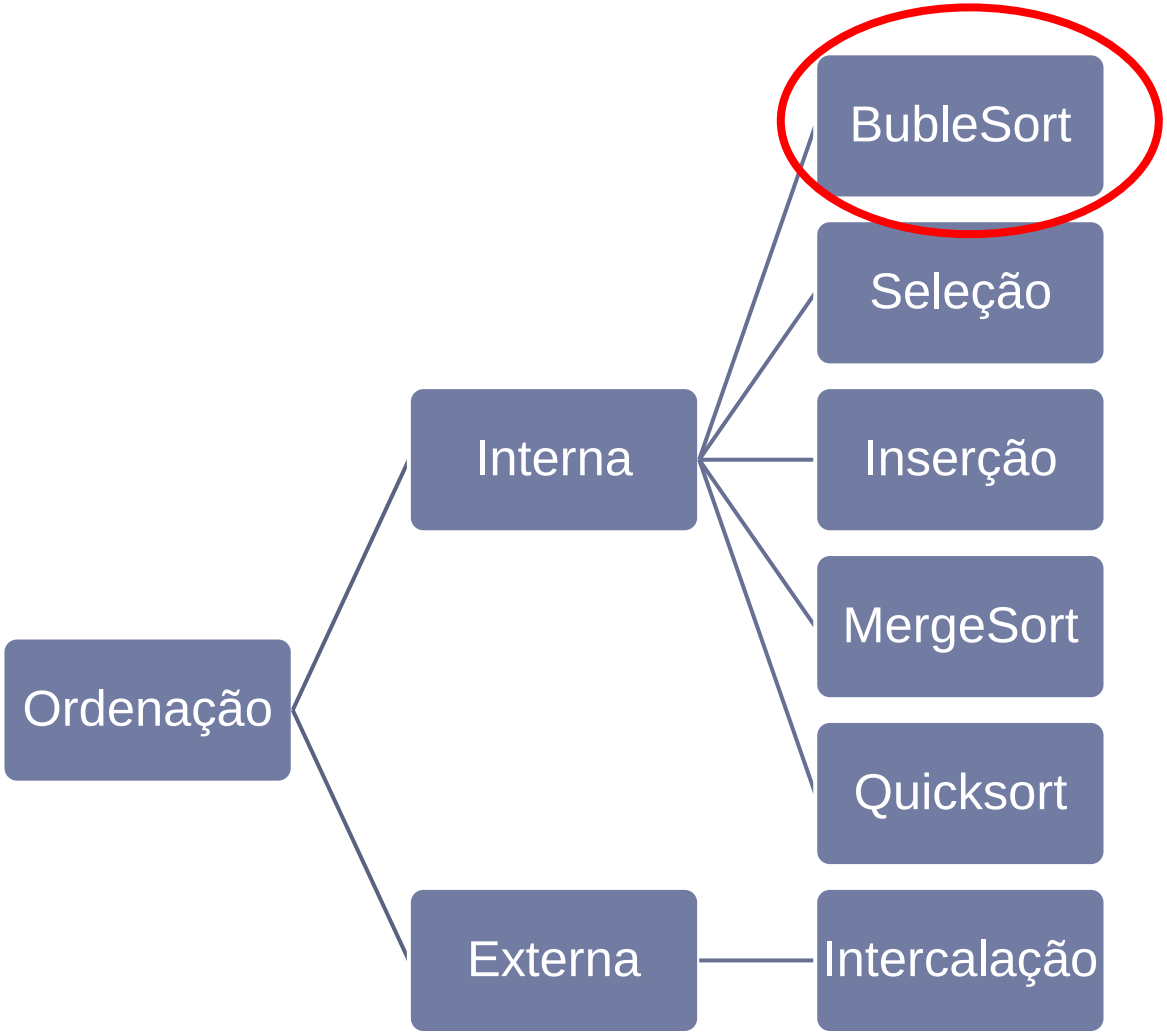
Universidade Federal de Itajubá – Instituto de Matemática e Computação

Ordenação

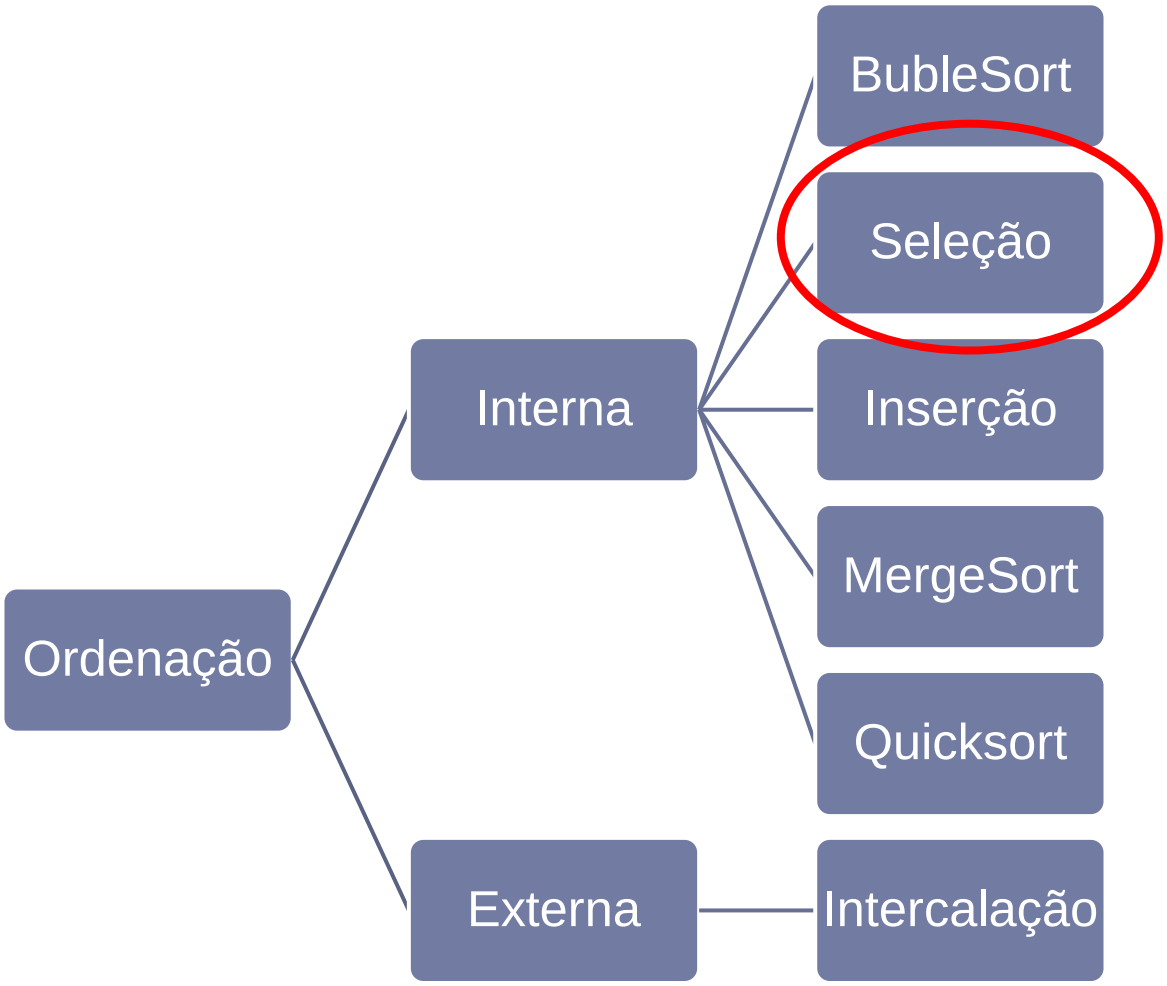
Classificação dos Métodos de Ordenação



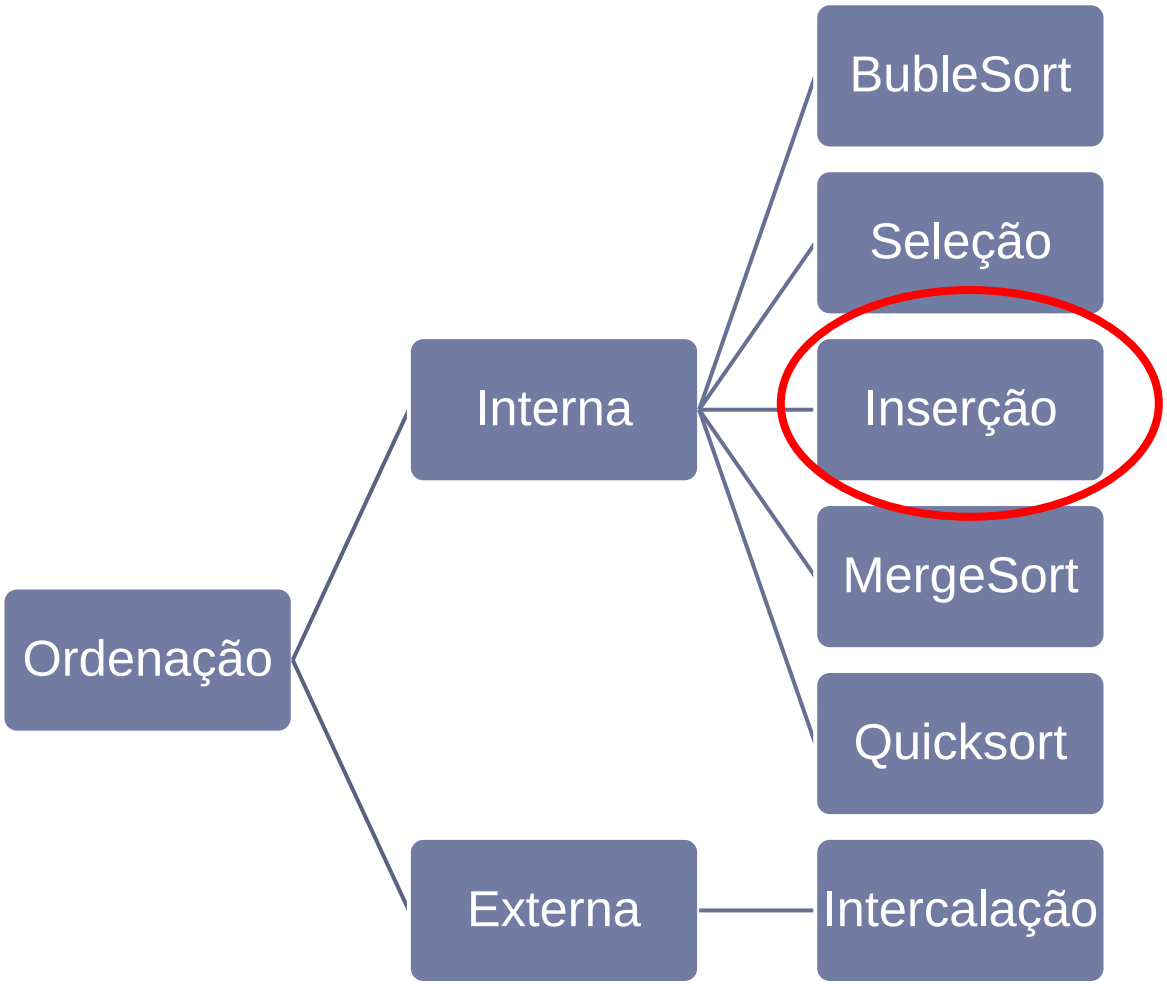
Classificação dos Métodos de Ordenação



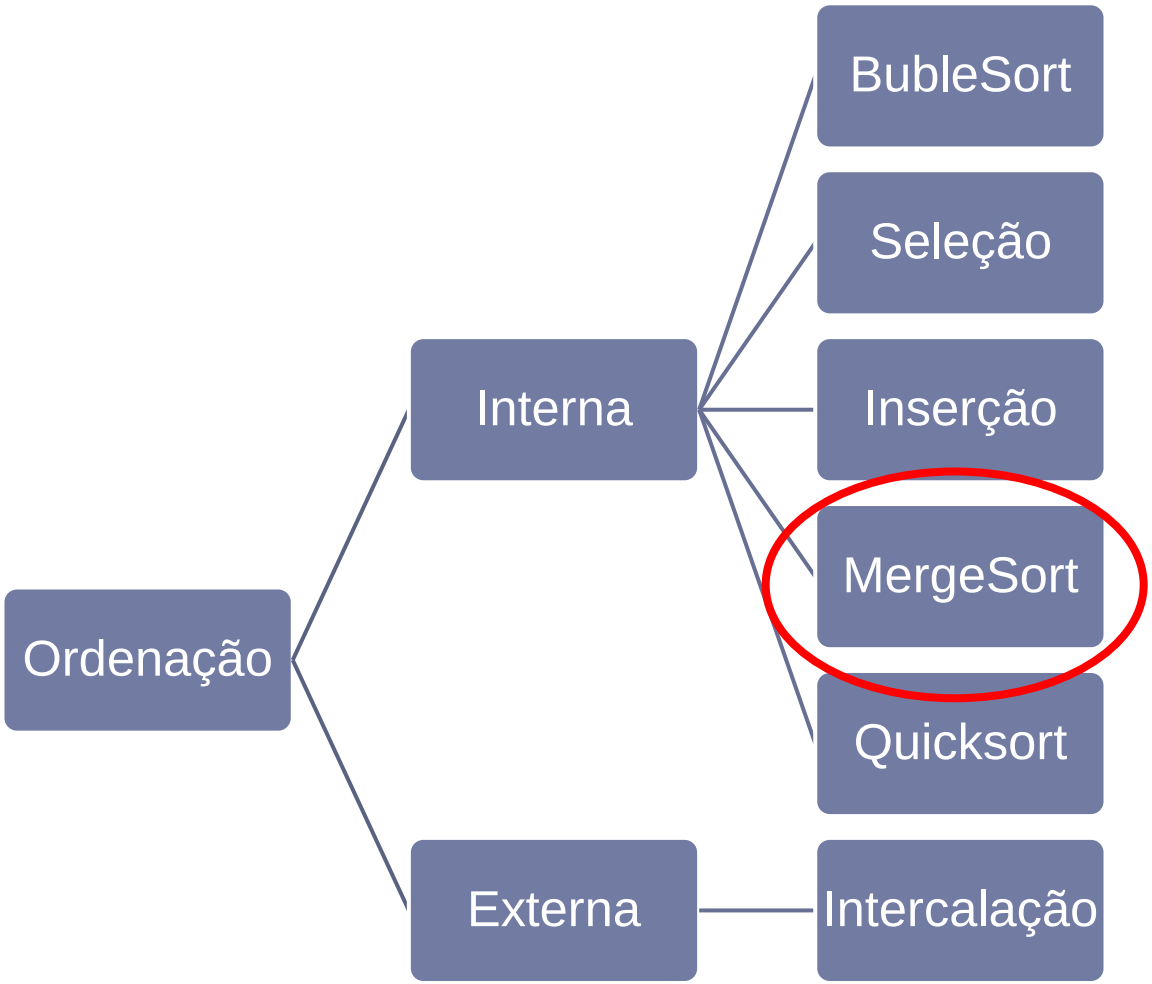
Classificação dos Métodos de Ordenação



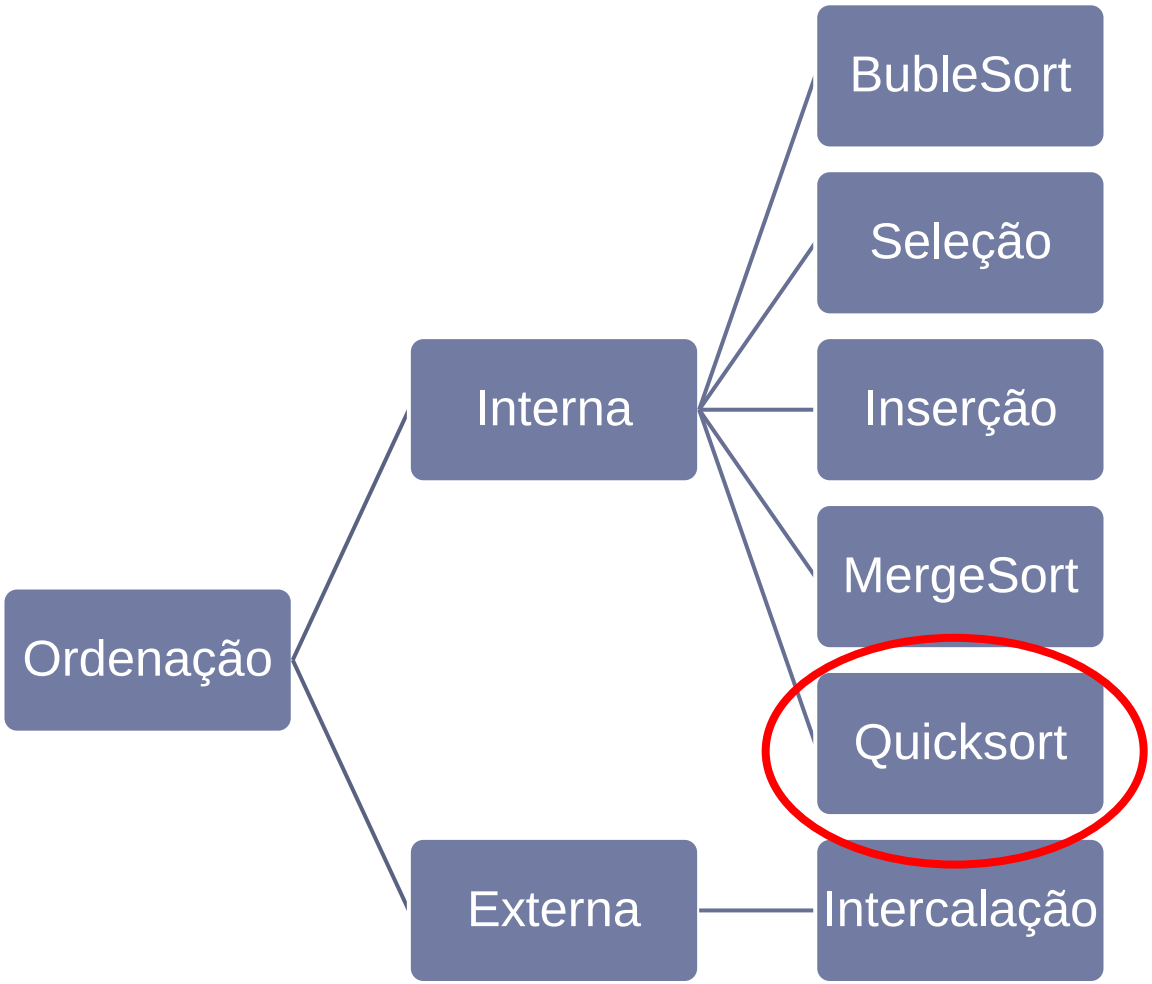
Classificação dos Métodos de Ordenação



Classificação dos Métodos de 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

3	5	4	1	2

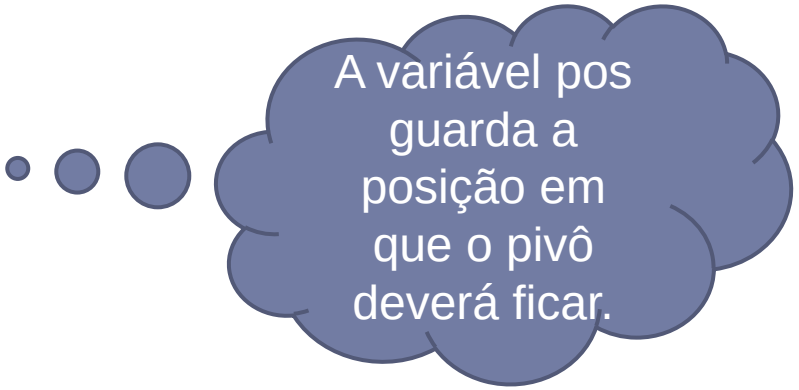
Particiona o vetor
Início = 0
Fim = 4

QuickSort

3	5	4	1	2

Pivô = vet[inicio]

pos = 0



QuickSort

3	5	4	1	2

Compara os demais elementos com o pivô

QuickSort

3	5	4	1	2

Se o elemento for maior que o pivô, não faz nada

pos = 0;

QuickSort

3	5	4	1	2

Se o elemento for maior que o pivô, não faz nada

pos = 0;

QuickSort

3	5	4	1	2

Se o elemento for menor que o pivô, a posição em que o pivô deve ficar cresce

pos = 1;

troca (vet[atual], vet[pos])

3	1	4	5	2

QuickSort

3	1	4	5	2

Se o elemento for menor que o pivô, a posição em que o pivô deve ficar cresce

pos = 2;

troca (vet[atual], vet[pos])

3	1	2	5	4

QuickSort

troca (pivô, vet[pos])

O 3 está na posição correta - ordenado

QuickSort

2	1	3	5	4

O problema se reduz a ordenar os outros vetores

Como?

QuickSort

2	1	3	5	4

Particiona o vetor
Início = 0
Fim = 1

Particiona o vetor
Início = 3
Fim = 4

2	1

5	4

QuickSort

Particiona o vetor
Início = 0
Fim = 1

Particiona o vetor
Início = 3
Fim = 4

Pos = 0

Pos = 3

QuickSort

Particiona o vetor
Início = 0
Fim = 1

Particiona o vetor
Início = 3
Fim = 4

Pos = 0

Pos = 3

Pos = 1

Pos = 3

QuickSort

Particiona o vetor
Início = 0
Fim = 1

Particiona o vetor
Início = 3
Fim = 4

Pos = 0

Pos = 3

Pos = 1

Pos = 3

Pos = 1

Pos = 4

QuickSort

Particiona o vetor
Início = 0
Fim = 1

Particiona o vetor
Início = 3
Fim = 4

Pos = 0

Pos = 3

Pos = 1

Pos = 3

Pos = 1

Pos = 4

QuickSort

Particiona o vetor
Início = 0
Fim = 1

Particiona o vetor
Início = 3
Fim = 4

Particiona o vetor
Início = 0
Fim = 0

Particiona o vetor
Início = 5
Fim = 4

QuickSort

2	1	3	5	4

Particiona o vetor
Início = 0
Fim = 1

Particiona o vetor
Início = 3
Fim = 4

1	2

4	5

Particiona o vetor
Início = 0
Fim = 0

Particiona o vetor
Início = 5
Fim = 4

QuickSort

1	2	3	4	5

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

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

22	33	55	77	99	11	44	66	88

Quicksort

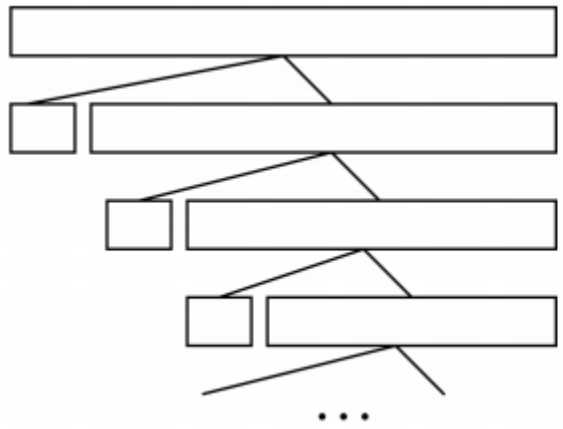
- A eficiência do algoritmo QuickSort depende do algoritmo de partição.

- 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.

QuickSort

- Exercício
  - Implementar o QuickSort

- DICA
  - Existem vários *applets* que ilustram a execução dos diferentes algoritmos de ordenação vistos em sala. Façam uso dessas ferramentas!
    - https://visualgo.net/sorting
    - https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html