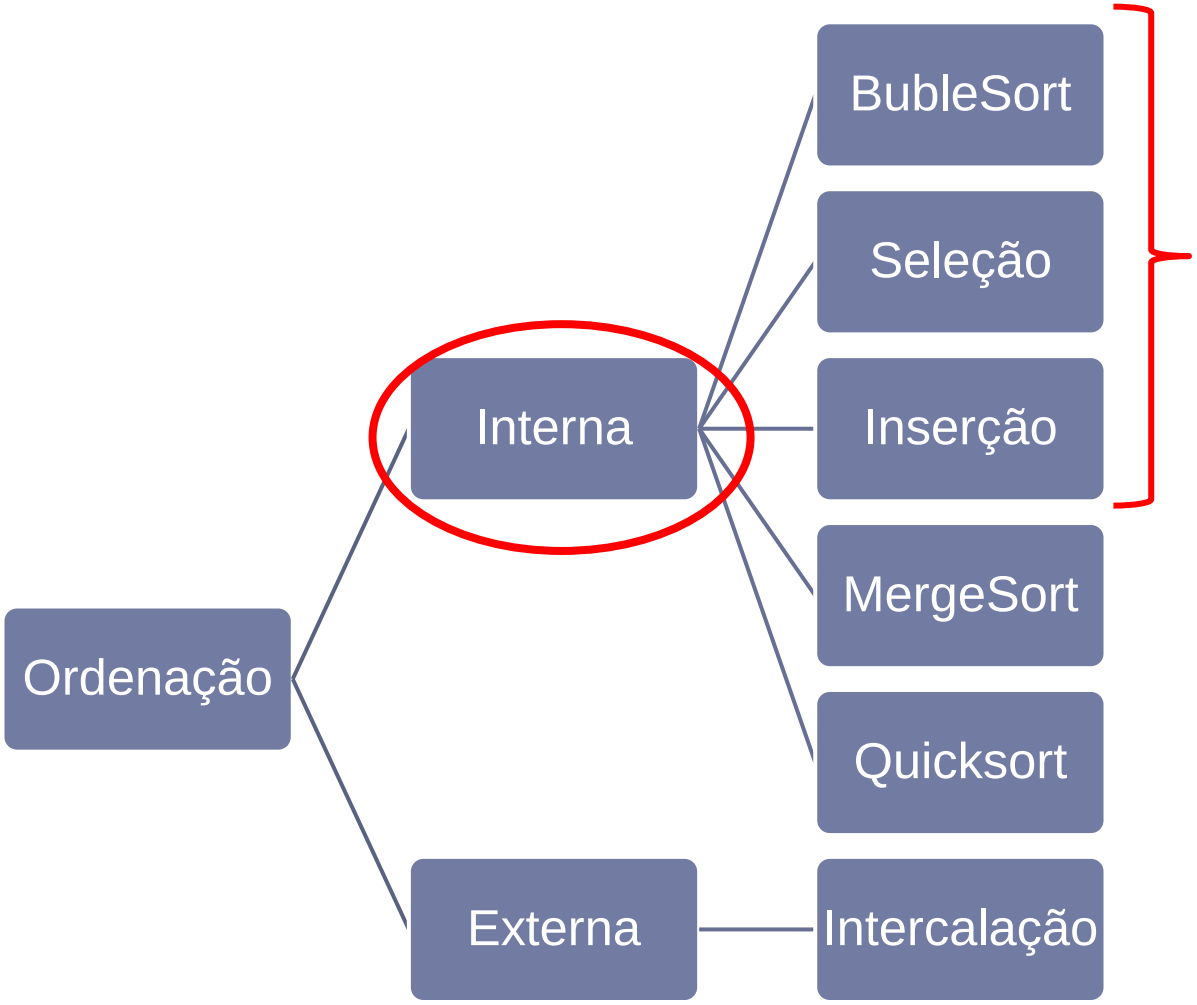
Algoritmo e Estrutura de Dados II
COM-112

Vanessa Souza

Universidade Federal de Itajubá – Instituto de Matemática e Computação

Ordenação

Classificação dos Métodos de Ordenação



Comparação entre os métodos

- Existe uma outra gama de algoritmos de ordenação mais eficientes ($O(n\log_2 n)$).
  - mergeSort
  - quickSort

- Esses algoritmos baseiam-se na estratégia de DIVIDIR PARA CONQUISTAR

- Implementações mais difíceis – estritamente recursivos

RECURSÃO

Revisão

FONTE : Ziviani

Algoritmo Recursivo

- Um método que chama a sim mesmo, direta ou indiretamente, é dito recursivo.

- O uso da recursividade geralmente permite uma descrição mais clara e concisa dos algoritmos, especialmente quando o problema a ser resolvido é recursivo por natureza ou utiliza estruturas recursivas, tais como as árvores.

Algoritmo Recursivo

- Um compilador implementa um método recursivo por meio de uma **pilha**, na qual são armazenados os dados usados em cada chamada de um método que ainda não terminou de processar.

- Todos os dados não globais vão para a pilha, pois o estado corrente da computação deve ser registrado para que possa ser recuperado de uma nova ativação de um método recursivo, quando a ativação anterior deverá prosseguir.

Algoritmo Recursivo

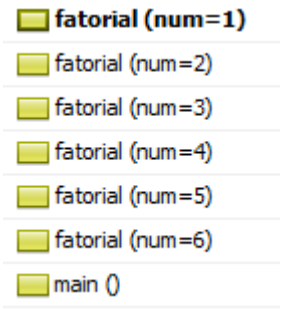
- Quando alcança sua condição de parada, o método retorna para quem chamou, utilizando o endereço de retorno que está no topo da pilha.

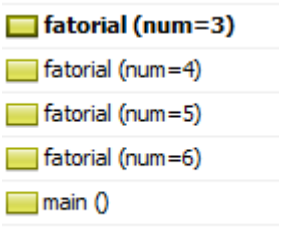
Algoritmo Recursivo

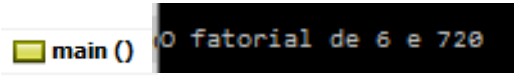
- Exemplo: Cálculo do Fatorial

```
int fatorial(int num)
{
    int fat;
    if (num <= 1)
        return 1;
    else
        fat = num * fatorial (num - 1);
    return fat;
}
```

```
Digite um numero para saber seu fatorial : 6
```







Algoritmo Recursivo

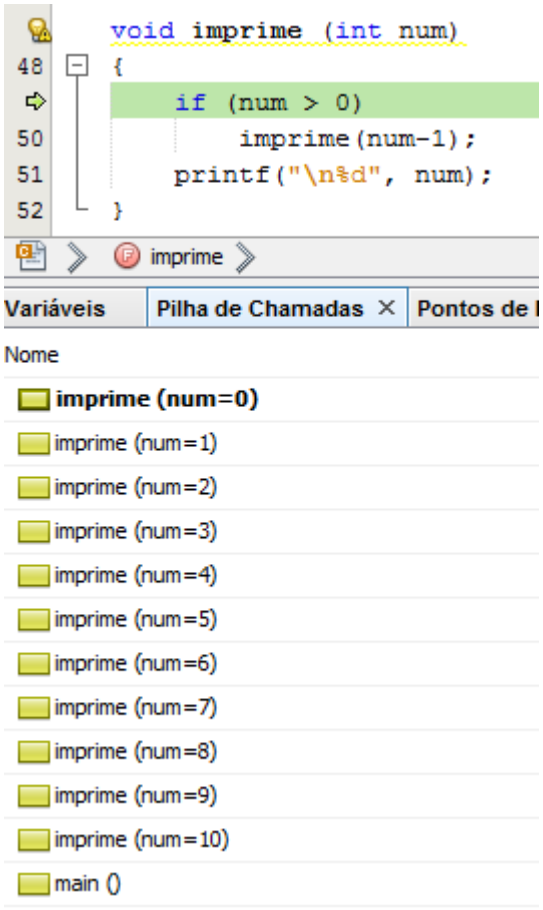
- Exemplo: Imprime recursivo

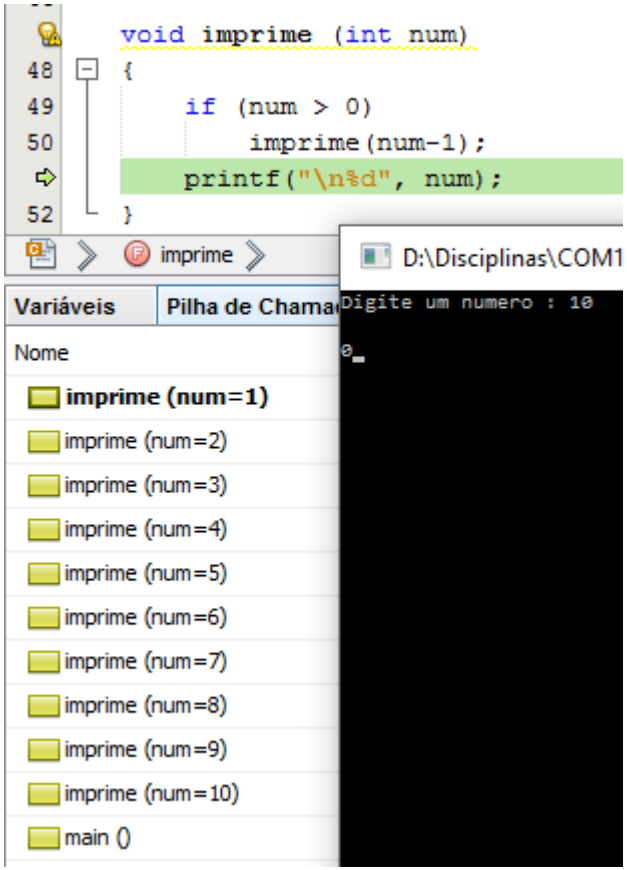
```
void imprime (int num)
{
    if (num > 0)
        imprime(num-1);
    printf("\n%d", num);
}
```

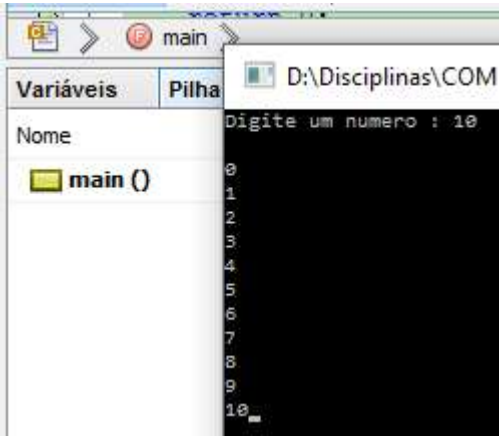
```
Digite um numero : 10
```

Algoritmo Recursivo

- Exemplo: Imprime recursivo







Recursividade

- Vantagens
  - Simplifica a solução de alguns problemas
  - Algoritmos recursivos são mais compactos para alguns tipos de algoritmo, mais legíveis e mais fáceis de ser compreendidos e implementados.

- Desvantagens
  - Por usarem intensamente a pilha de execução, os algoritmos recursivos tendem a ser mais lentos e a consumir mais memória que os iterativos, porém pode valer a pena sacrificar a eficiência em benefício da clareza.
  - Erros de implementação podem levar a estoure de pilha.

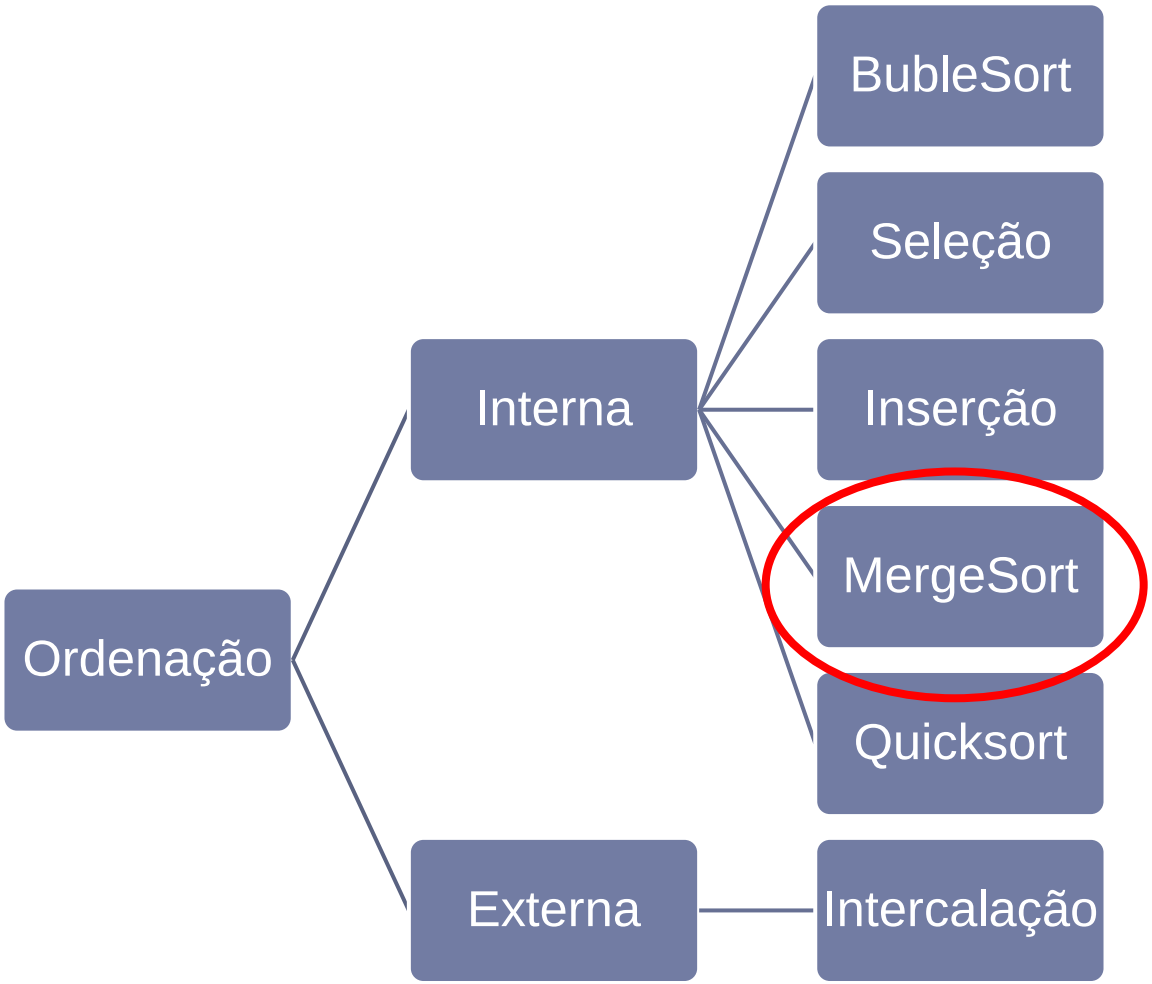
Recursividade

- Quadro comparativo da execução de algoritmos para o cálculo do Fibonacci.

n	10	20	30	50	100
Recursivo	8 ms	1 s	2 min	21 dias	10^9 anos
Iterativo	0.17 ms	0.33 ms	0.5 ms	0.75 ms	1.5 ms

- Evitar uso de recursividade quando existe uma solução óbvia por iteração.

Classificação dos Métodos de Ordenação



Ordenação por Fusão ou Intercalação

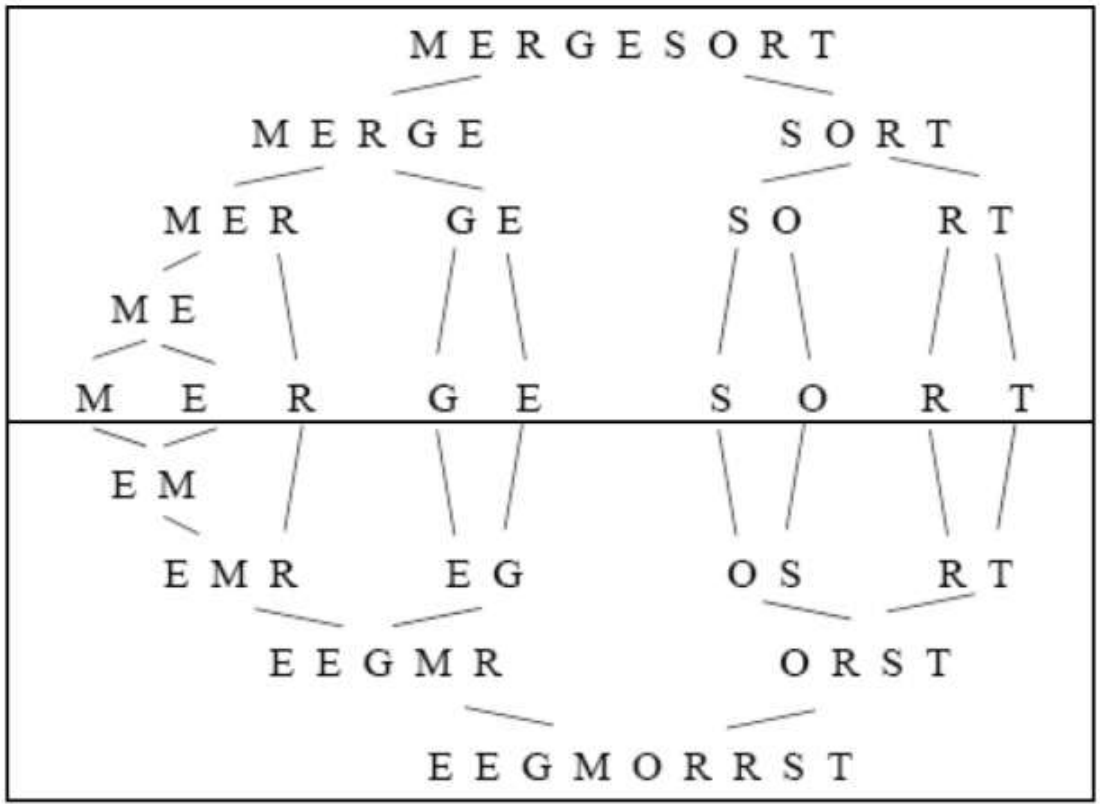
MergeSort

MergeSort

- **Ideia :** reduzir um problema em problemas menores, resolver cada um destes subproblemas e combinar as soluções parciais para obter a solução do problema original.

- É composto de duas fases:
  - Divisão
    - Divide o vetor original em dois outros de tamanhos menores, recursivamente, até obter vetores de tamanho 1
  - Junção ou Merge
    - Intercala os elementos dos dois vetores ordenados para obter a ordenação total.

MergeSort

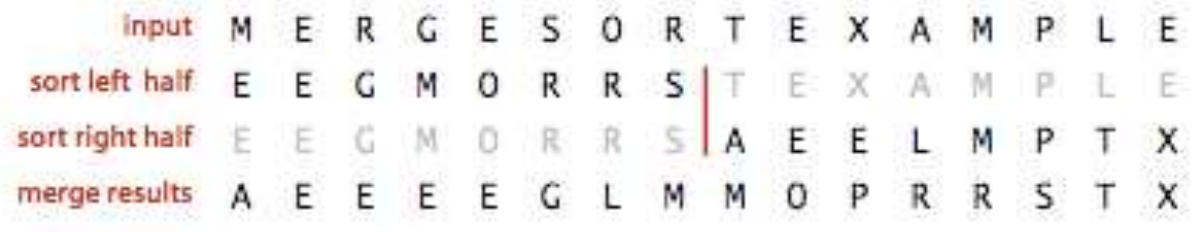


Fase de divisão

Fase de intercalação (merge)

MergeSort

- O mergeSort não é paralelo!



MergeSort

Algorithm 1 MergeSort

```
procedure MERGESORT(V, inicio, fim)                ▷ inicio e fim sao indices do vetor
    meio ← ⌊(inicio+fim)/2⌋
    if inicio < fim then
        MergeSort(V, inicio, meio)
        MergeSort(V, meio+1, fim)
        Merge(V, inicio, meio, fim)
    end if
end procedure

procedure MERGE(V, inicio, meio, fim)
    v1 ← V[inicio, meio]
    v2 ← V[meio+1, fim]
    vAux                                           ▷ vetor auxiliar com tamanho igual a (fim-inicio)+1
    while v1.size > 0 E v2.size > 0 do
        Compara os elementos de v1 e v2 e ordena-os no vetor auxiliar
    end while
    Copia o resto de v1 ou o resto de v2 para o vetor auxiliar
    Copia o vetor auxiliar para o vetor original
end procedure
```

MergeSort

- Montar a pilha de execução do procedimento MergeSort

0	1	2	3	4	5	6	7	8
M	E	R	G	E	S	O	R	T

```
procedure MERGESORT(V, inicio, fim)
    meio ← ⌊(inicio+fim)/2⌋
    if inicio < fim then
        MergeSort(V, inicio, meio)
        MergeSort(V, meio+1, fim)
        Merge(V, inicio, meio, fim)
    end if
end procedure
```

MergeSort

- Dividir
  - Início = 0
  - Fim = 4

0	1	2	3	4
5	3	1	2	4

MergeSort

- Dividir
  - Início = 0
  - Fim = 4
  - Meio = 2

Encontra o meio

5	3	1	2	4

MergeSort

- Dividir
  - Início = 0
  - Fim = 4
  - Meio = 2

Encontra o meio

5	3	1	2	4

Divide

5	3	1

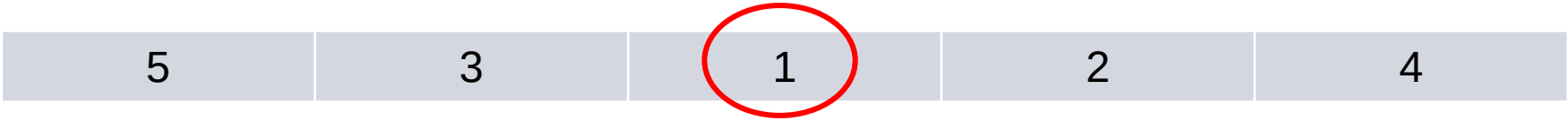
Início = 0
Fim = 2

2	4

Início = 3
Fim = 4

MergeSort

- Dividir
  - Início = 0
  - Fim = 4
  - Meio = 2



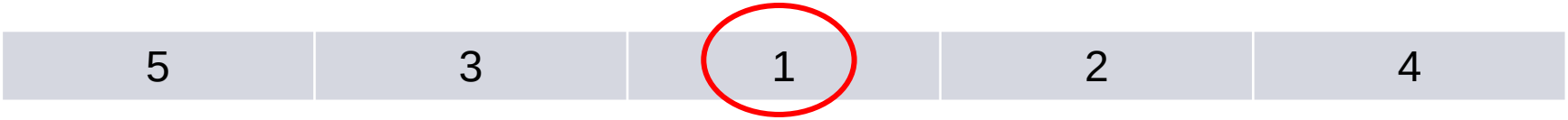
Encontra o meio

Início = 0
Fim = 2
Meio = 1

Início = 3
Fim = 4
Meio = 3

MergeSort

- Dividir
  - Início = 0
  - Fim = 4
  - Meio = 2



Divide

Início = 0
Fim = 1

Início = 2
Fim = 2

Início = 3
Fim = 3

Início = 4
Fim = 4

MergeSort

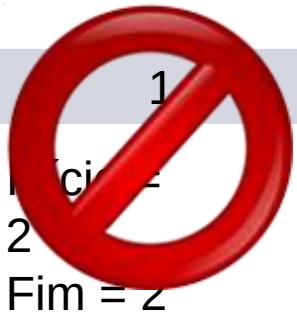
- Dividir
  - Início = 0
  - Fim = 4
  - Meio = 2

5	3	1	2	4

Encontra o meio

5	3	1	2	4

Início = 0
Fim = 1
Meio = 0



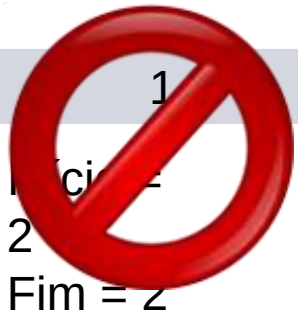
MergeSort

- Dividir
  - Início = 0
  - Fim = 4
  - Meio = 2

5	3	1	2	4

Divide

5	3	1	2	4
Início = 0 Fim = 0	Início = 1 Fim = 1	Início = 2 Fim = 2	Início = 3 Fim = 3	Início = 4 Fim = 4

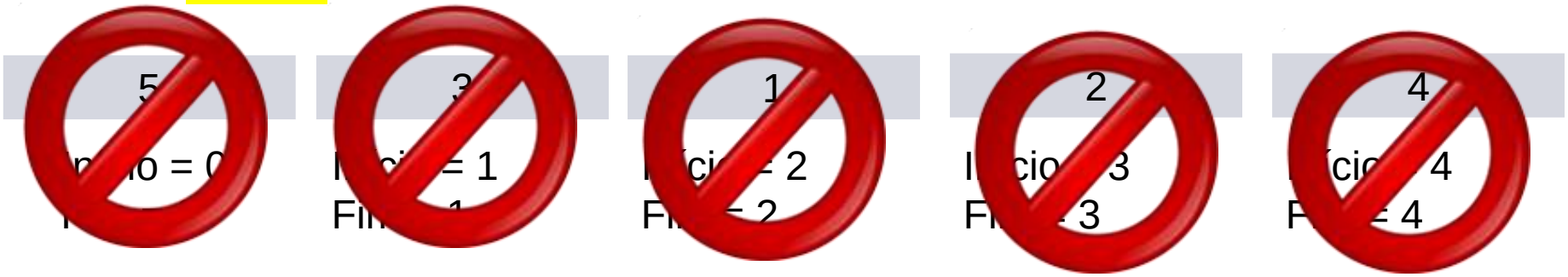


MergeSort

- Dividir
  - Início = 0
  - Fim = 4
  - Meio = 2

5	3	1	2	4

Divide

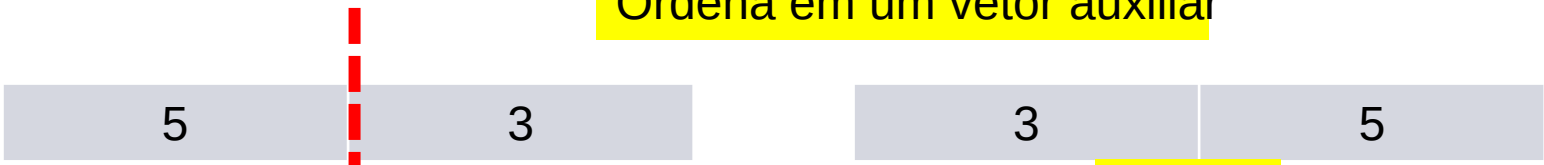


MergeSort

- Merge
  - Topo da pilha

Início = 0
Fim = 1
Meio = 0

Ordena em um vetor auxiliar



Vet aux

Início = 0
Fim = 1
Meio = 0

MergeSort

- Merge
  - Topo da pilha

Início = 0
Fim = 1
Meio = 0

3	5	1	2	4

Copia o vetor auxiliar para o vetor original

5	3

Início = 0
Fim = 1
Meio = 0

3	5

Vet aux

MergeSort

- Merge
  - Topo da pilha

Início = 0
Fim = 2
Meio = 1

Ordena em um vetor auxiliar

Início = 0
Fim = 2
Meio = 1

Vet aux

MergeSort

- Merge
  - Topo da pilha

Início = 0
Fim = 2
Meio = 1

1	3	5	2	4

Copia o vetor auxiliar para o vetor original

3	5	1

Início = 0
Fim = 2
Meio = 1

1	3	5

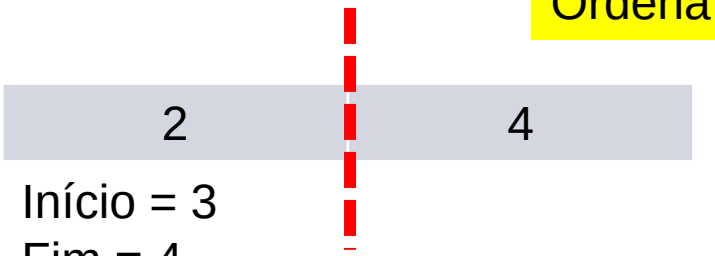
Vet aux

MergeSort

- Merge
  - Topo da pilha

Início = 3
Fim = 4
Meio = 3

Ordena em um vetor auxiliar



Início = 3
Fim = 4
Meio = 3

Vet aux

MergeSort

- Merge
  - Topo da pilha

Início = 3
Fim = 4
Meio = 3

Copia o vetor auxiliar para o vetor original

Início = 3
Fim = 4
Meio = 3

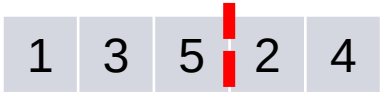
Vet aux

MergeSort

- Merge
  - Topo da pilha

Início = 0
Fim = 4
Meio = 2

Ordena em um vetor auxiliar



Início = 3
Fim = 4
Meio = 3

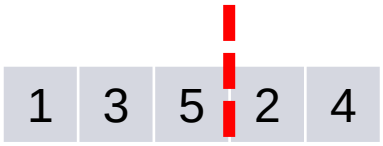
Vet aux

MergeSort

- Merge
  - Topo da pilha

Início = 0
Fim = 4
Meio = 2

Ordena em um vetor auxiliar



Início = 3
Fim = 4
Meio = 3

Vet aux

MergeSort

- Merge
  - Pilha vazia

1	2	3	4	5

Vetor Ordenado

MergeSort

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

18	7	1	3	8	6

MergeSort

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

22	33	55	77	99	11	44	66	88

MergeSort

- A eficiência do algoritmo depende de quão eficientemente será a intercalação dos dois vetores (ordenados) em um único vetor ordenado.

- A intercalação pode ser feita fazendo-se, no máximo, (n – 1) comparações, onde *n* é o número total de elementos dos dois vetores originais, ou seja, o algoritmo de intercalação é O(n).

- Como o número de elementos do vetor é reduzido à metade em cada chamada do mergesort, o número total de "rodadas" é $\log_2 n$.

- Assim, a complexidade assintótica do MergeSort é O(nlogn)

MergeSort

- O MergeSort é considerado um algoritmo ótimo, uma vez que ele tem sempre a mesma complexidade.

- No entanto, o MergeSort ocupa mais espaço na memória para fazer a intercalação dos sub-vetores.

Trabalho Prático

- 9 grupos de 5 pessoas
- Sorteio!

Tabela 1 : Atividades dos grupos.

Grupo 1	Analysis on Bubble Sort Algorithm Optimization
Grupo 2	The Rapid Sort
Grupo 3	MergeSort in-place
Grupo 4	MergeSort em uma lista encadeada
Grupo 5	Shellshort
Grupo 6	HeapSort
Grupo 7	TimSort
Grupo 8	QuickSort Empilha-Inteligente
Grupo 9	QuickSort com diferentes escolhas de pivô (meio, mediana, aleatório)

Exercício

- Implementar o MergeSort

Algorithm 1 MergeSort

```
procedure MERGESORT(V, inicio, fim)          ▷ inicio e fim sao indices do vetor
    meio ← ⌊(inicio+fim)/2⌋
    if inicio < fim then
        MergeSort(V, inicio, meio)
        MergeSort(V, meio+1, fim)
        Merge(V, inicio, meio, fim)
    end if
end procedure

procedure MERGE(V, inicio, meio, fim)
    v1 ← V[inicio, meio]
    v2 ← V[meio+1, fim]
    vAux                                     ▷ vetor auxiliar com tamanho igual a (fim-inicio)+1
    while v1.size > 0 E v2.size > 0 do
        Compara os elementos de v1 e v2 e ordena-os no vetor auxiliar
    end while
    Copia o resto de v1 ou o resto de v2 para o vetor auxiliar
    Copia o vetor auxiliar para o vetor original
end procedure
```