

Estrutura de Dados – 1º semestre de 2020

Professor Mestre Fabio Pereira da Silva

Divisão e Conquista

- Construção incremental
- Consiste em, inicialmente, resolver o problema para um subconjunto dos elementos da entrada e, então adicionar os demais elementos um a um.
- Em muitos casos, se os elementos forem adicionados em uma ordem ruim, o algoritmo não será eficiente.
- Ex: Calcule $n!$, recursivamente

Divisão e Conquista

- Dividir o problema em determinado número de subproblemas.
- Conquistar os subproblemas, resolvendo os recursivamente.
- Se o tamanho do subproblema for pequeno o bastante, então a solução é direta.
- Combinar as soluções fornecidas pelos subproblemas, a fim de produzir a solução para o problema original.

Merge Sort

- Este algoritmo tem como objetivo a reordenação de uma estrutura linear por **meio da quebra, intercalação e união dos n elementos existentes.**
- Em outras palavras, a estrutura a ser reordenada será, de forma **recursiva, subdividida em estruturas menores** até que não seja mais possível fazê-lo.
- **Classificação por Intercalação**

Merge Sort

- Em seguida, os elementos serão organizados de modo que cada subestrutura ficará ordenada. Feito isso, as subestruturas menores (agora ordenadas) serão unidas, sendo seus elementos ordenados por meio de intercalação.
- O mesmo processo repete-se até que todos os elementos estejam unidos em uma única estrutura organizada.

Merge Sort

- Merge Sort com uma sequência de entrada S com n elementos consiste de três passos:
- Divide: dividir S em duas sequências S_1 e S_2 de aproximadamente $n/2$ elementos cada
- Recursão: recursivamente ordene S_1 e S_2
- Conquista: junte S_1 e S_2 em uma única sequência ordenada

Merge Sort

- Dividir o vetor original em n sub-partes de tamanho 1;
- Intercalar os pares de sub-partes adjacentes, da esquerda para a direita em ordem crescente;
- Repetir o passo anterior até obter um único vetor de tamanho n , que evidentemente estará ordenado.

Algoritmo Merge Sort

```
01. mergesort(A[0...n - 1], inicio, fim)
02. |   se(inicio < fim)
03. |   |   meio ← (inicio + fim) / 2 //calcula o meio
04. |   |   mergesort(A, inicio, meio) //ordena o subvetor esquerdo
05. |   |   mergesort(A, meio + 1, fim) //ordena o subvetor direito
06. |   |   merge(A, inicio, meio, fim) //funde os subvetores esquerdo e direito
07. |   fim_se
08. fim_mergesort
```

Intercalação

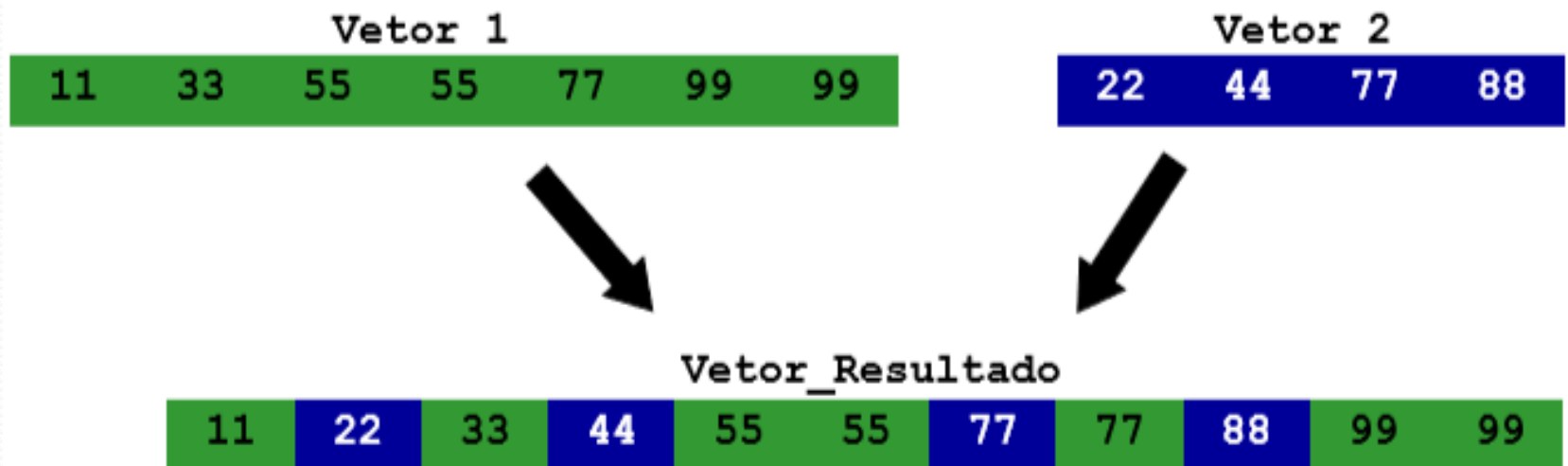
- Generalidades
 - Intercalação é o processo através do qual diversos arquivos sequenciais classificados por um mesmo critério são mesclados gerando um único arquivo sequencial.
- Algoritmo básico
 - De cada um dos arquivos a intercalar basta ter em memória um registro.
 - Consideramos cada arquivo como uma pilha. O registro atual em memória pode ser considerado o topo deste arquivo.
 - Em cada iteração do algoritmo e leitura dos registros, o topo da pilha com menor chave é gravado, e substituído pelo seu sucessor. Pilhas vazias têm topo igual ao maior valor.
 - O algoritmo termina quando todos os topos da pilha tiverem o maior valor

Intercalação

- A intercalação deve ser utilizada também quando há necessidade de unir dados de dois arquivos de dados.
- Desta forma, os dados poderiam ser acessados por meio de suas estruturas.
- Através de comandos de manipulação de arquivos, os dados entre os arquivos poderiam ser intercalados, gerando um novo arquivo de dados.

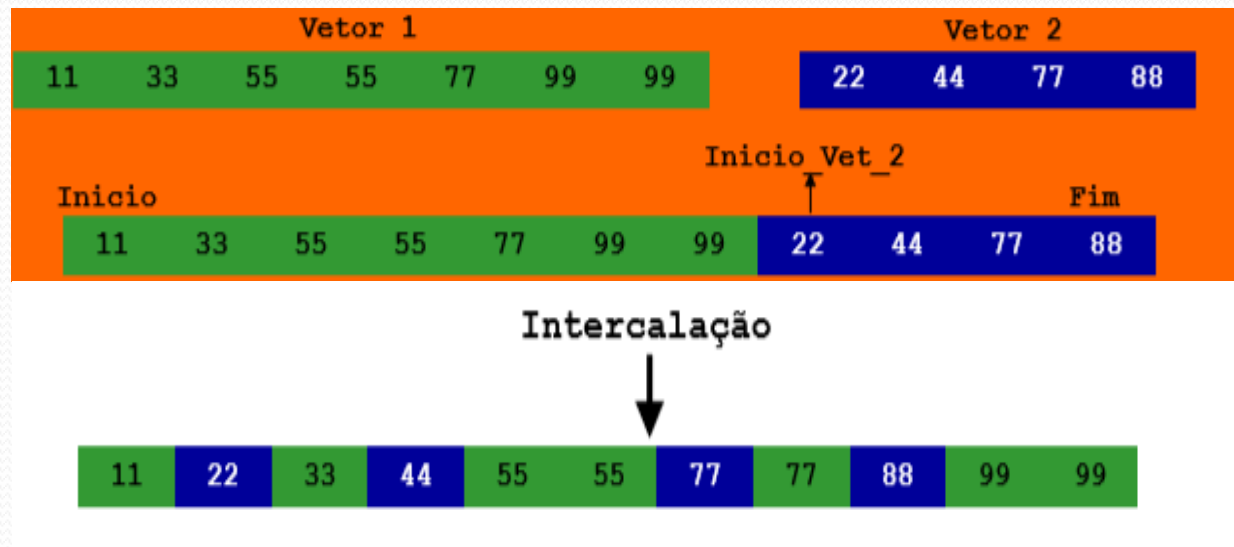
Intercalação

- A forma mais comum de intercalação é mesclar dois vetores (ordenados previamente).
- O resultado final é um vetor ordenado, com os elementos dos vetores utilizados na mesclagem.

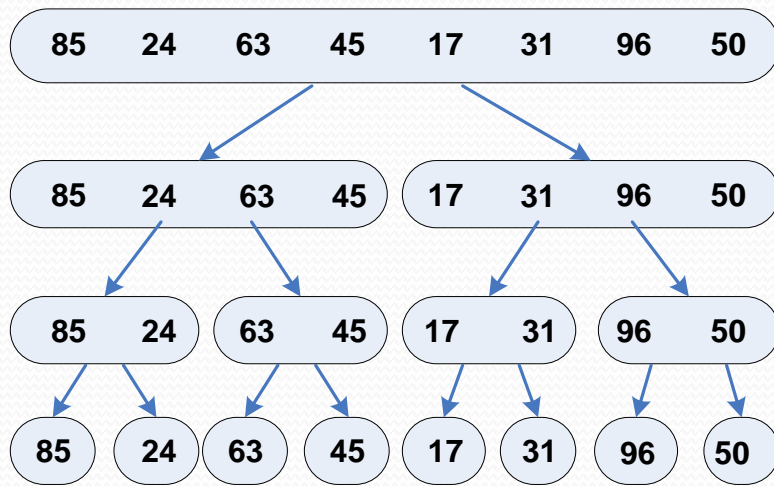


Intercalação

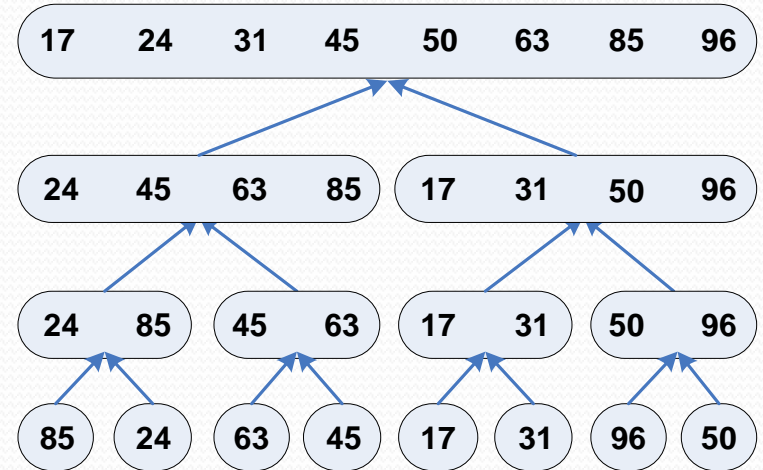
- Inicialmente, para que aconteça a intercalação (merge), os elementos dos dois vetores devem ser copiados para apenas um vetor.
- Para execução do algoritmo de intercalação, o índice de início do segundo vetor e o tamanho do novo vetor devem ser encontrados.



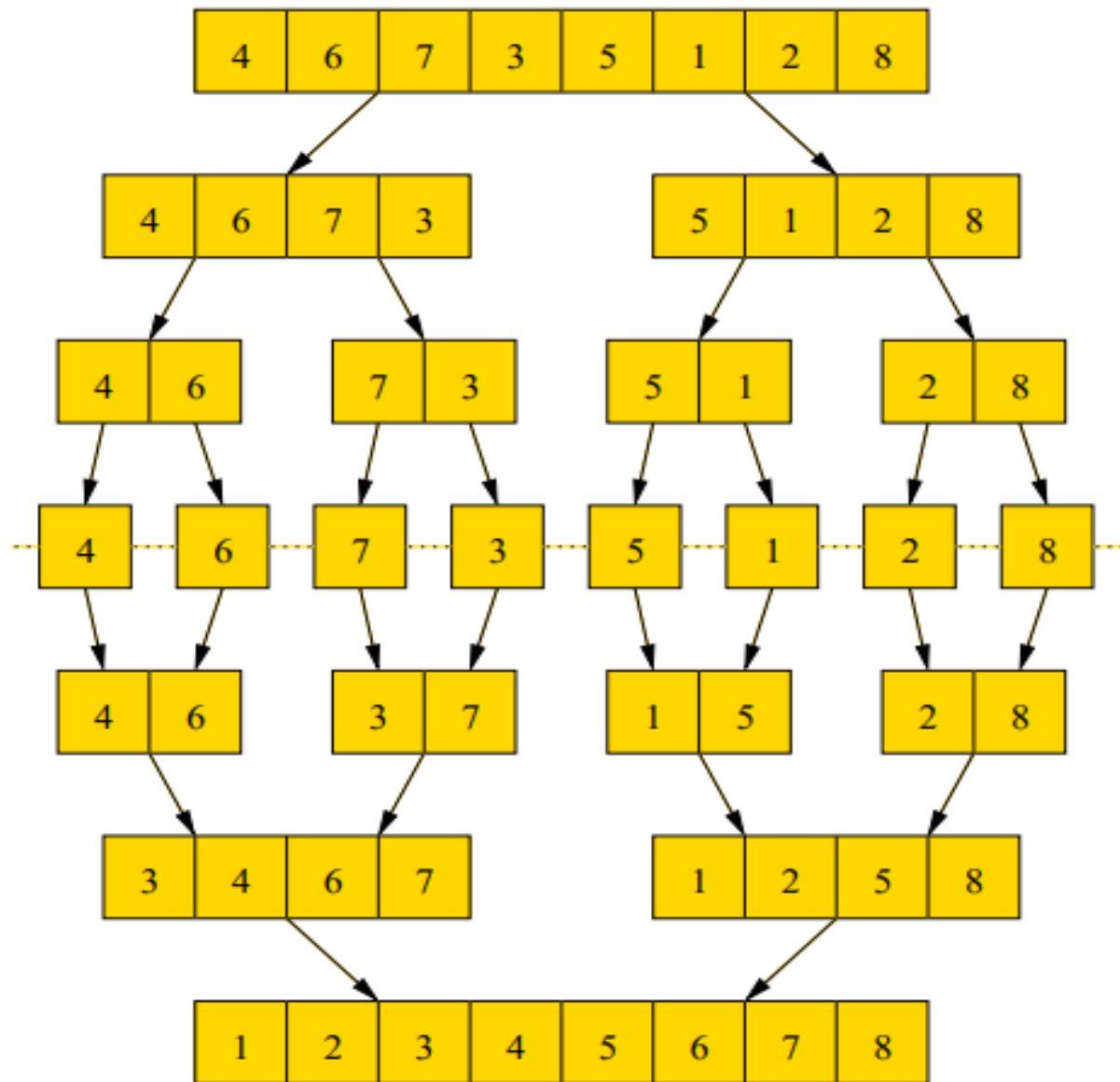
Exemplo Divisão e Conquista (MergeSort)

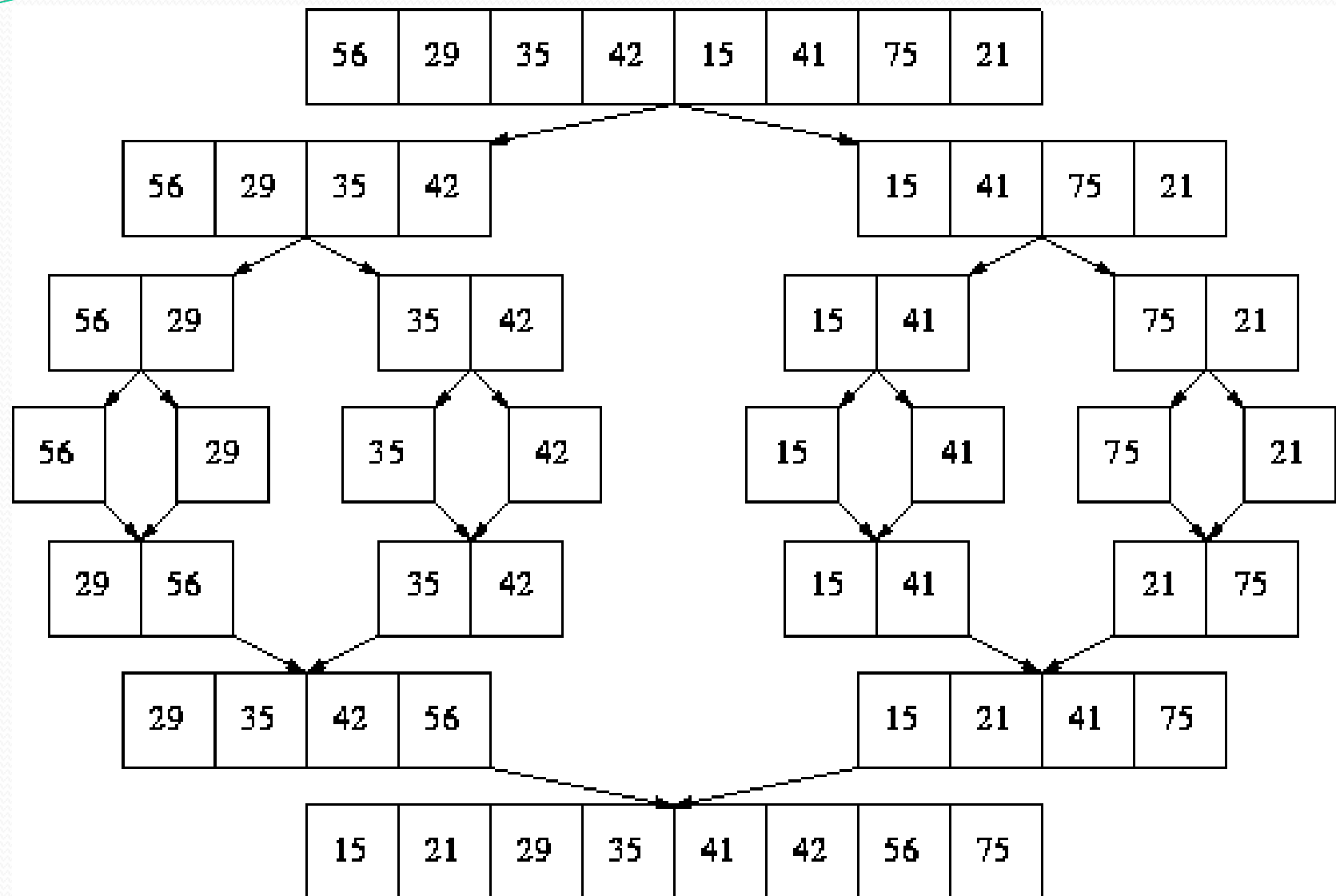


(a) Fase de Divisão

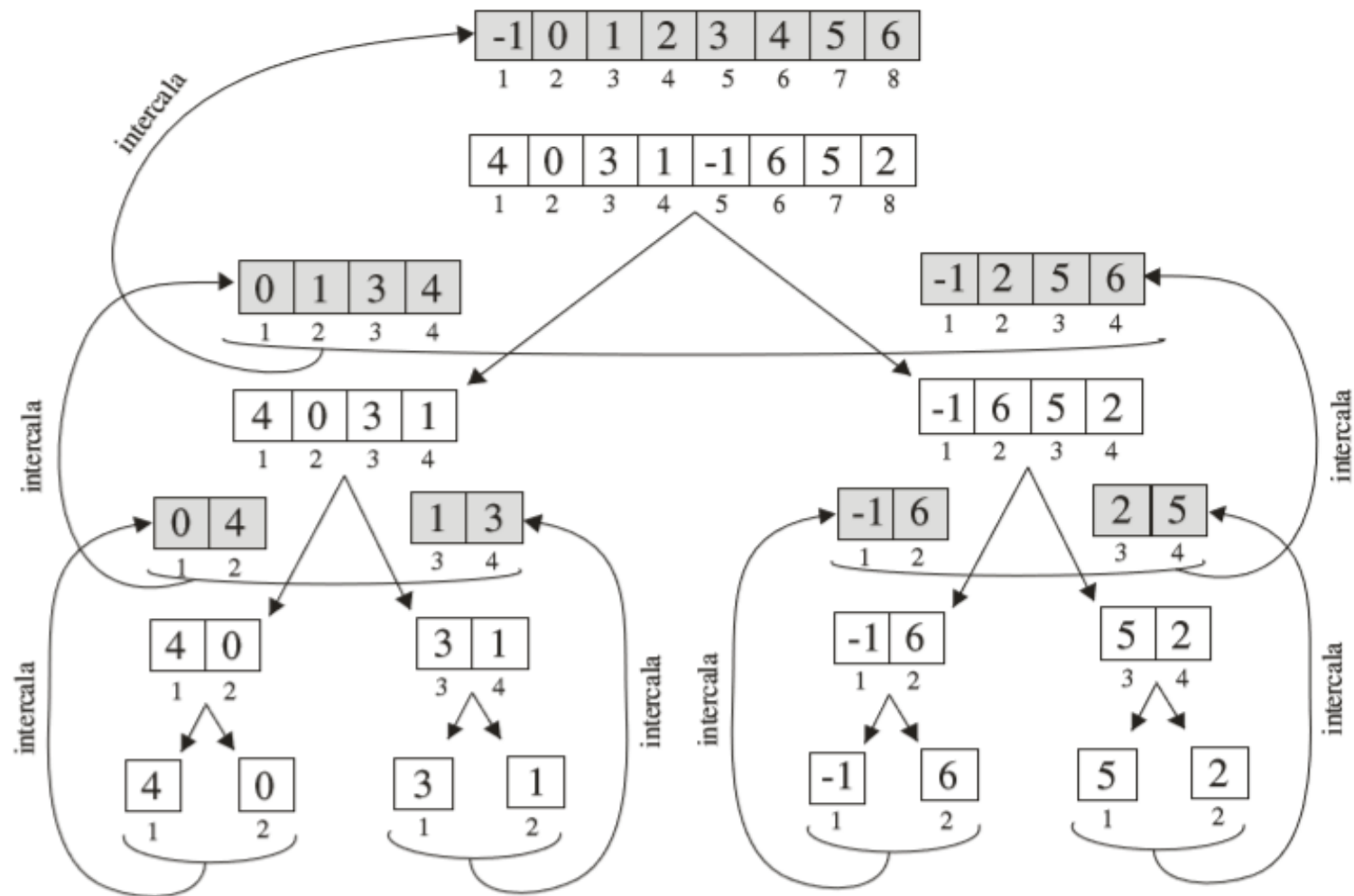


(b) Fase de Conquista





MergeSort (A, 1, 8)



Intercalação para ordenação

- Algoritmo MergeSort utiliza a ideia de intercalação para ordenar registros.
- Algoritmo criado por von Neumann
- Complexidade $O(N \log N)$ no caso médio e pior
- No pior caso é mais rápido do que o QuickSort
- Exemplo: Ordenar 10000 chaves
- Algoritmos de $O(N^2)$: 100.000.000 comparações
- MergeSort: 40.000 comparações

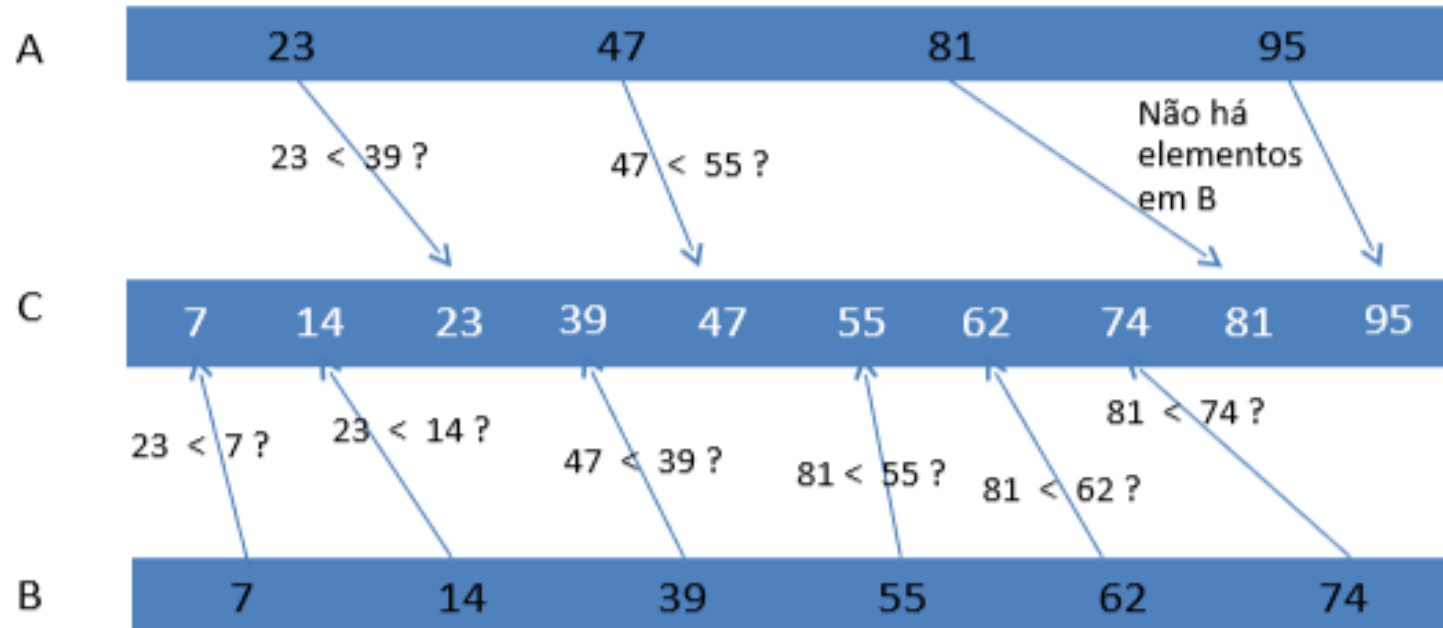
Intercalação para ordenação

- A ideia central é unir dois arrays que já estejam ordenados.
- Ou seja, unir dois arrays A e B já ordenados
- Em seguida, criar um terceiro array C que contenha os elementos de A e B já ordenados na ordem correta.

Exemplo

- Considere que temos dois arrays já ordenados A e B que não precisam ser do mesmo tamanho onde A possui 4 elementos e B possui 6 elementos.
- Eles serão unidos para a criação de um array C com 10 elementos ao final do processo de união

Exemplo

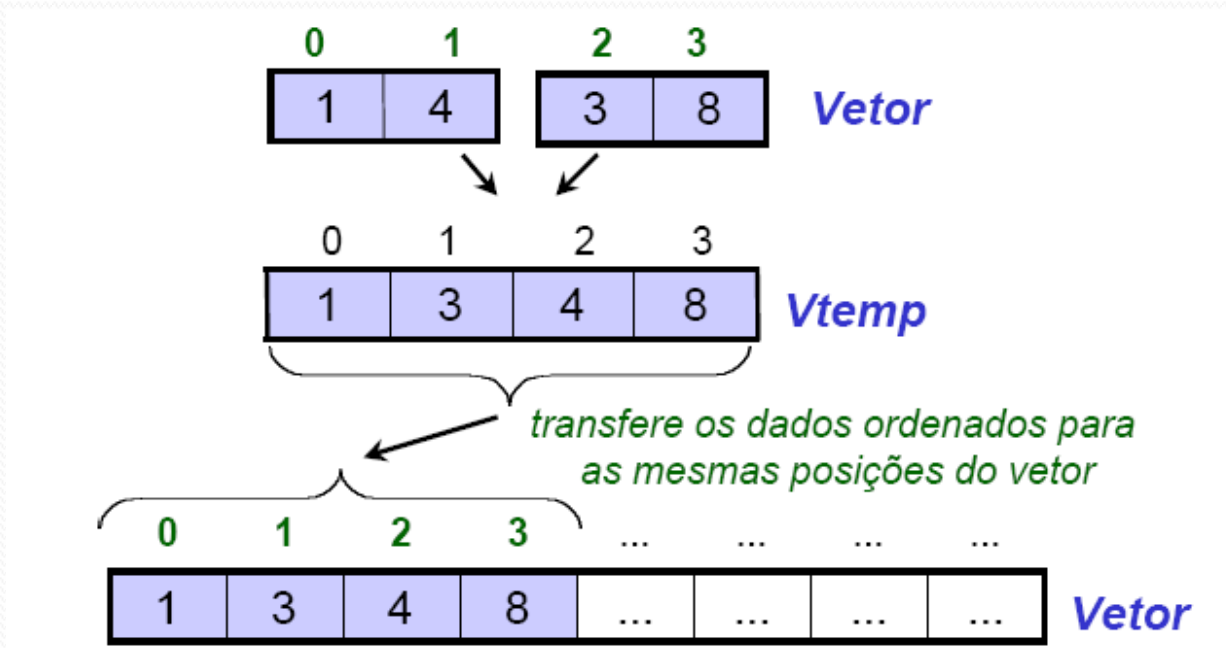


Ordenação

- A ideia do método MergeSort é dividir um array ao meio, ordenar cada metade e depois unir estas duas metades novamente formando o array original, porém ordenado. Como seria feita essa divisão e ordenação para que as metades possam ser unidas?

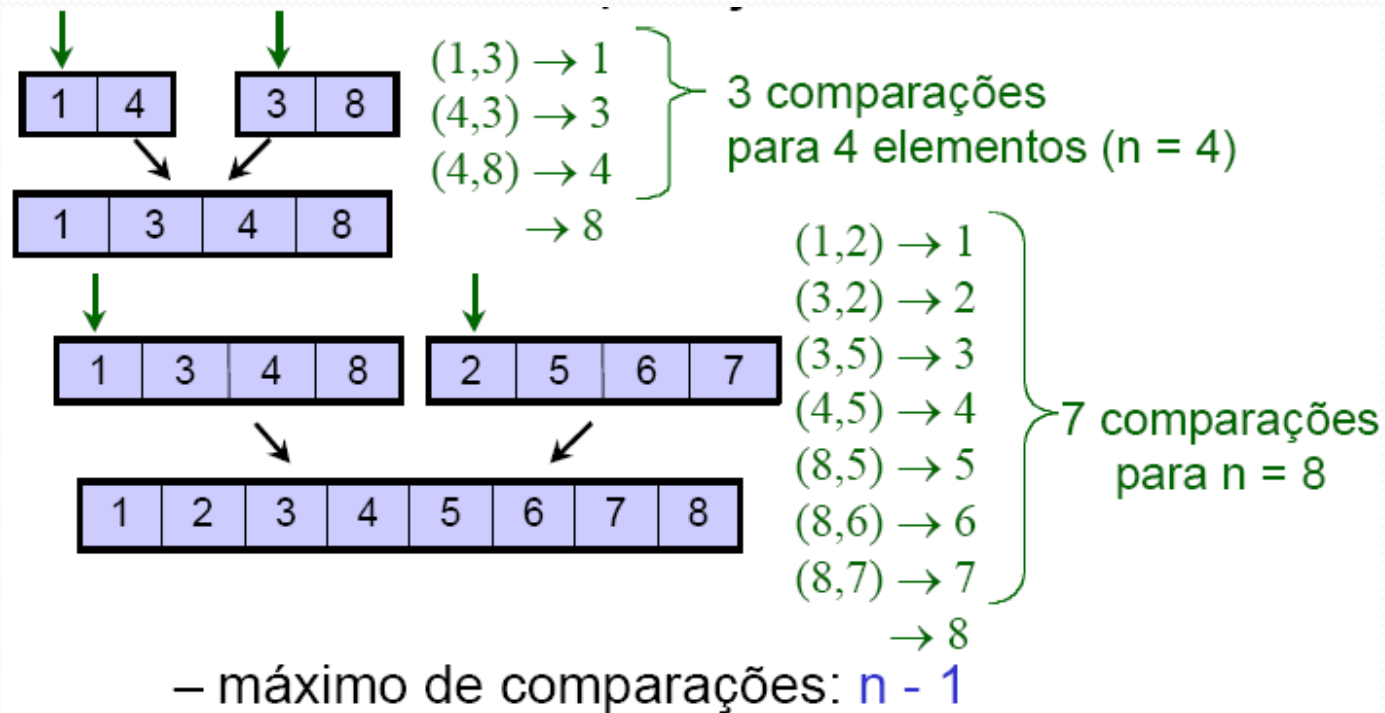
MergeSort: Junção ou Merge

- Após a ordenação, o conteúdo de *Vtemp* é transferido para o vetor.



MergeSort: Junção ou Merge

- Número de operações críticas ?



ArrayOriginal:

1ª chamada ao Merge-Sort:
Dividir o array em 2 partes:

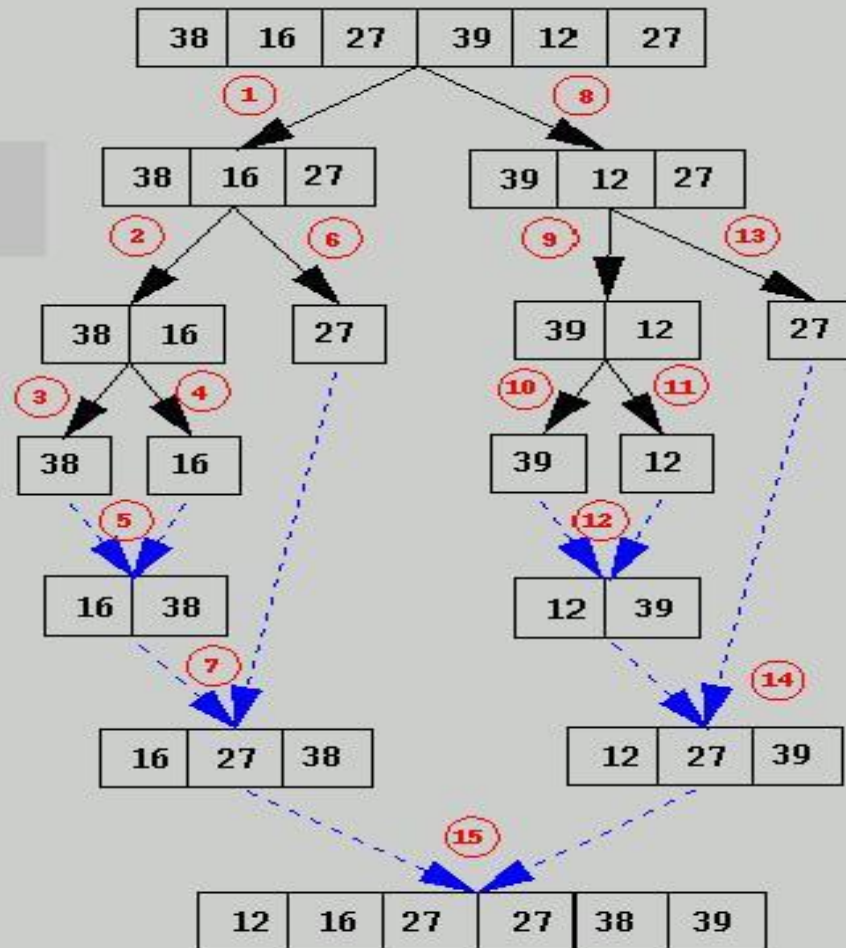
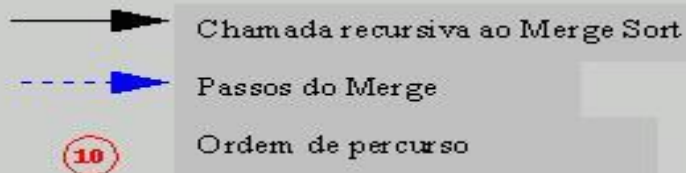
Segundo nível da
chamada à
recursividade:

Terceiro nível da
chamada à
recursividade:

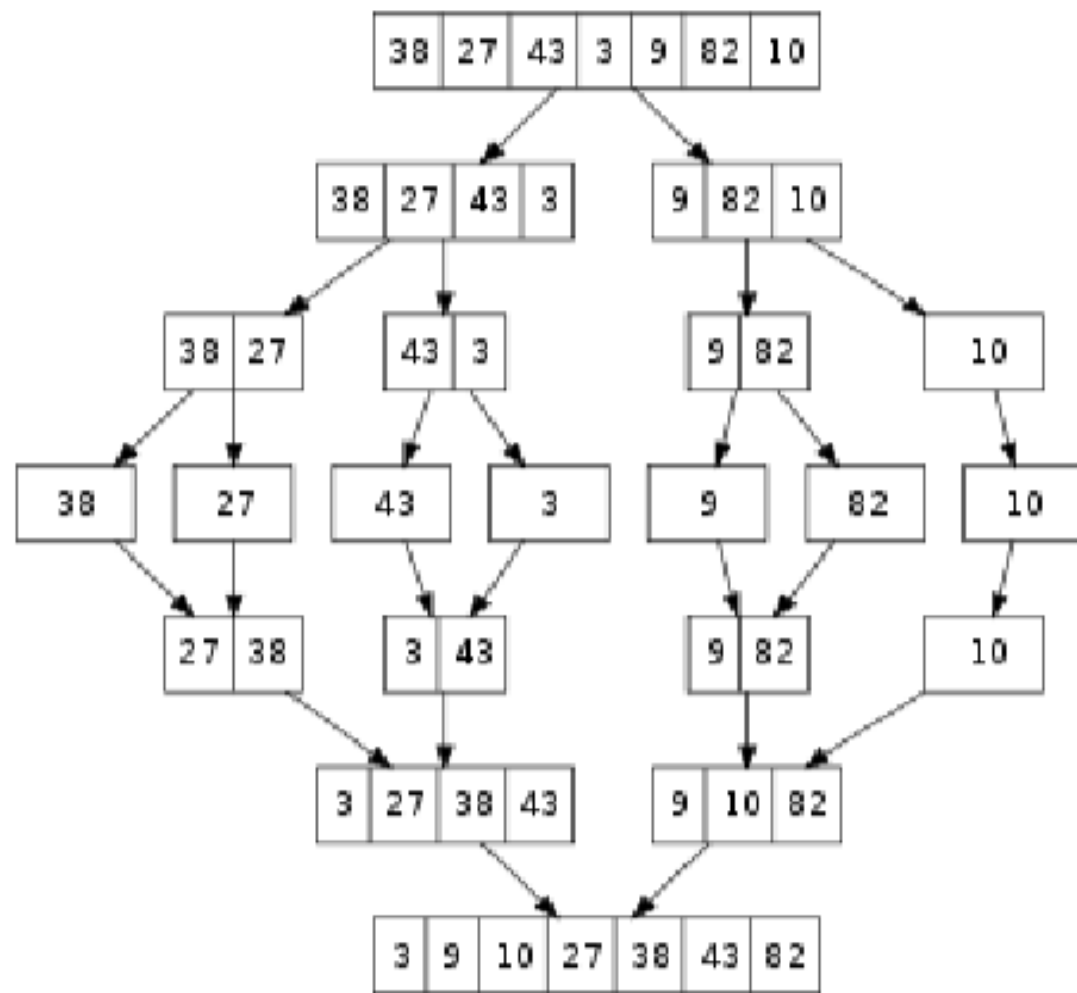
Fusão:

Fusão (ao 2º nível de
recursividade):

Fusão das duas listas já
ordenadas:



Ordenação



Implementação da Intercalação

```
void merge(int [] A, int p, int q, int r) {  
    // A subsequência A[p...q] está ordenada  
    // A subsequência A[q+1...r] está ordenada  
1:  int i, j, k;  
    // Faz cópias - seq1 = A[p...q] e seq2 = A[q+1...r]  
2:  int tamseq1 = q - p + 1; // tamanho da subsequência 1  
3:  int tamseq2 = r - q; // tamanho da subsequência 2  
4:  int [] seq1 = new int [tamseq1];  
5:  for(i=0; i < seq1.length; i++) {  
        seq1[i] = A[p+i];  
    }  
  
6:  int [] seq2 = new int [tamseq2];  
7:  for(j=0; j < seq2.length; j++) {  
        seq2[j] = A[q+j+1];  
    }  
}
```

Implementação da Intercalação

```
// Faz a junção das duas subsequências

8:  k = p; i = 0; j = 0;

9:  while (i < seq1.length && j < seq2.length) {
    // Pega o menor elemento das duas seqüências

10:    if(seq2[j] < seq1[i]) {
11:      A[k] = seq2[j];
12:      j++;
    }
    else {
13:      A[k] = seq1[i];
14:      i++;
    }
15:    k++;
  }
```

Implementação da Intercalação

```
// Completa com a seqüência que ainda não acabou

16: while (i < seq1.length) {
17:     A[k] = seq1[i];
18:     k++;
19:     i++;
    }

20: while (j < seq2.length) {
21:     A[k] = seq2[j];
22:     k++;
23:     j++;
    }
    // A subsequência A[p...r] está ordenada
}
```

Implementação da Ordenação

```
void mergeSort(int [] numeros, int ini, int fim) {  
  
    if(ini < fim) {  
        //Divisao  
1:   int meio = (ini + fim)/2;  
  
        // Conquista  
2:   mergeSort(numeros, ini, meio);  
3:   mergeSort(numeros, meio+1, fim);  
  
        // Combinação  
4:   merge(numeros, ini, meio, fim);  
    }  
    // Solução trivial: ordenacao de um único número.  
}
```

Exemplo

Considere o vetor abaixo:

26 69 25 53 59 27 41 0 33 16 35 43

```
mergesort(inteiro *vetor, inteiro inicio, inteiro fim)
{
    inteiro meio;

    Se (inicio < fim) {

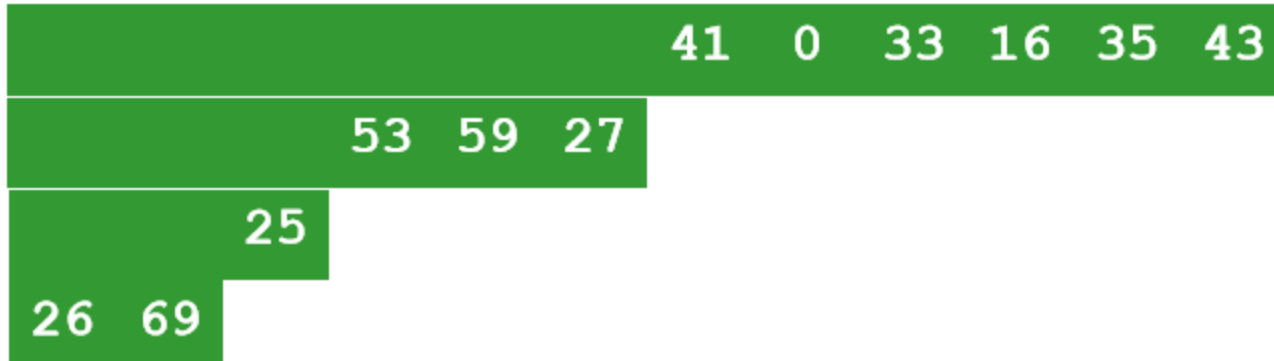
        meio = (inicio + fim) / 2;
        mergesort(vetor, inicio, meio);
        mergesort(vetor, meio+1, fim);
        intercala(vetor, inicio, meio, fim);
    }
}
```


■ Execução:

						41	0	33	16	35	43
26	69	25	53	59	27						
26	69	25									

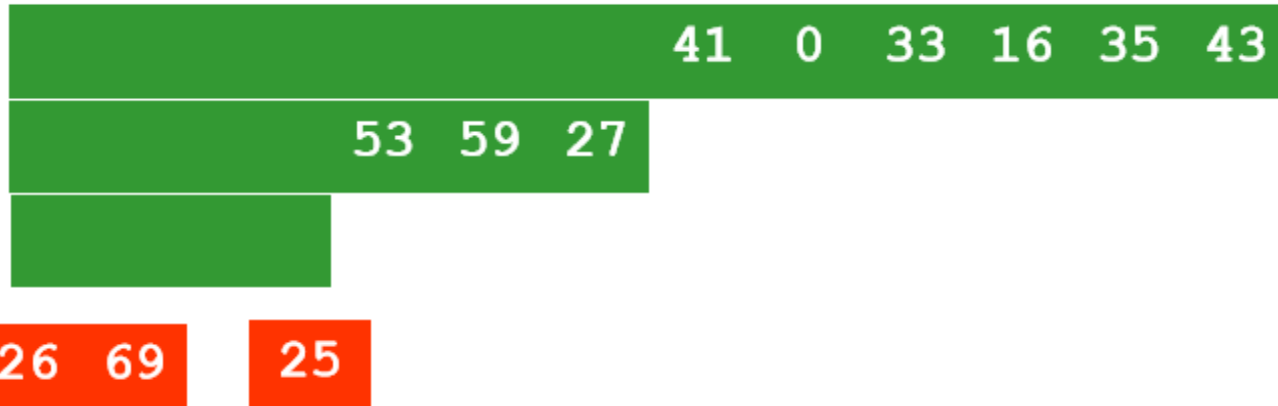
```
...  
Se (inicio < fim) {  
    meio = (inicio + fim) / 2;  
    → mergesort(vetor, inicio, meio);  
    mergesort(vetor, meio+1, fim);  
    intercala(vetor, inicio, meio, fim);  
}  
...
```


■ Execução:



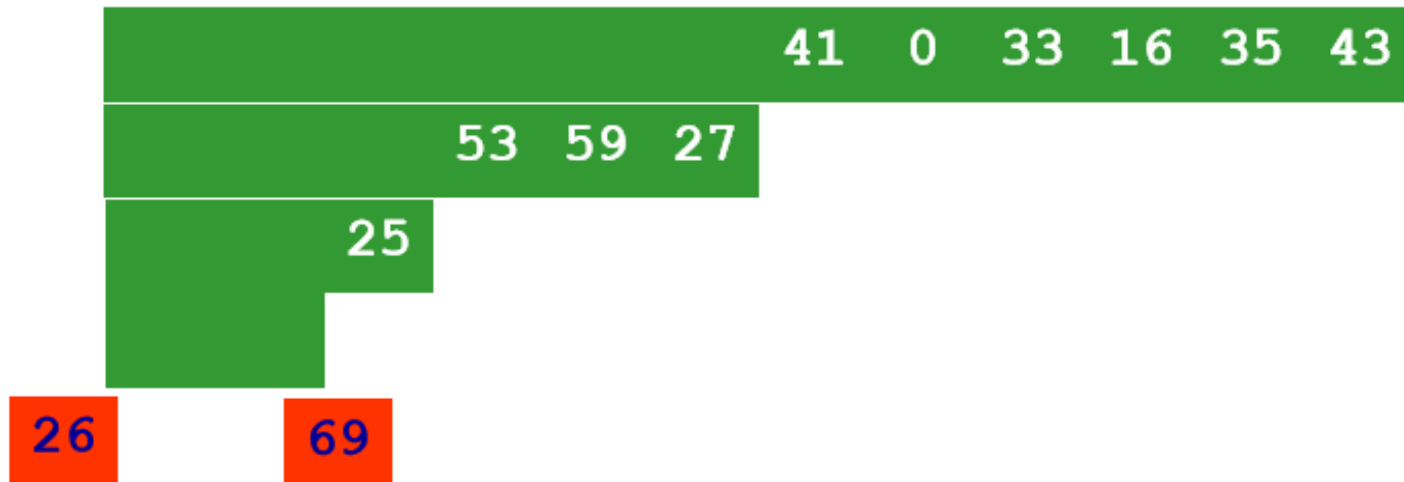
```
...  
Se (inicio < fim) {  
    meio = (inicio + fim) / 2;  
    → mergesort(vetor, inicio, meio);  
    mergesort(vetor, meio+1, fim);  
    intercala(vetor, inicio, meio, fim);  
}  
...
```

■ Execução:



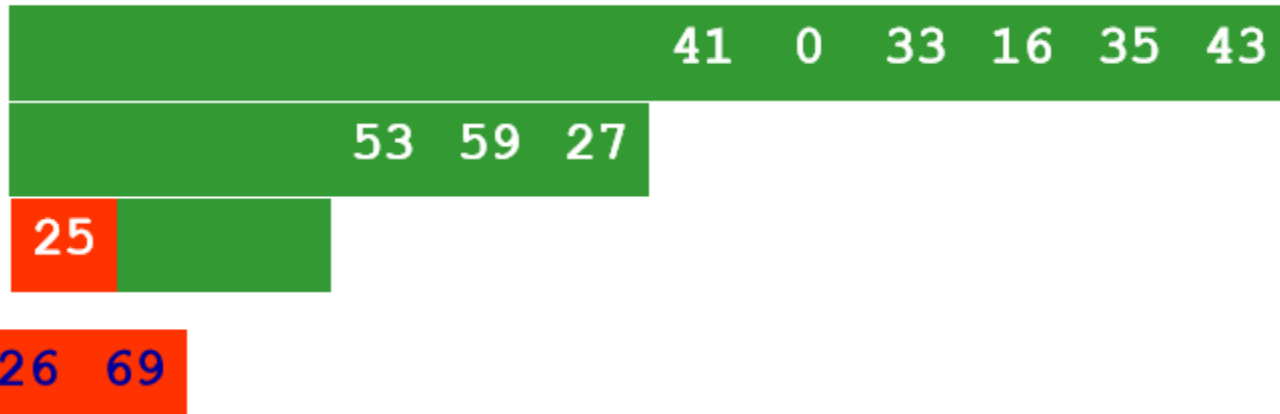
```
...  
Se (inicio < fim) {  
    meio = (inicio + fim) / 2;  
    mergesort(vetor, inicio, meio);  
    → mergesort(vetor, meio+1, fim);  
    intercala(vetor, inicio, meio, fim);  
}  
...
```

■ Execução:



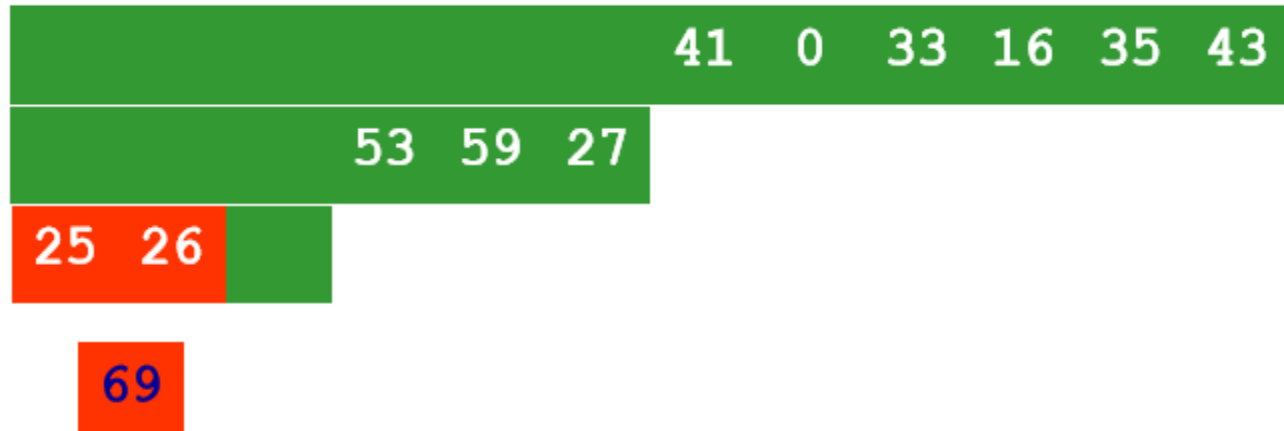
```
...  
Se (inicio < fim) {  
    meio = (inicio + fim) / 2;  
    mergesort(vetor, inicio, meio);  
    mergesort(vetor, meio+1, fim);  
    → intercala(vetor, inicio, meio, fim);  
}  
...
```

■ Execução:



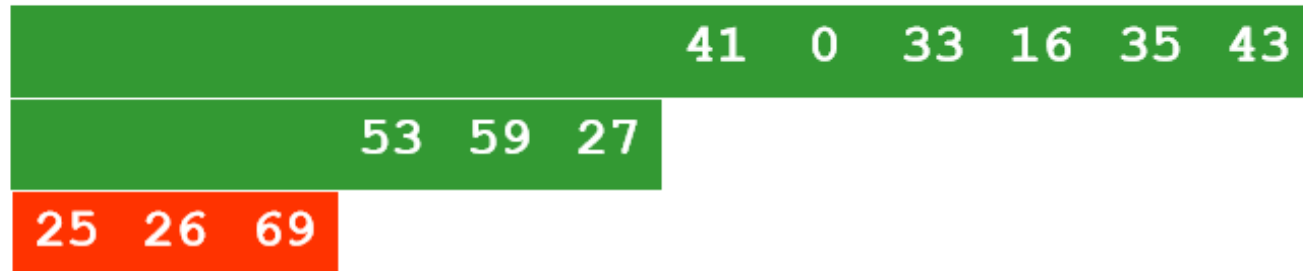
```
...  
Se (inicio < fim) {  
    meio = (inicio + fim) / 2;  
    mergesort(vetor, inicio, meio);  
    mergesort(vetor, meio+1, fim);  
    → intercala(vetor, inicio, meio, fim);  
}  
...
```

■ Execução:



```
...  
Se (inicio < fim) {  
  
    meio = (inicio + fim) / 2;  
    mergesort(vetor, inicio, meio);  
    mergesort(vetor, meio+1, fim);  
    → intercala(vetor, inicio, meio, fim);  
}  
...
```

■ Execução:



```
...  
Se (inicio < fim) {  
    meio = (inicio + fim) / 2;  
    mergesort(vetor, inicio, meio);  
    → mergesort(vetor, meio+1, fim);  
    intercala(vetor, inicio, meio, fim);  
}  
...
```

Desempenho dos algoritmos de Ordenação

	<i>QuickSort</i>	<i>HeapSort</i>	<i>MergeSort</i>
Pior caso	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Caso médio	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Melhor caso	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

- Classificando por melhor desempenho médio:
 - 1º) *MergeSort* (algoritmo mais simples)
 - 2º) *QuickSort*
 - 3º) *HeapSort*

Desempenho dos algoritmos de Ordenação

VETOR [10.000]									
Lista	Ordem Crescente			Ordem Decrescente			Ordem Aleatória		
Algoritmo	Tempo (s)	Comp.	Trocas	Tempo (s)	Comp.	Trocas	Tempo (s)	Comp.	Trocas
<i>Bubble Sort</i>	0,4269048	49995000	0	0,9847921	49995000	49995000	0,7649256	49995000	25084128,1
<i>Insertion Sort</i>	0,0003026	9999	0	0,4580984	9999	49995000	0,225615	9999	24963151
<i>Selection Sort</i>	0,3637704	49995000	0	0,3827789	49995000	5000	0,360824	49995000	9988
<i>Merge Sort</i>	0,0058387	135423	250848	0,0056613	74911	254944	0,006185	132011,1	252879
<i>Quick Sort</i>	0,4415975	49995000	0	1,192945	49995000	49995000	0,1867259	158055	25098217,7
<i>Shell Sort</i>	0,001431	75243	0	0,0019362	75243	161374	0,0034228	75243	161374

Contatos

- Email: fabio.silva321@fatec.sp.gov.br
- LinkedIn: <https://br.linkedin.com/in/b41a5269>
- Facebook: <https://www.facebook.com/fabio.silva.56211>