

## • Merge Sort (Buckets)

→ algoritmo de Ordenação por mistura

→ ideia básica: dividir e conquistar

- divide, recursivamente, o conjunto de dados até que cada subconjunto possua 1 elemento

- combina 2 subconjuntos de forma a obter 1 conjunto maior e mais ordenado

- processo se repete até que exista apenas 1 conjunto único

## \* Performance

- Melhor caso :  $O(N \log N)$

- Pior caso :  $O(N \log N)$

→ Estável

→ Desvantagens: recursivo, usa um vetor auxiliar durante a ordenação

- Código → 2 funções
  - merge Sort : ordenação, principal
  - merge : auxiliar, combina 2 métodos de forma ordenada

```

void mergeSort (int *V, int inicio, int fim) {
    int meio;
    if (inicio < fim) {
        meio = floor ((inicio + fim) / 2);
        mergeSort (V, inicio, meio); } chama a função para as
        mergeSort (V, meio+1, fim); } duas metades
        merge (V, inicio, meio, fim); } combina as duas
    }
}
  
```

02

• função auxiliar merge

void merge (int \*v, int inicio, int meio, int fim) {

int \*temp, p1, p2, tamanho, i, j, k;

int fim1 = 0, fim2 = 0;

tamanho = fim - inicio + 1;

p1 = inicio;

p2 = meio + 1;

temp = (int \*) malloc (tamanho \* sizeof (int));

if (temp != NULL) {

for (i = 0; i < tamanho; i++) {

    if (!fim1 && !fim2) {

        if (v[p1] < v[p2])

            temp[i] = v[p1++];

    else

            temp[i] = v[p2++];

    if (p1 > meio) fim1 = 1;

    if (p2 > fim) fim2 = 1;

    Combinar  
    Ordemando

    Vetor acabou?

} else {

    if (fim1) temp[i] = v[p1++];

    else temp[i] = v[p2++];

    Copia o que

    sobrou

}

}

```

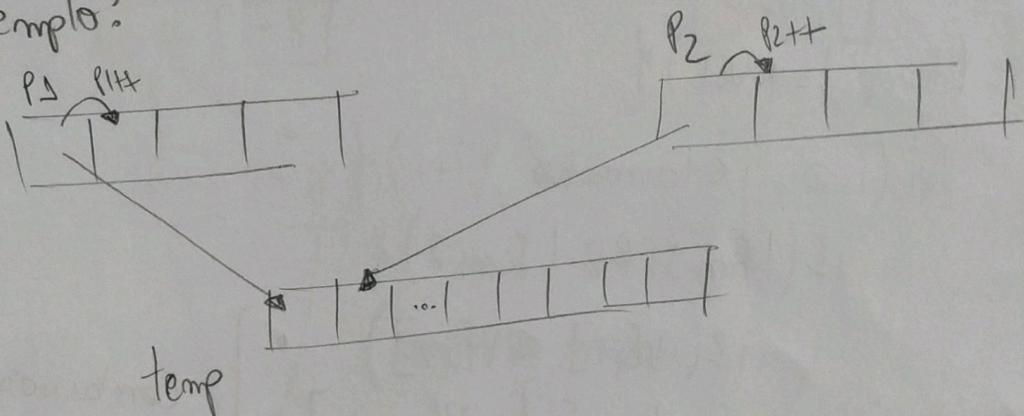
    for( j=0, k=inicio ; j < tamanho ; j++, k++ ) {
        v[k] = temp[j];
    }
}

// if
free( temp );
}

// merge

```

Exemplo:



- $p_1$  e  $p_2$  são dois variáveis indexadoras que são usadas para comparar as posições dois a dois, de cada metade
- quando uma das metades acaba, copia o resto

• Exemplo de funcionamento:

Vetor não-ordenado							
23	4	67	-8	90	54	21	
0	1	2	3	4	5	6	

Início	23	4	67	meio	-8	90	54	21	Fim
--------	----	---	----	------	----	----	----	----	-----

merge sort

Início	23	4	67	Fim
meio	23	4	67	-8

mais	23	4
------	----	---

meio	67	-8
------	----	----

23
----

4	23
---	----

-8	67
----	----

4	23
---	----

ETAPA DE MERGE

90
----

54	90
----	----

21
----

merge

-8	4	23	67
----	---	----	----

21	54	90
----	----	----

merge

-8	4	21	23	54	67	90
----	---	----	----	----	----	----

merge

Vetor Ordenado: -8 4 21 23 54 67 90

105

## • Pseudo Códigos

mergesort ( $v$ ,  $\text{início}$ ,  $\text{fim}$ )

Se  $\text{início} < \text{fim}$  então:

$$\text{meio} = ((\text{início} + \text{fim}) / 2)$$

mergeSort ( $v$ ,  $\text{início}$ ,  $\text{meio}$ )

mergeSort ( $v$ ,  $\text{meio} + 1$ ,  $\text{fim}$ )

merge ( $v$ ,  $\text{início}$ ,  $\text{meio}$ ,  $\text{fim}$ )

merge ( $v$ ,  $\text{início}$ ,  $\text{meio}$ ,  $\text{fim}$ )

• aloca um vetor temporário

•  $p_1 = \text{início}$

•  $p_2 = \text{meio} + 1$

• Enquanto ( $p_1 < \text{meio}$  E  $p_2 < \text{fim}$ ) faça:

• copia para o vetor temporário o menor valor entre  $v[p_1]$  e  $v[p_2]$

• Incrementa o contador correspondente

• Se  $p_1 == \text{meio}$  então:

• copia o que sobrou a partir de  $p_2$

• Senão

• copia o que sobrou a partir de  $p_1$

• Copia tudo do vetor temporário para o vetor original

## • Observações Gerais

Vantagem: estável

Desvantagens: gasto extra de memória para o vetor auxiliar

tempo de Execução:  $O(n \log n)$  em todos os casos