EDB-I

Estrutura de Dados Básicas I

Aula 14
Algoritmos de Ordenação
Merge Sort

(material baseado nas notas de aula do Prof. César Rennó Costa e Prof. Eiji Adachi)

UFRN UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

IMD INSTITUTO METRÓPOLE DIGITAL

[10 99 82 83 84 19 29]

Selection Sort

```
10 99 82 83 84 19 29
10 19 82 83 84 99 29
10 19 29 83 84 99 82
10 19 29 82 84 99 83
10 19 29 82 83 99 84
10 19 29 82 83 84 99
```

Insertion Sort

```
10 99 82 83 84 19 29
10 82 99 83 84 19 29
10 82 83 99 84 19 29
10 82 83 84 99 19 29
10 19 82 83 84 99 29
10 19 29 82 83 84 99
```

Bubble Sort

```
10 82 83 84 19 29 99
10 82 83 19 29 84 99
10 82 19 29 83 84 99
10 19 29 82 83 84 99
10 19 29 82 83 84 99
10 19 29 82 83 84 99
```

[51 39 33 21 24 27 25 39 56 43 23 40]

Selection Sort

```
21 39 33 51 24 27 25 39 56 43 23 40
21 23 33 51 24 27 25 39 56 43 39 40
21 23 24 51 33 27 25 39 56 43 39 40
21 23 24 25 33 27 51 39 56 43 39 40
21 23 24 25 27 33 51 39 56 43 39 40
21 23 24 25 27 33 51 39 56 43 39 40
21 23 24 25 27 33 39 51 56 43 39 40
21 23 24 25 27 33 39 39 56 43 51 40
21 23 24 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
```

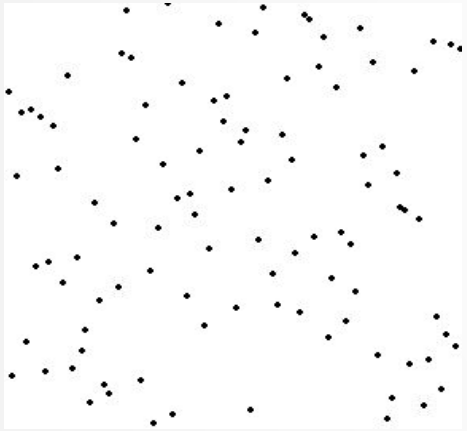
Insertion Sort

```
39 51 33 21 24 27 25 39 56 43 23 40
33 39 51 21 24 27 25 39 56 43 23 40
21 33 39 51 24 27 25 39 56 43 23 40
21 24 33 39 51 27 25 39 56 43 23 40
21 24 27 33 39 51 25 39 56 43 23 40
21 24 25 27 33 39 51 39 56 43 23 40
21 24 25 27 33 39 39 51 56 43 23 40
21 24 25 27 33 39 39 51 56 43 23 40
21 24 25 27 33 39 39 43 51 56 23 40
21 23 24 25 27 33 39 39 43 51 56 40
21 23 24 25 27 33 39 39 40 43 51 56
```

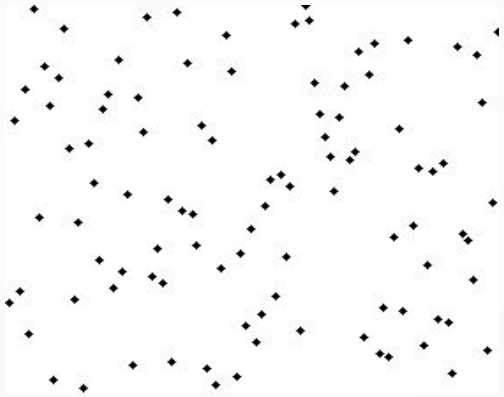
Bubble Sort

```
39 33 21 24 27 25 39 51 43 23 40 56
33 21 24 27 25 39 39 43 23 40 51 56
21 24 27 25 33 39 39 23 40 43 51 56
21 24 25 27 33 39 23 39 40 43 51 56
21 24 25 27 33 23 39 39 40 43 51 56
21 24 25 27 23 33 39 39 40 43 51 56
21 24 25 23 27 33 39 39 40 43 51 56
21 24 23 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
```

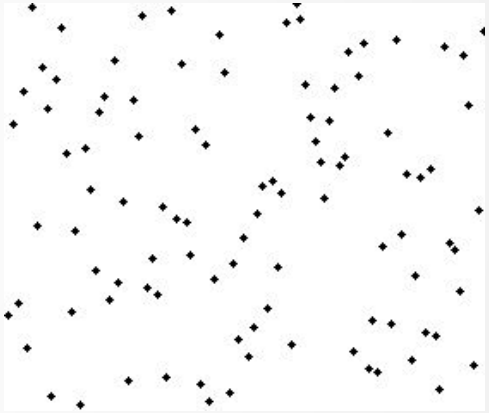
Seleção



Inserção



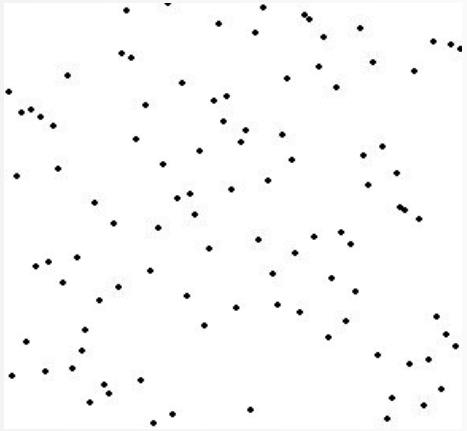
Bubble sort



[10 99 82 83 84 19 29]

[51 39 33 21 24 27 25 39 56 43 23 40]

Seleção



Selection Sort

```
10 99 82 83 84 19 29
10 19 82 83 84 99 29
10 19 29 83 84 99 82
10 19 29 82 84 99 83
10 19 29 82 83 99 84
10 19 29 82 83 84 99
```

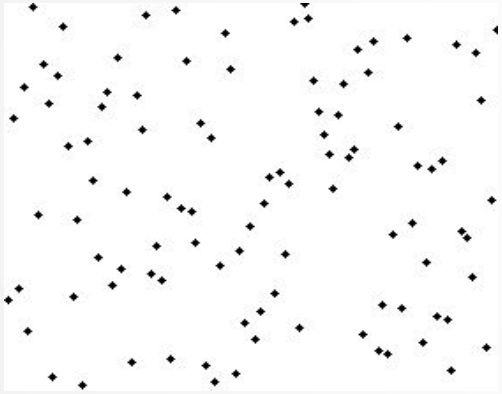
Selection Sort

```
21 39 33 51 24 27 25 39 56 43 23 40
21 23 33 51 24 27 25 39 56 43 39 40
21 23 24 51 33 27 25 39 56 43 39 40
21 23 24 25 33 27 51 39 56 43 39 40
21 23 24 25 27 33 51 39 56 43 39 40
21 23 24 25 27 33 51 39 56 43 39 40
21 23 24 25 27 33 39 51 56 43 39 40
21 23 24 25 27 33 39 39 56 43 51 40
21 23 24 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
```

[10 99 82 83 84 19 29]

[51 39 33 21 24 27 25 39 56 43 23 40]

Inserção



Insertion Sort

```
10 99 82 83 84 19 29
10 82 99 83 84 19 29
10 82 83 99 84 19 29
10 82 83 84 99 19 29
10 19 82 83 84 99 29
10 19 29 82 83 84 99
```

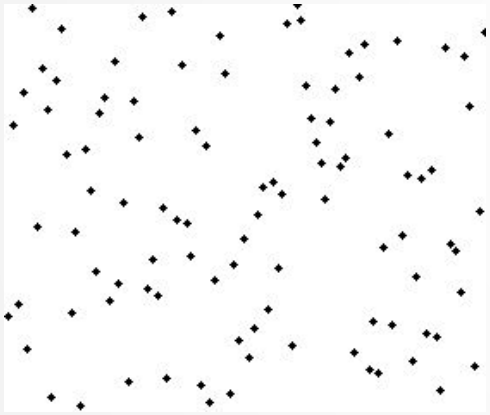
Insertion Sort

```
39 51 33 21 24 27 25 39 56 43 23 40
33 39 51 21 24 27 25 39 56 43 23 40
21 33 39 51 24 27 25 39 56 43 23 40
21 24 33 39 51 27 25 39 56 43 23 40
21 24 27 33 39 51 25 39 56 43 23 40
21 24 25 27 33 39 51 39 56 43 23 40
21 24 25 27 33 39 39 51 56 43 23 40
21 24 25 27 33 39 39 51 56 43 23 40
21 24 25 27 33 39 39 43 51 56 23 40
21 23 24 25 27 33 39 39 43 51 56 40
21 23 24 25 27 33 39 39 40 43 51 56
```

[10 99 82 83 84 19 29]

[51 39 33 21 24 27 25 39 56 43 23 40]

Bubble sort



Bubble Sort

```
10 82 83 84 19 29 99
10 82 83 19 29 84 99
10 82 19 29 83 84 99
10 19 29 82 83 84 99
10 19 29 82 83 84 99
10 19 29 82 83 84 99
```

Bubble Sort

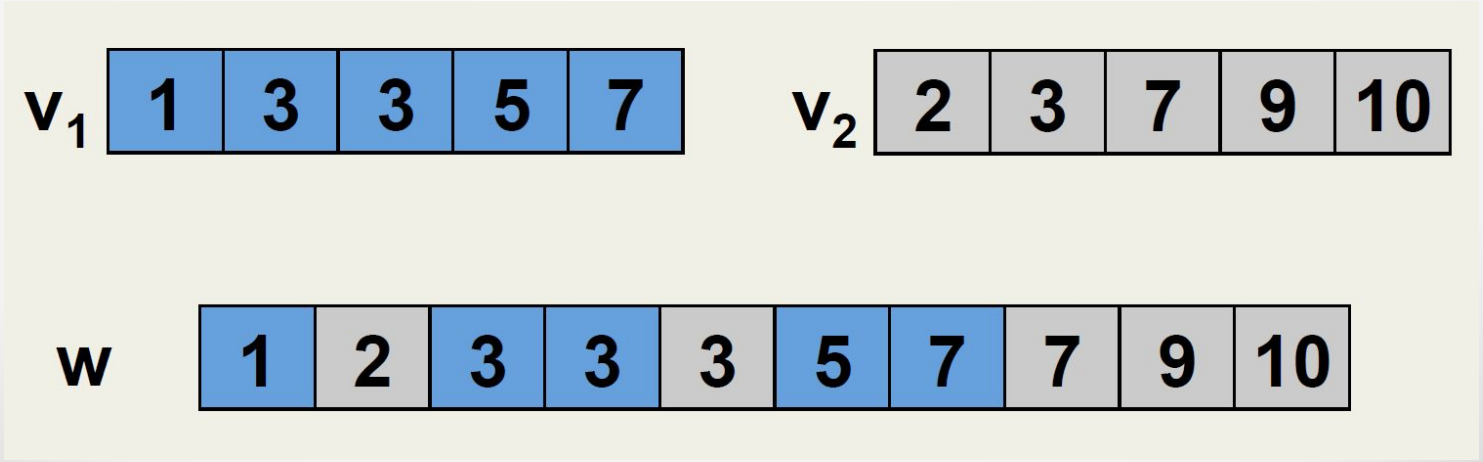
```
39 33 21 24 27 25 39 51 43 23 40 56
33 21 24 27 25 39 39 43 23 40 51 56
21 24 27 25 33 39 39 23 40 43 51 56
21 24 25 27 33 39 23 39 40 43 51 56
21 24 25 27 33 23 39 39 40 43 51 56
21 24 25 27 23 33 39 39 40 43 51 56
21 24 25 23 27 33 39 39 40 43 51 56
21 24 23 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
21 23 24 25 27 33 39 39 40 43 51 56
```

Merge Sort

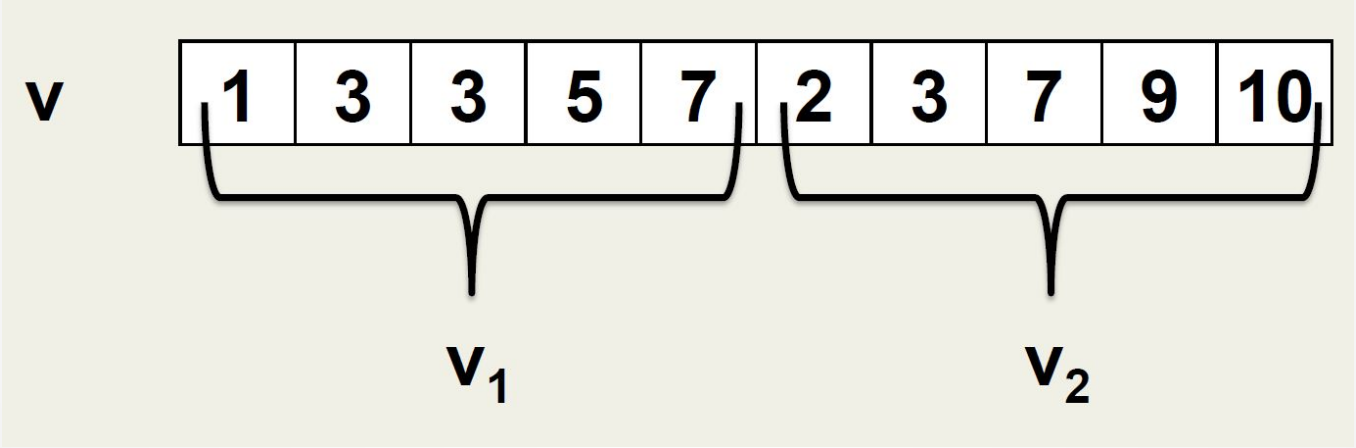
ordenação por intercalação

Idéia Geral

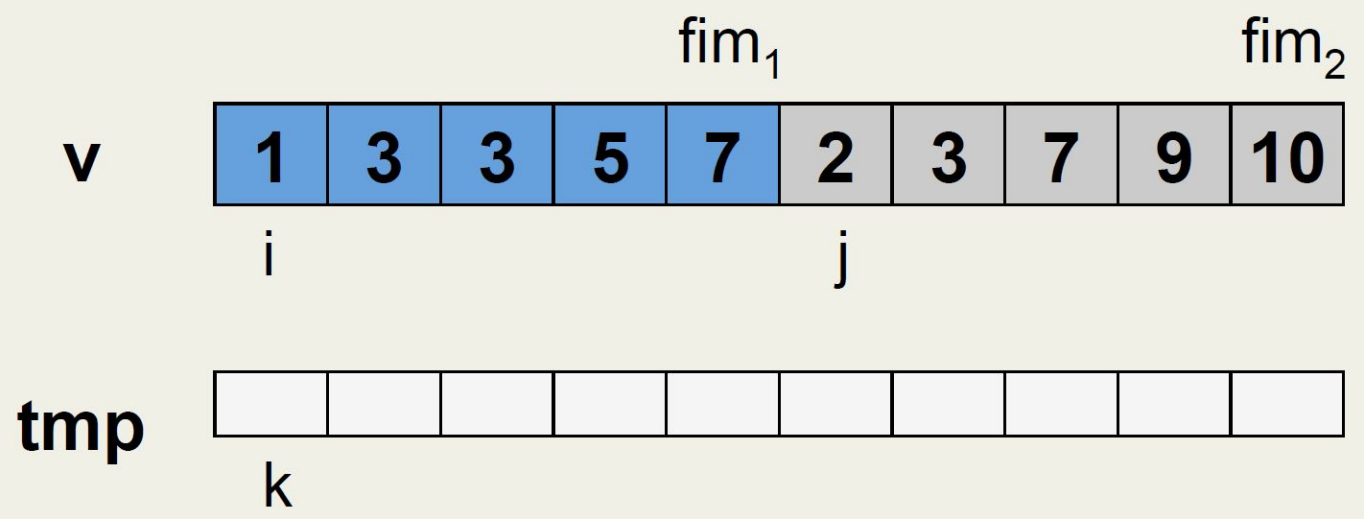
A partir de dois vetores ordenados, posso construir um outro também ordenado



Merge Sort - Ordenação por intercalação

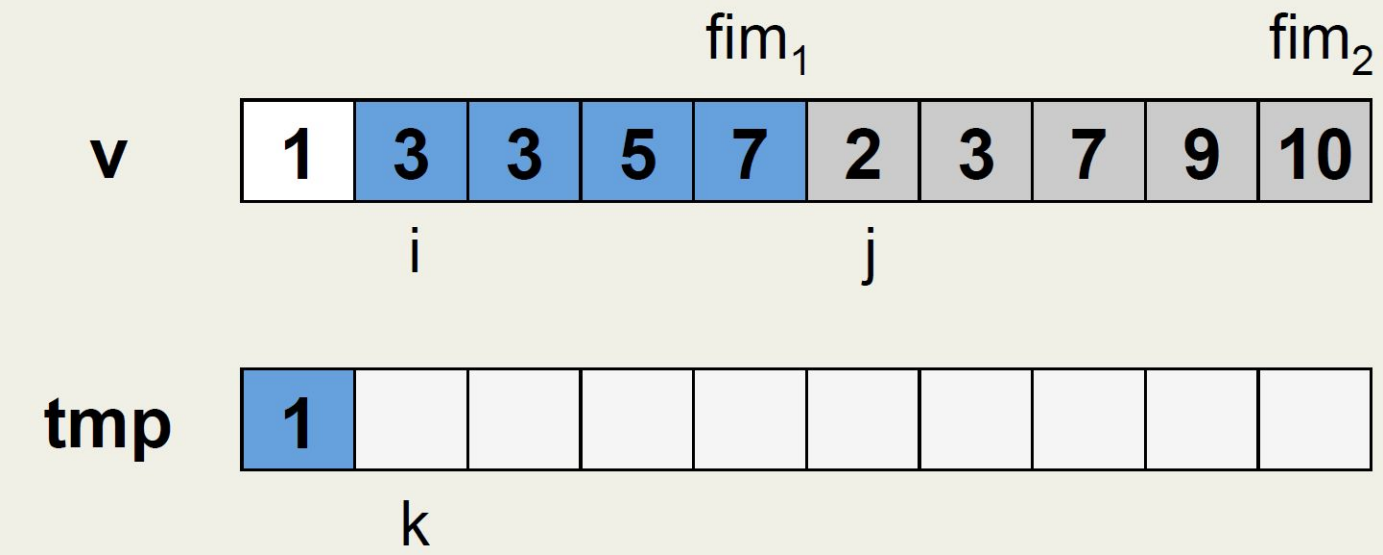


Merge Sort - Ordenação por intercalação



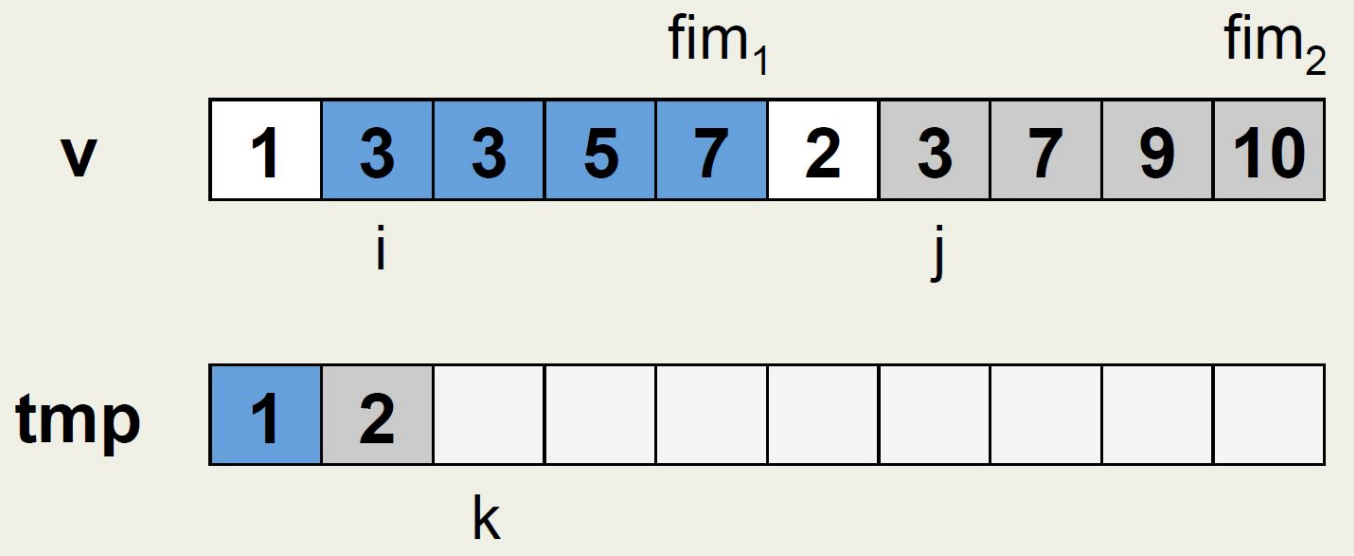
i: elemento atual de v_1
j: elemento atual de v_2
k: 1ª posição livre de tmp

Merge Sort - Ordenação por intercalação



SE v[i] ≤ v[j] ENTÃO :
tmp[k] = v[i], i = i +1, k = k+1

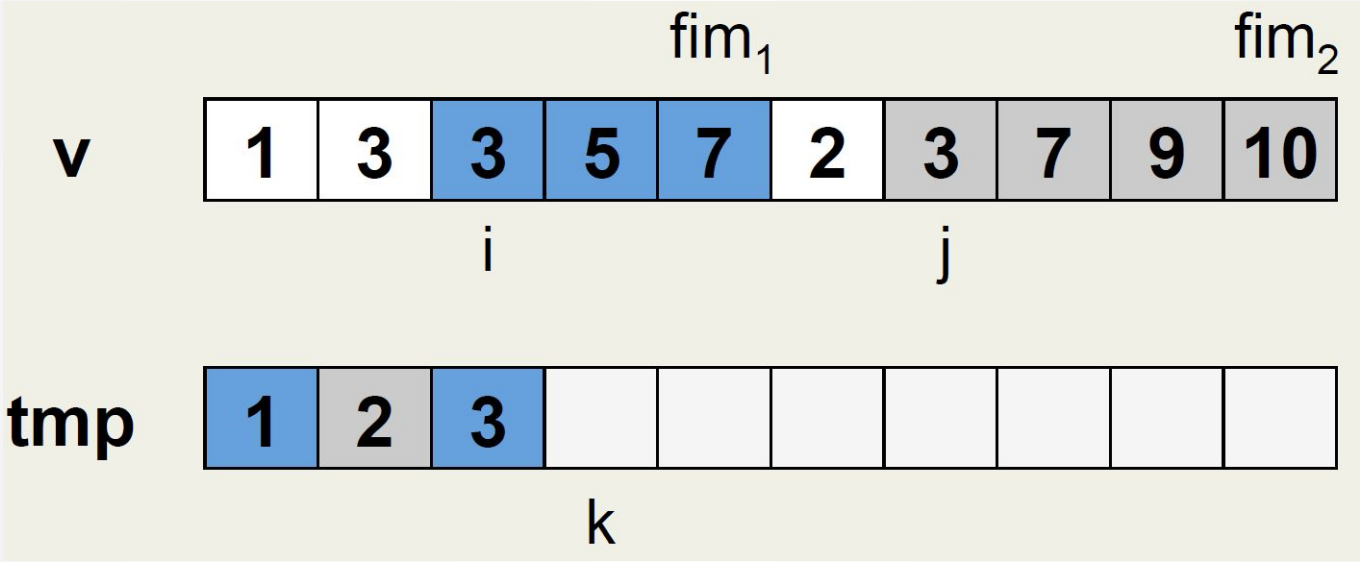
Merge Sort - Ordenação por intercalação



i: elemento atual de v_1
j: elemento atual de v_2
k: 1ª posição livre de tmp

```
SE v[i] ≤ v[j] ENTÃO :
    tmp[k] = v[i], i = i +1, k = k+1
SENÃO :
    tmp[k] = v[j], j = j+1, k = k+1
```

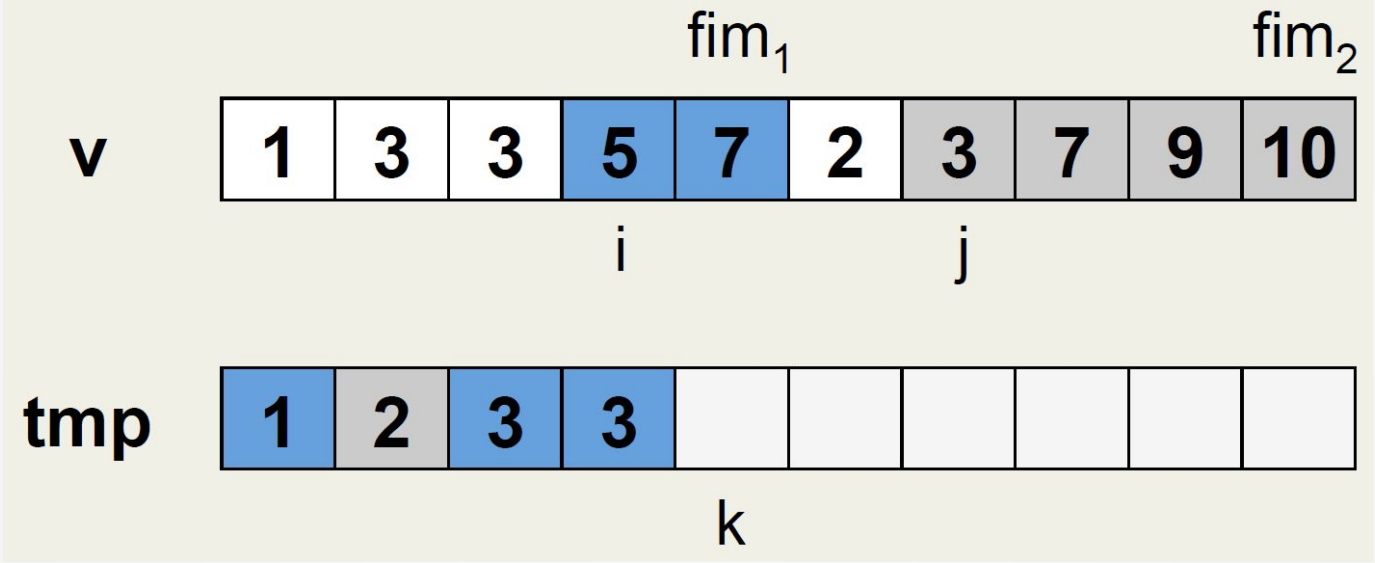
Merge Sort - Ordenação por intercalação



i: elemento atual de v_1
j: elemento atual de v_2
k: 1ª posição livre de tmp

```
SE v[i] ≤ v[j] ENTÃO :
    tmp[k] = v[i], i = i +1, k = k+1
SENÃO :
    tmp[k] = v[j], j = j+1, k = k+1
```

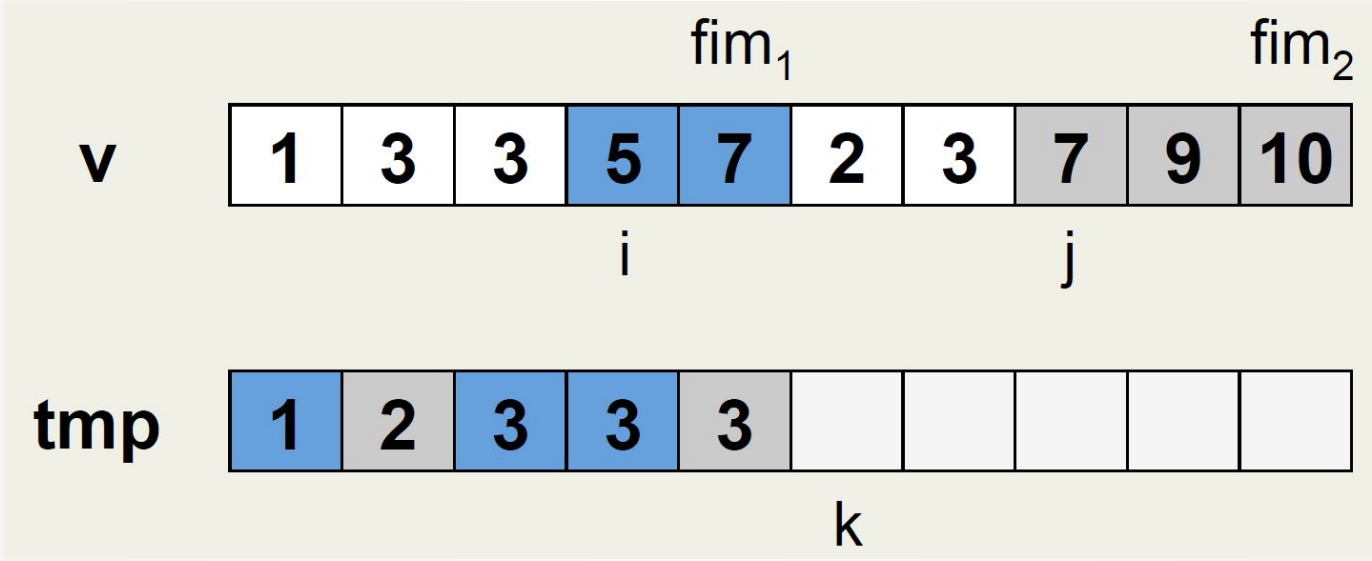
Merge Sort - Ordenação por intercalação



i: elemento atual de v_1
j: elemento atual de v_2
k: 1ª posição livre de tmp

```
SE v[i] ≤ v[j] ENTÃO :
    tmp[k] = v[i], i = i +1, k = k+1
SENÃO :
    tmp[k] = v[j], j = j+1, k = k+1
```

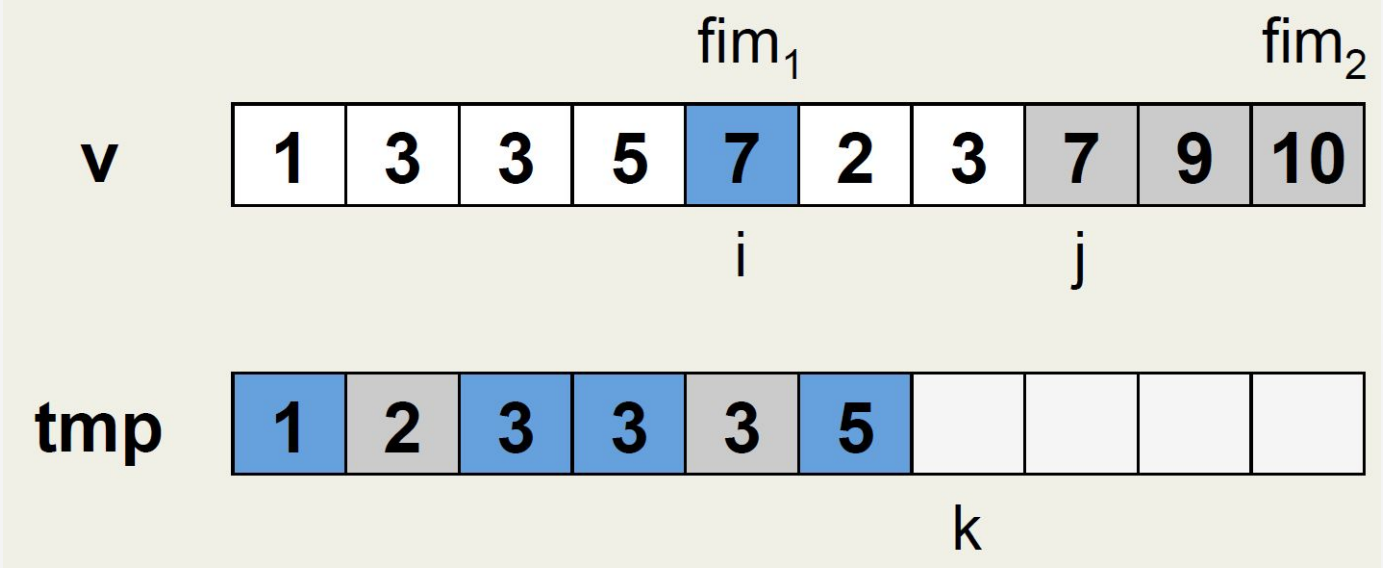
Merge Sort - Ordenação por intercalação



i: elemento atual de v_1
j: elemento atual de v_2
k: 1ª posição livre de tmp

```
SE v[i] ≤ v[j] ENTÃO :
    tmp[k] = v[i], i = i +1, k = k+1
SENÃO :
    tmp[k] = v[j], j = j+1, k = k+1
```

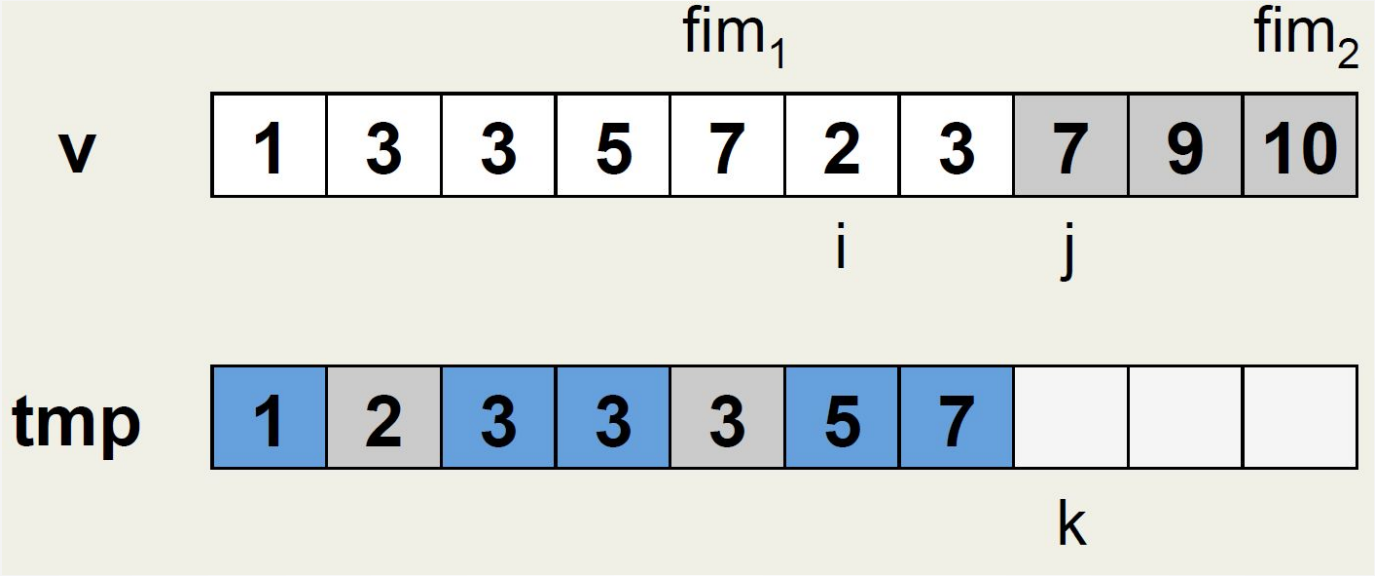
Merge Sort - Ordenação por intercalação



i: elemento atual de v_1
j: elemento atual de v_2
k: 1ª posição livre de tmp

```
SE v[i] ≤ v[j] ENTÃO :
    tmp[k] = v[i], i = i +1, k = k+1
SENÃO :
    tmp[k] = v[j], j = j+1, k = k+1
```

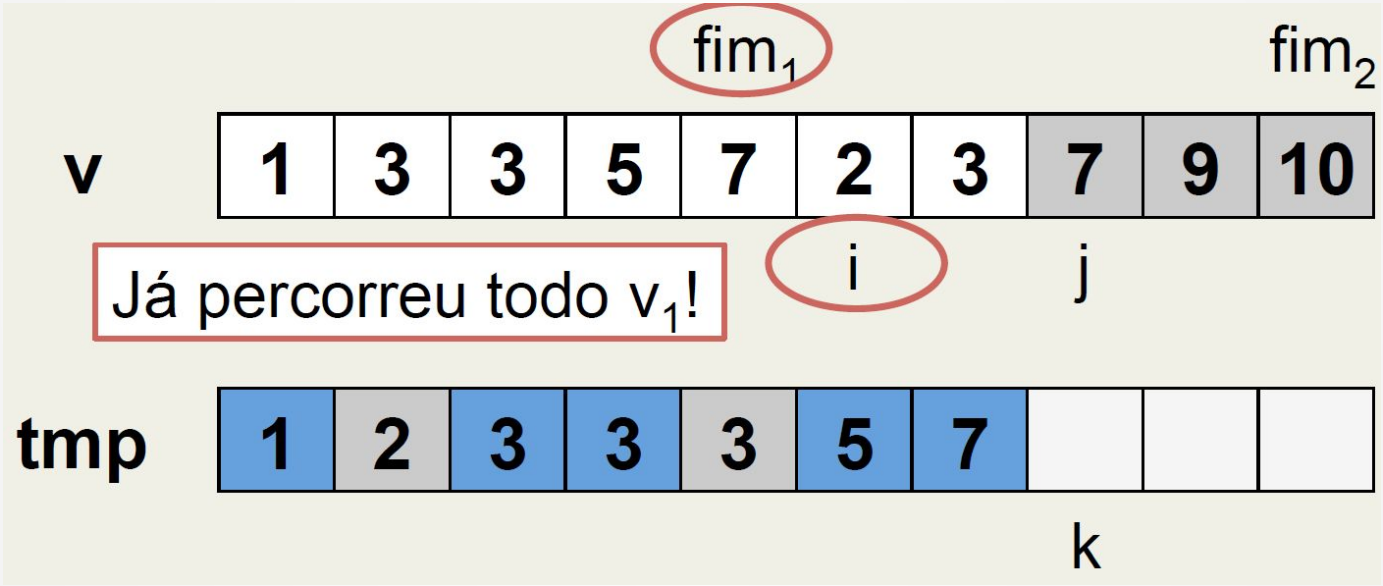
Merge Sort - Ordenação por intercalação



i: elemento atual de v_1
j: elemento atual de v_2
k: 1ª posição livre de tmp

```
SE v[i] ≤ v[j] ENTÃO :
    tmp[k] = v[i], i = i +1, k = k+1
SENÃO :
    tmp[k] = v[j], j = j+1, k = k+1
```

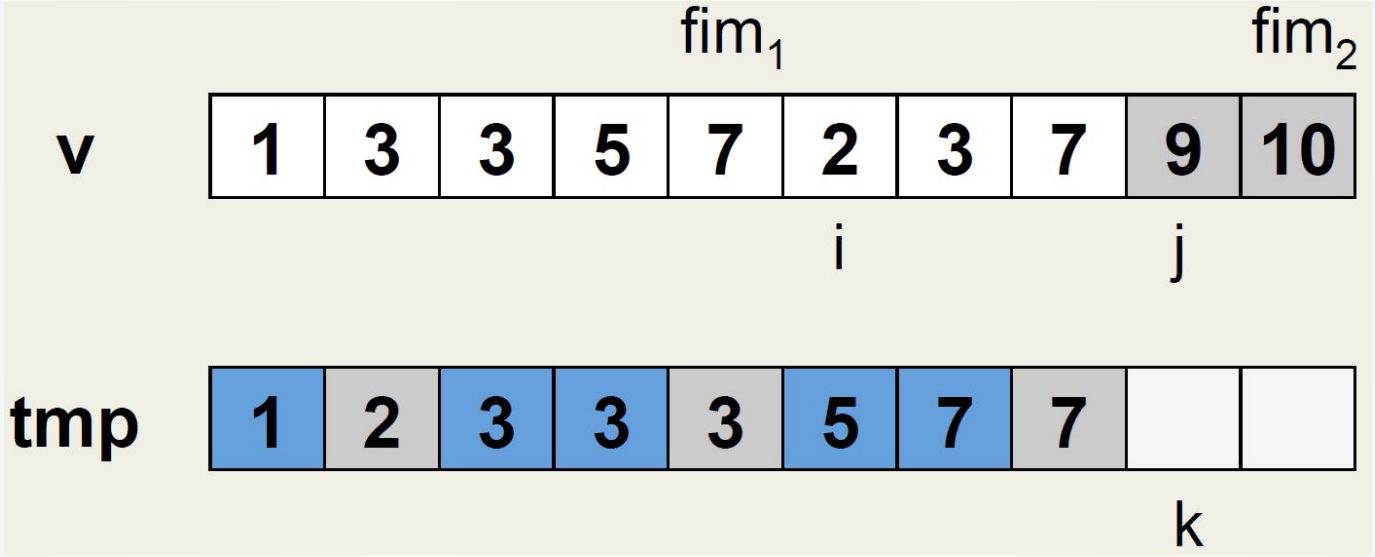
Merge Sort - Ordenação por intercalação



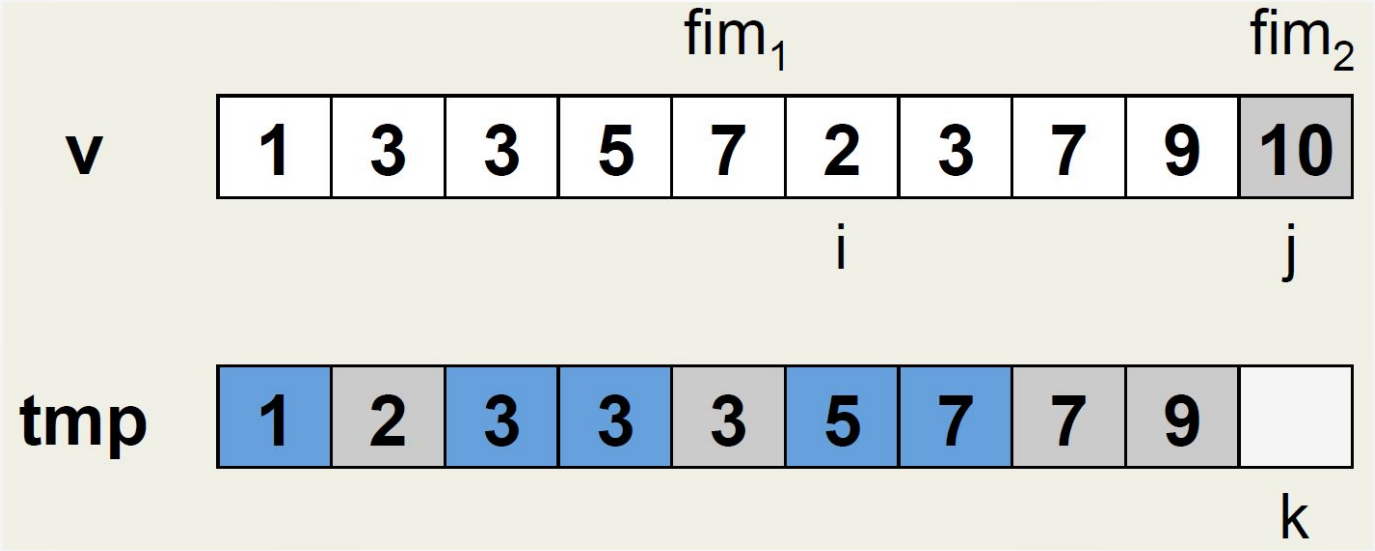

```
SE v[i] ≤ v[j] ENTÃO :
    tmp[k] = v[i], i = i +1, k = k+1
SENÃO :
    tmp[k] = v[j], j = j+1, k = k+1
```

```
ENQUANTO j ≤ fim2 FAÇA:
    tmp[k] = v[j]
    k = k+1, j = j+1
FIM_ENQUANTO
```

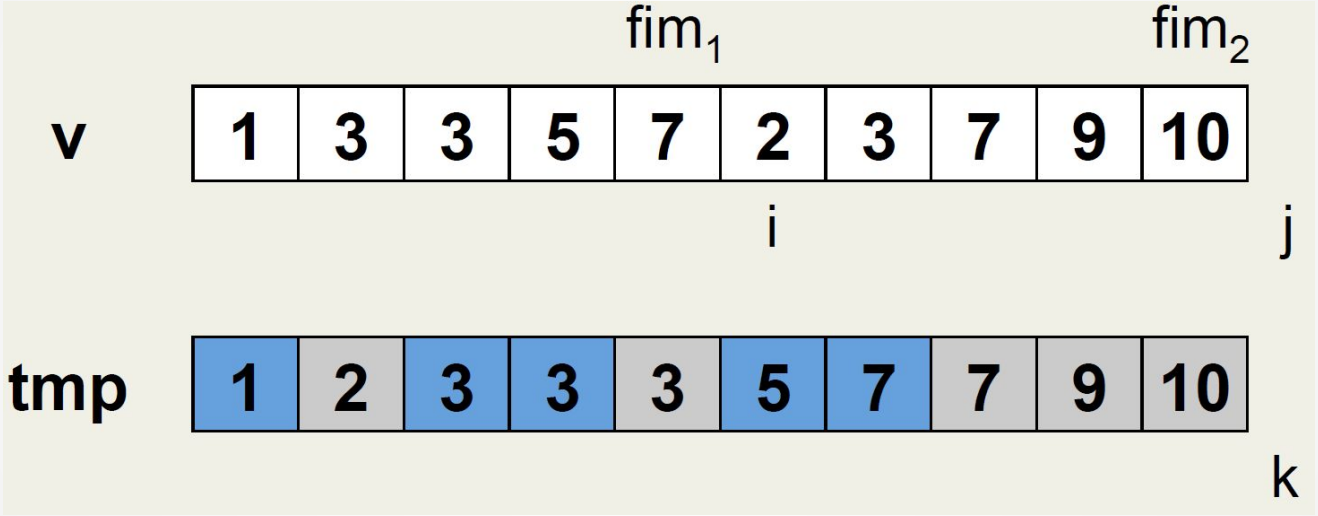
Merge Sort - Ordenação por intercalação



Merge Sort - Ordenação por intercalação



Merge Sort - Ordenação por intercalação

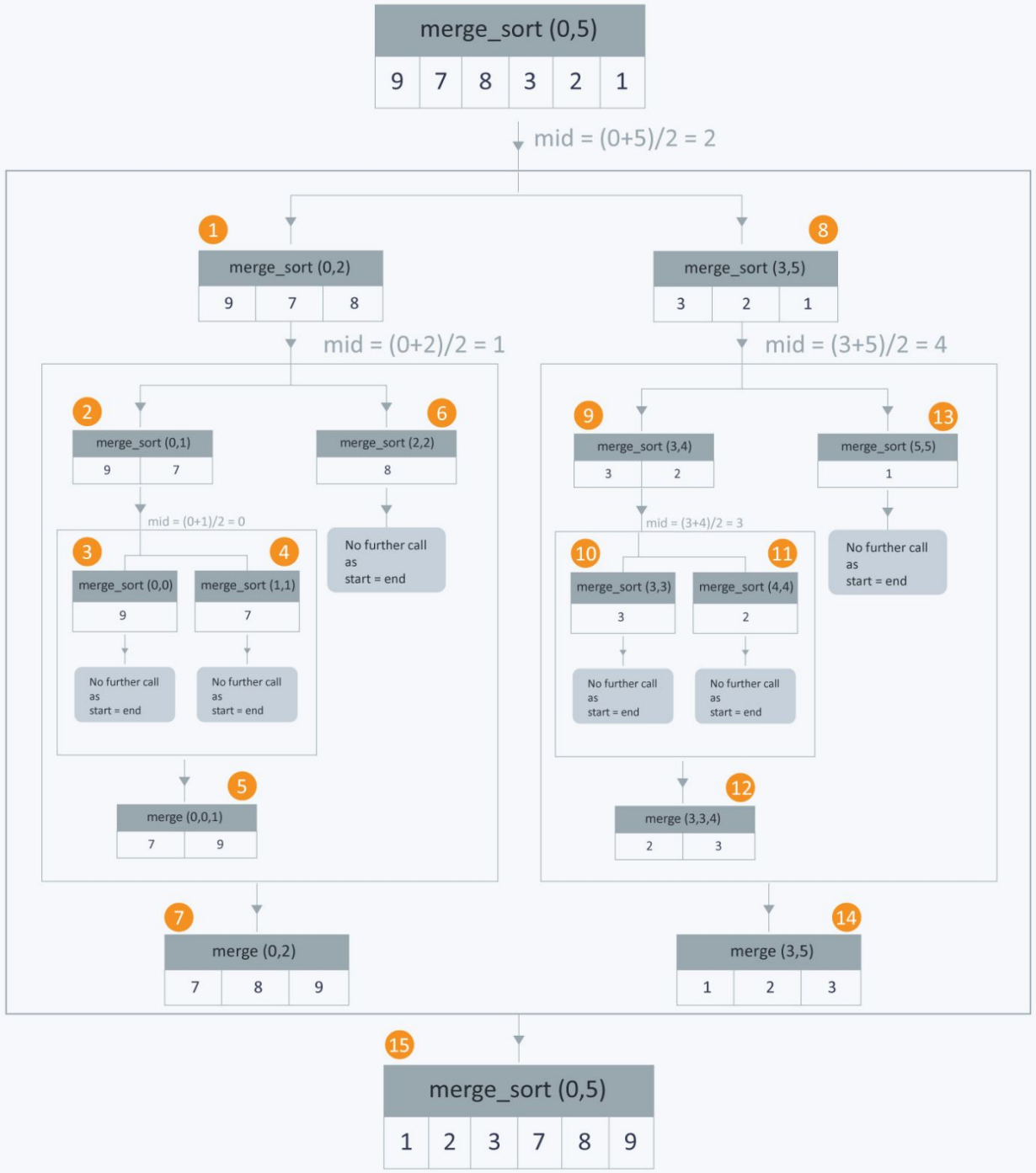


```c
void merge(int *aux, int *v, int inicio, int meio, int fim){
    int i, j, k;
    i = inicio;
    j = meio + 1;
    k = inicio;
    while(i <= meio && j <= fim){
        if(v[i] < v[j]){
            aux[k] = v[i];
            i++;
        }
        else{
            aux[k] = v[j];
            j++;
        }
        k++;
    }

    while(i <= meio){
        aux[k] = v[i];
        i++;
        k++;
    }

    while(j <= fim){
        aux[k] = v[j];
        j++;
        k++;
    }
    //Copia os elementos que foram ordenados para o auxiliar
    for(int p = inicio; p <= fim; p++)
        v[p] = aux[p];
}
```

Intercalar ok, mas e ordenar?

A ideia do Merge Sort é dividir o vetor em dois subvetores, cada um com metade dos elementos do vetor original. Esse procedimento é então reaplicado aos dois subvetores recursivamente.

Quando os subvetores têm apenas um elemento (caso base), a recursão para. Então, os subvetores ordenados são fundidos (ou intercalados) num único vetor ordenado.



```c
void merge(int *aux, int *v, int inicio, int meio, int fim){
    int i, j, k;
    i = inicio;
    j = meio + 1;
    k = inicio;
    while(i <= meio && j <= fim){
        if(v[i] < v[j]){
            aux[k] = v[i];
            i++;
        }
        else{
            aux[k] = v[j];
            j++;
        }
        k++;
    }

    while(i <= meio){
        aux[k] = v[i];
        i++;
        k++;
    }

    while(j <= fim){
        aux[k] = v[j];
        j++;
        k++;
    }
    //Copia os elementos que foram ordenados para o auxiliar
    for(int p = inicio; p <= fim; p++)
        v[p] = aux[p];
}

void mergeSort(int *aux, int *v, int inicio, int fim){
    if(inicio < fim){
        int meio = (inicio + fim) / 2;
        mergeSort(aux, v, inicio, meio);
        mergeSort(aux, v, meio + 1, fim);
        merge(aux, v, inicio, meio, fim);
    }
}
```

tinyurl.com/ycyczm8f

Complexidade

$T(n) = 2*T(n/2) + n$

Pior e melhor caso: $O(n*\log_2 n)$