# Algoritmo MergeSort

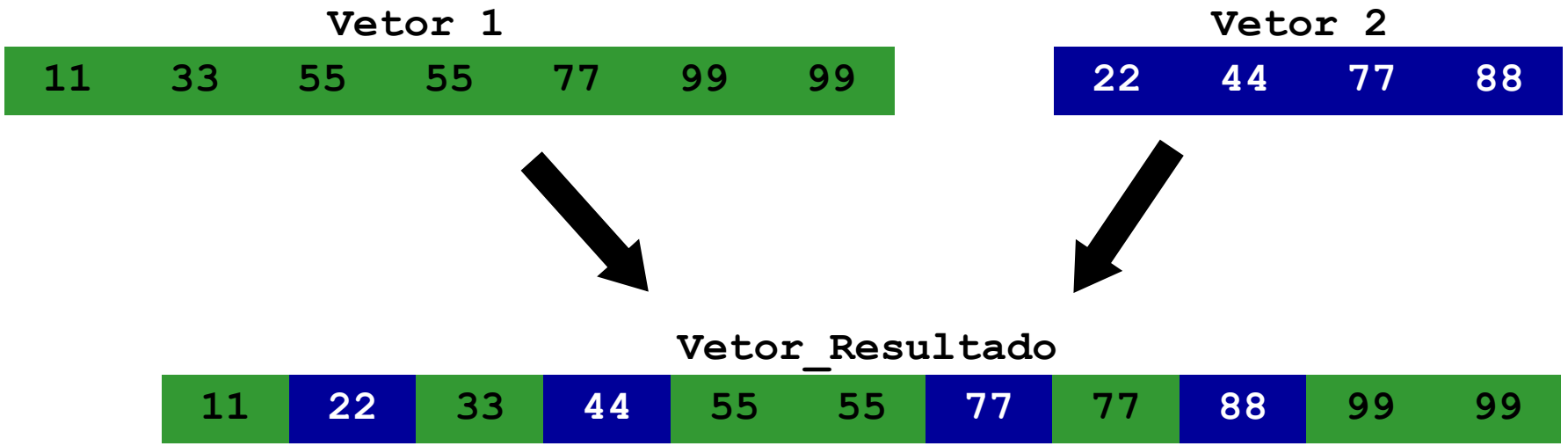
Estrutura de Dados II
Prof Jairo Francisco de Souza

# Intercalação

- Generalidades
  - Intercalação é o processo através do qual diversos arquivos seqüenciais classificados por um mesmo critério são mesclados gerando um único arquivo seqüencial
- Algoritmo Básico
  - De cada um dos arquivos a intercalar basta ter em memória um registro. Considera-se cada arquivo como uma pilha, e o registro em memória seu topo
  - 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 a high value
  - O algoritmo termina quando todos os topos da pilha tiverem high value

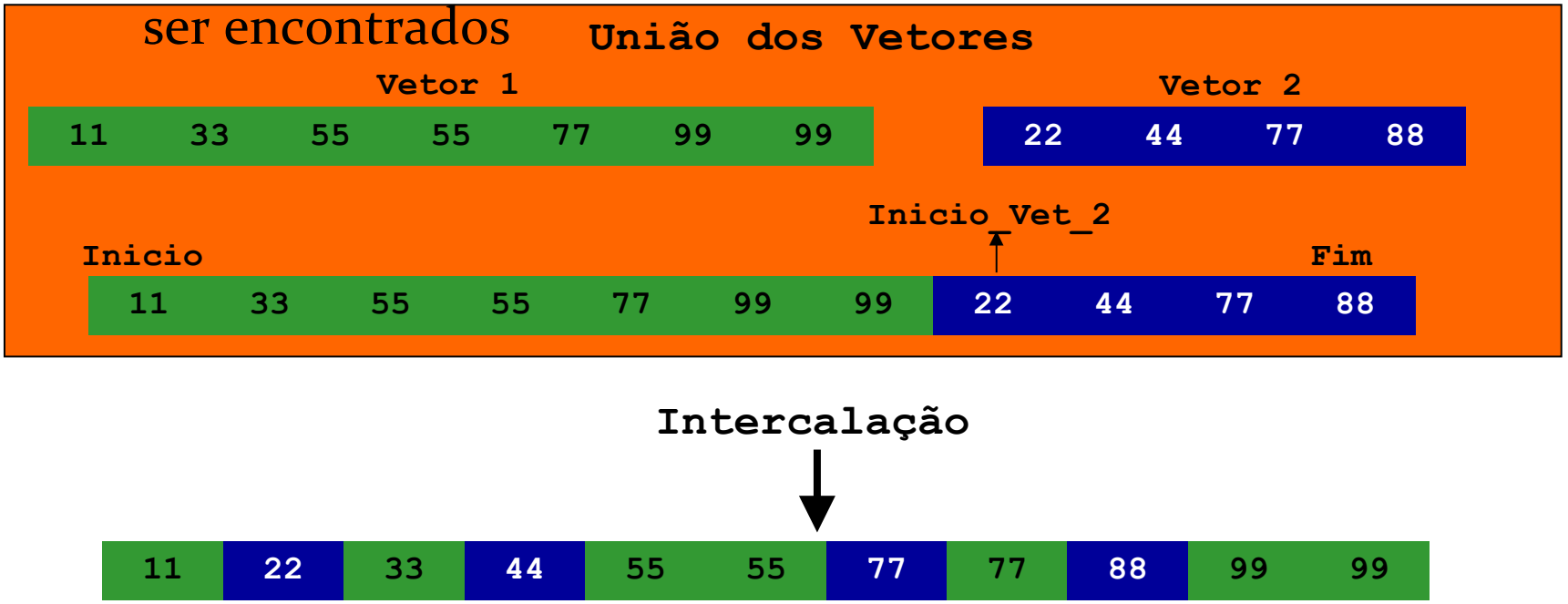
# Intercalação

- Outra forma de intercalação é mesclar dois vetores (ordenados previamente)
- O resultado é um vetor ordenado, com os elementos dos vetores usados na mesclagem



# Intercalação

- Inicialmente, para que aconteça a intercalação (ou ***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



# Intercalação

- A intercalação deve ser usada também quando há necessidade de unir dados de dois arquivos de dados
- Desta maneira, 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
- Neste caso, o desenvolvedor deve se preocupar principalmente com a coincidência de chaves primárias, e realizar o tratamento das transações problemáticas

# Algoritmo de Intercalação

```
intercala (inteiro inicio_vet, inteiro inicio_vet_2, inteiro fim_vet, inteiro vetor[])
{
   inteiro indice_inicio, indice_inicio_vet_2, indice_aux;
   inteiro *vetor_aux;
   vetor_aux = (inteiro *) aloca_memoria (fim_vet * sizeof (inteiro));
   indice_inicio = inicio_vet;
   indice_inicio_vet_2 = inicio_vet_2;
   indice_aux = 0;
   Enquanto (indice_inicio < inicio_vet_2 e indice_inicio_vet_2  < fim_vet)
   {
      Se (vetor[indice_inicio] <= vetor[indice_inicio_vet_2])
      {
         vetor_aux [indice_aux] = vetor[indice_inicio];
         indice_aux = indice_aux + 1;
         indice_inicio = indice_inicio + 1;
      } senão {
         vetor_aux [indice_aux] = vetor[indice_inicio_vet_2];
         indice_aux = indice_aux + 1;
         indice_inicio_vet_2 = indice_inicio_vet_2 + 1;
      }
   }

   Enquanto (indice_inicio < inicio_vet_2) {
         vetor_aux [indice_aux] = vetor[indice_inicio];
         indice_aux = indice_aux + 1;
         indice_inicio = indice_inicio + 1;
   }

   Enquanto (indice_inicio_vet_2 < fim_vet)  {
         vetor_aux [indice_aux] = vetor[indice_inicio_vet_2];
         indice_aux = indice_aux + 1;
         indice_inicio_vet_2 = indice_inicio_vet_2 + 1;
   }

   Para (indice_inicio = inicio_vet; indice_inicio < fim_vet; indice_inicio++)  {
          vetor[indice_inicio] = vetor_aux [indice_inicio - inicio_vet];
   }

   liberar_memoria (vetor_aux);
}
```

# Intercalação usada para ordenação

- Algoritmo MergeSort utiliza a idéia de intercalação para ordenar registros.
- Algoritmo criado por von Neumann
- Complexidade O(NlogN) 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

# MergeSort

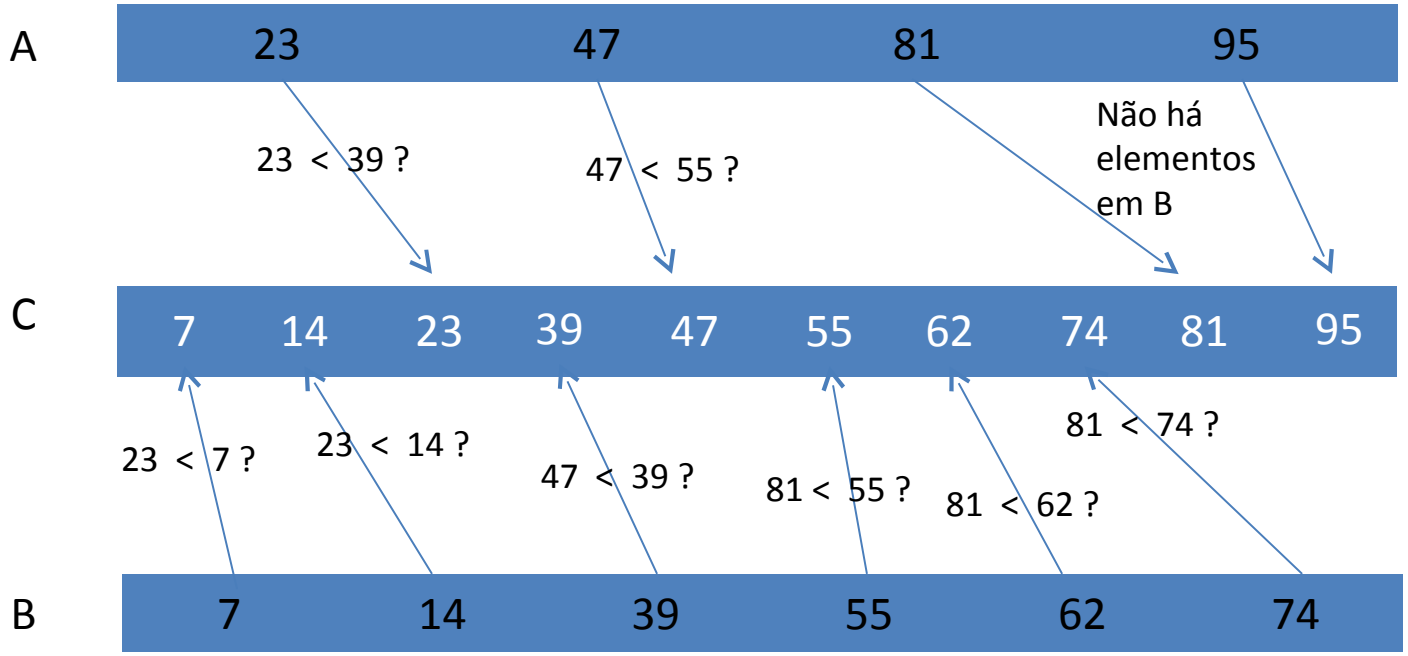
A idéia central é unir dois arrays que já estejam ordenados. Ou seja, unir dois arrays A e B já ordenados e criar um terceiro array C que contenha os elementos de A e B já ordenados na ordem correta.

O foco inicial é no processo de junção dos arrays e posteriormente vamos trabalhar o processo de ordenação

# Cenário

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

# Cenário


A
23
47
81
95
23 < 39 ?
47 < 55 ?
Não há elementos em B
C
7
14
23
39
47
55
62
74
81
95
23 < 7 ?
23 < 14 ?
47 < 39 ?
81 < 55 ?
81 < 62 ?
81 < 74 ?
B
7
14
39
55
62
74

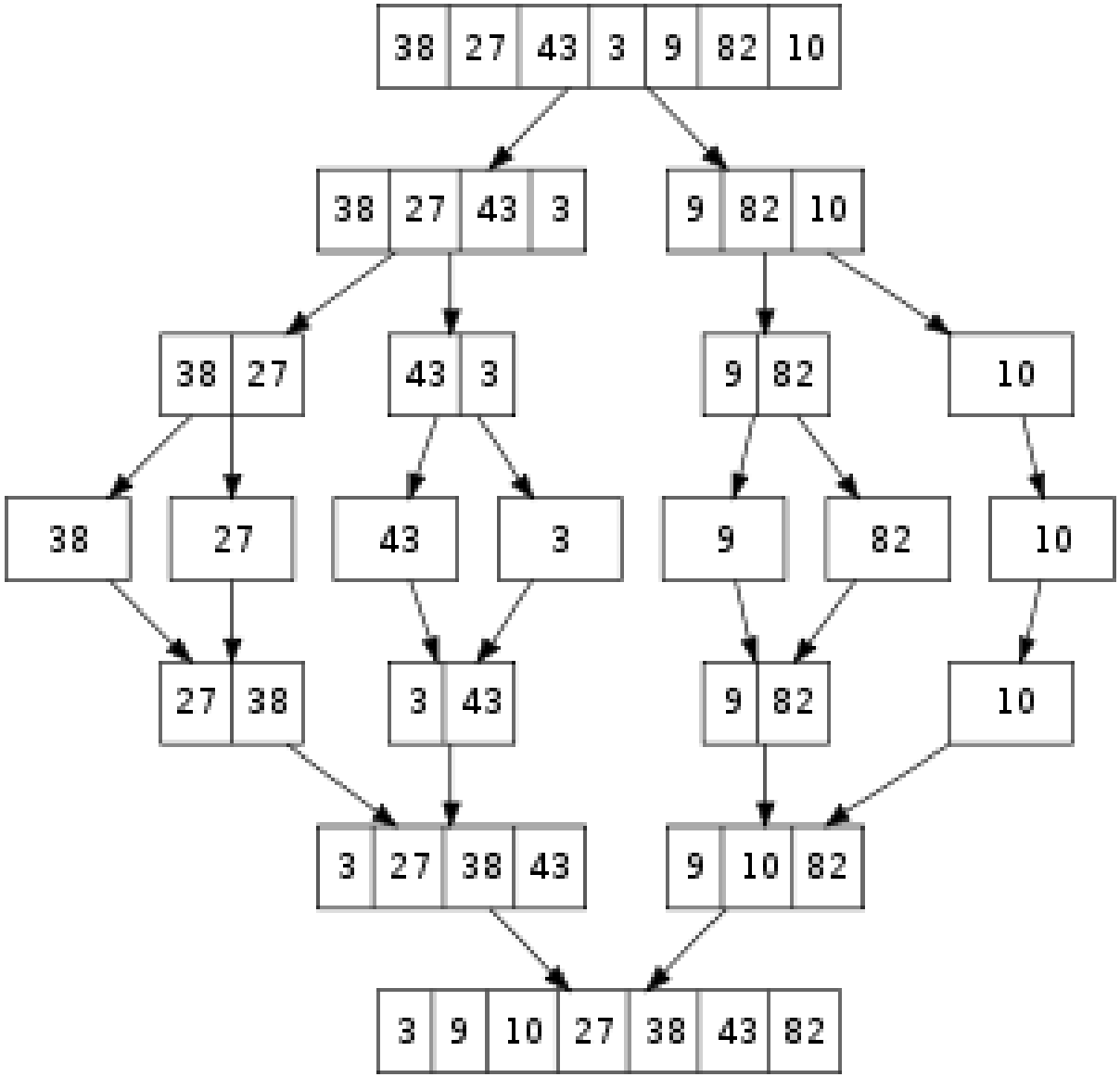
# Ordenação

A idéia 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?

RECURSIVIDADE!

# Ordenação


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

# O algoritmo MergeSort

- O algoritmo divide a sequência original em pares de dados, classificando e intercalando os elementos das partições
- Três passos são fundamentais para a execução do ***mergesort***:
  - Dividir: Dividir os dados em subsequências pequenas;
  - Conquistar: Classificar as duas metades recursivamente aplicando o ***mergesort***;
  - Combinar: Juntar as duas metades em um único conjunto já classificado

# O algoritmo MergeSort

- Dividir para conquistar
- O algoritmo é executado de forma recursiva
- Primeiro ordenar a primeira metade, em seguida a segunda metade

```
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);
    }
}
```

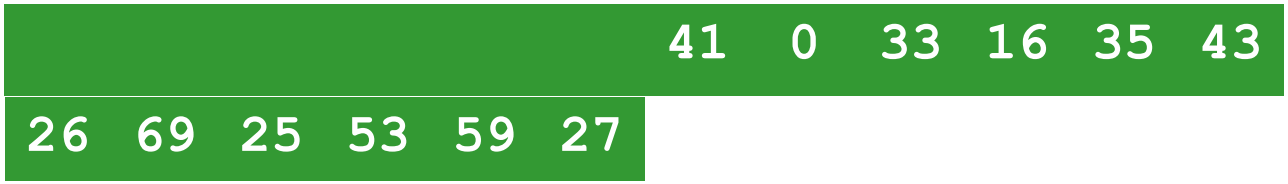
# O algoritmo MergeSort

- Considere o vetor abaixo:

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

# O algoritmo MergeSort

- Execução:

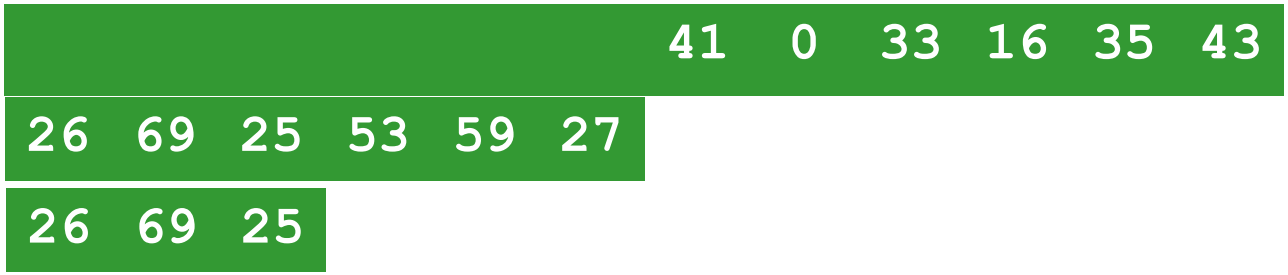



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

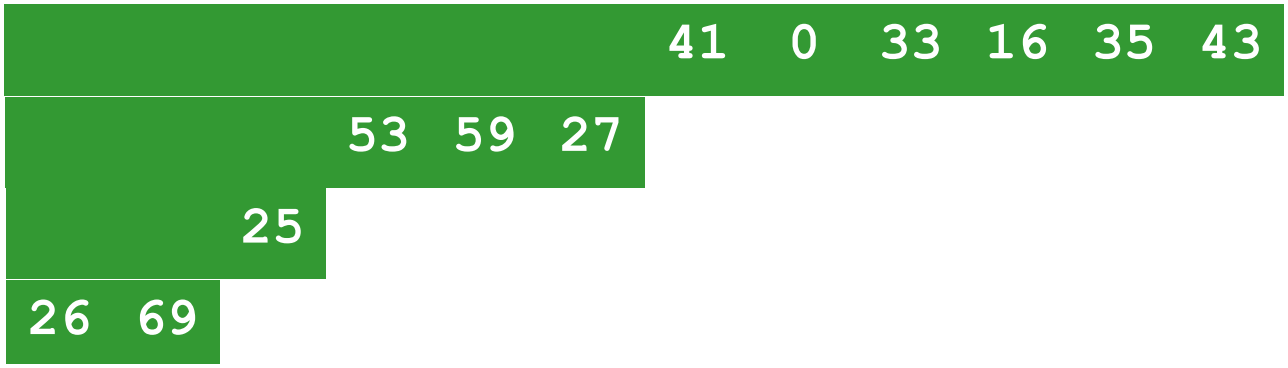



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

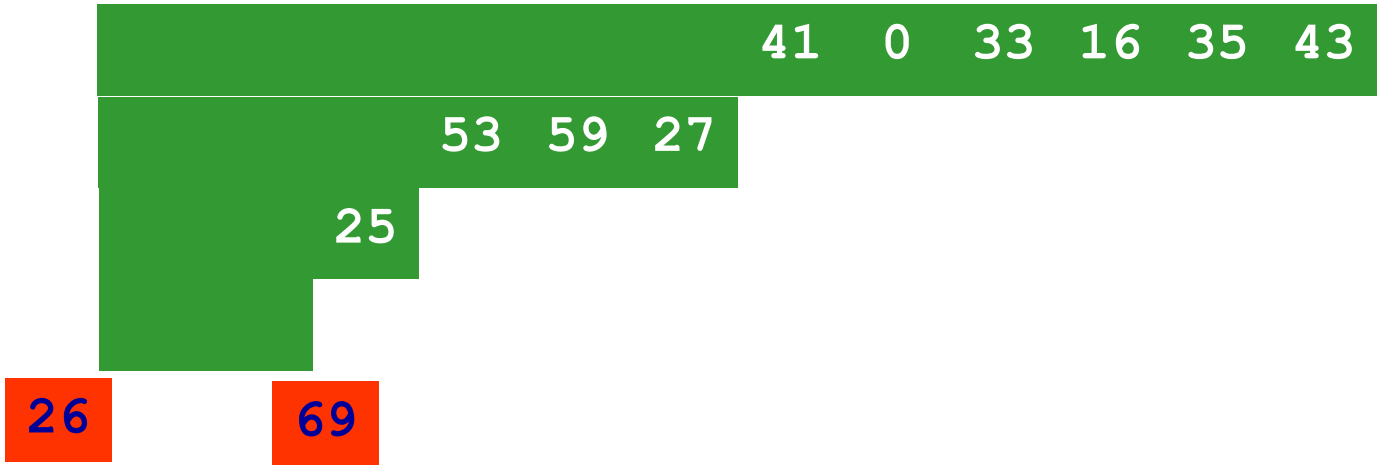



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

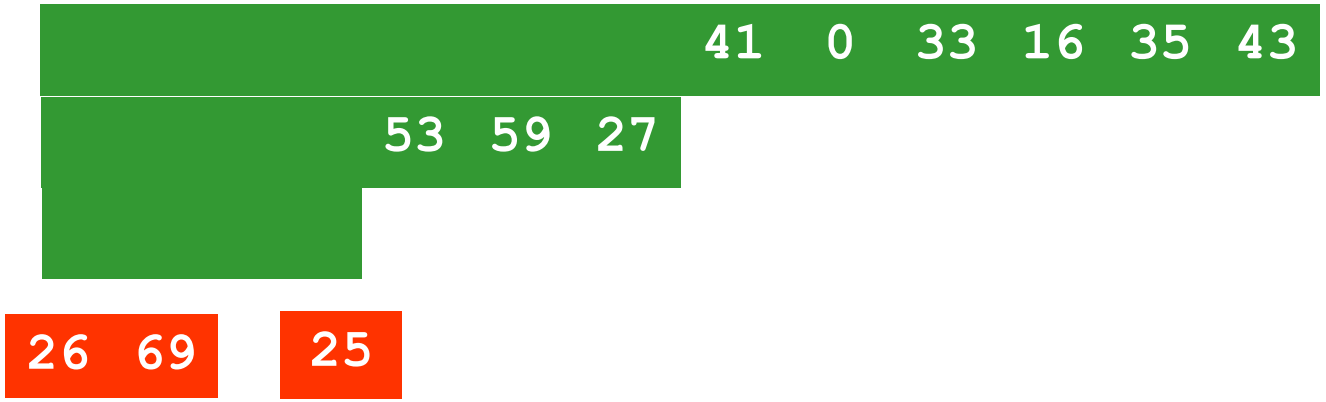



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

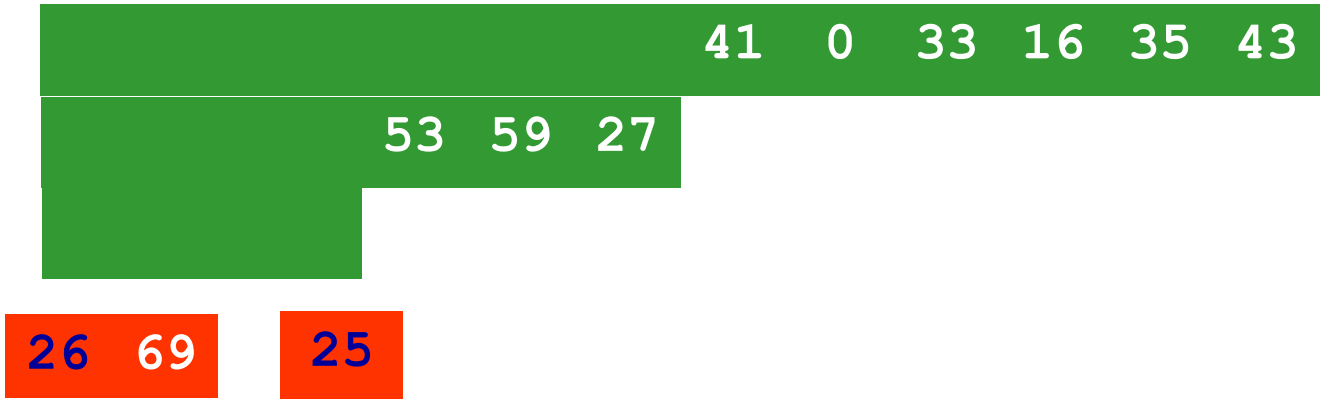



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

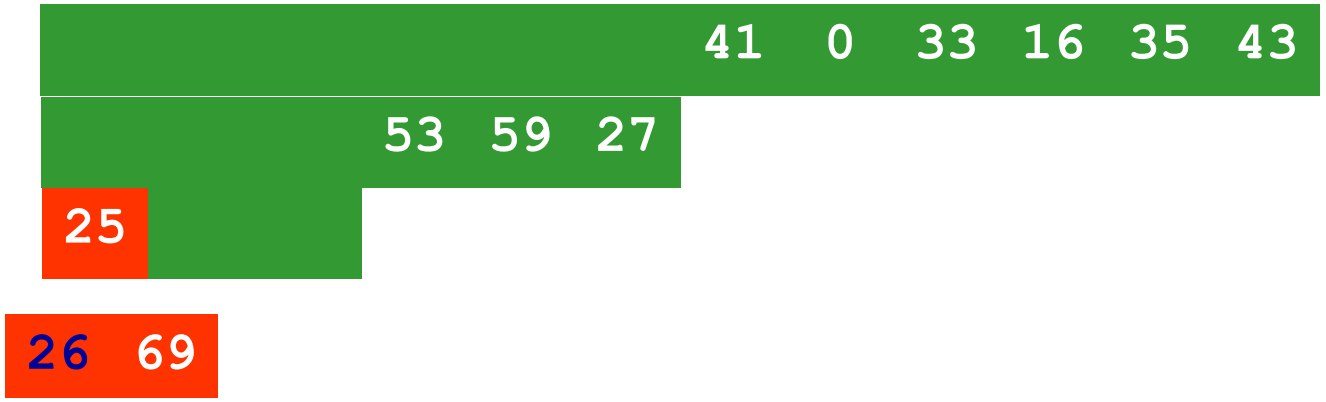



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

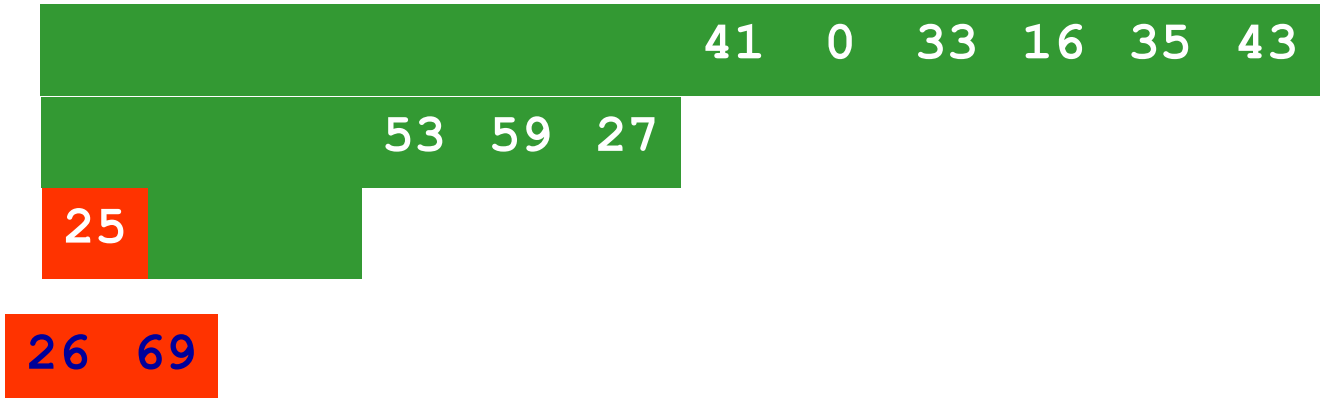



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

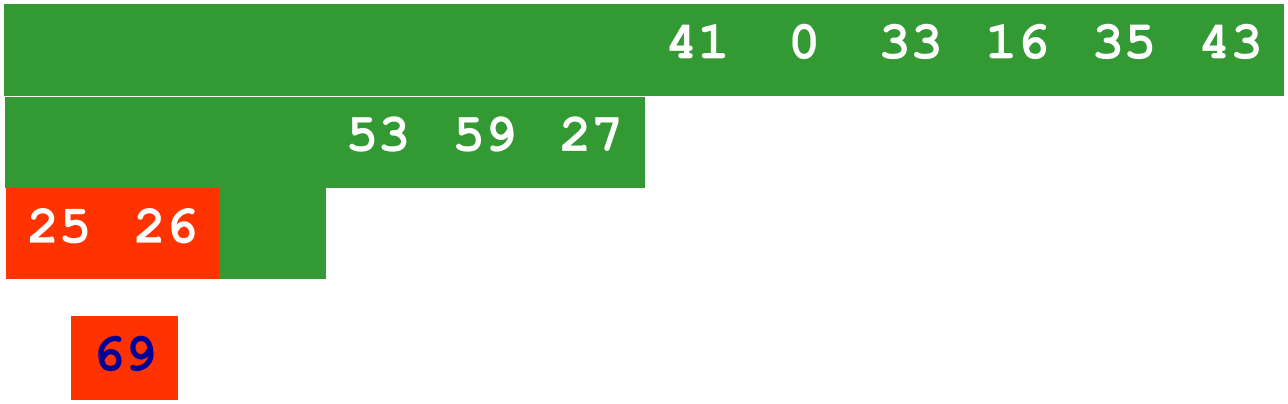



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

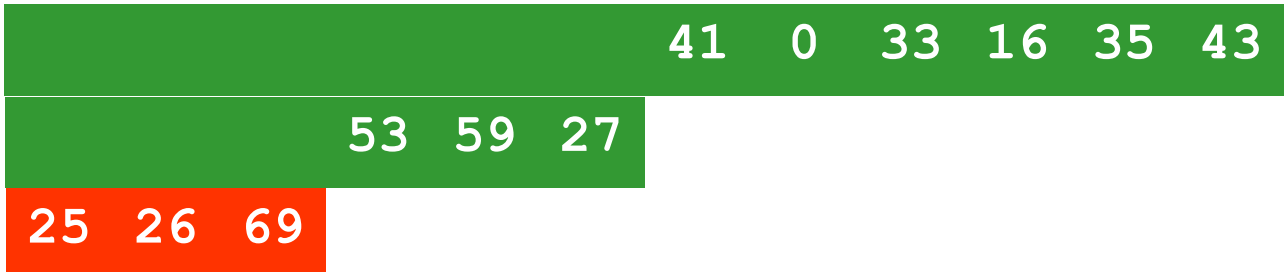



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

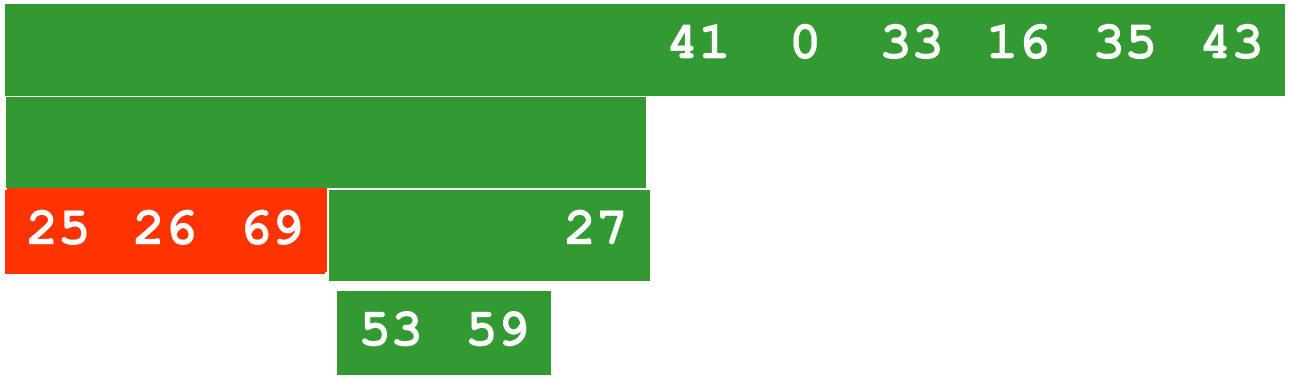



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

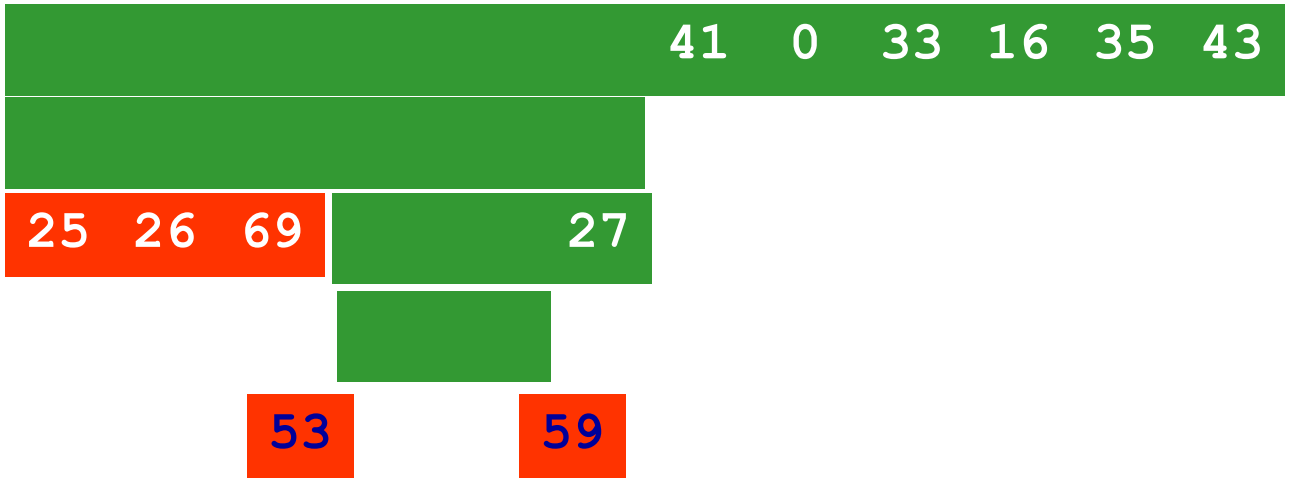



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

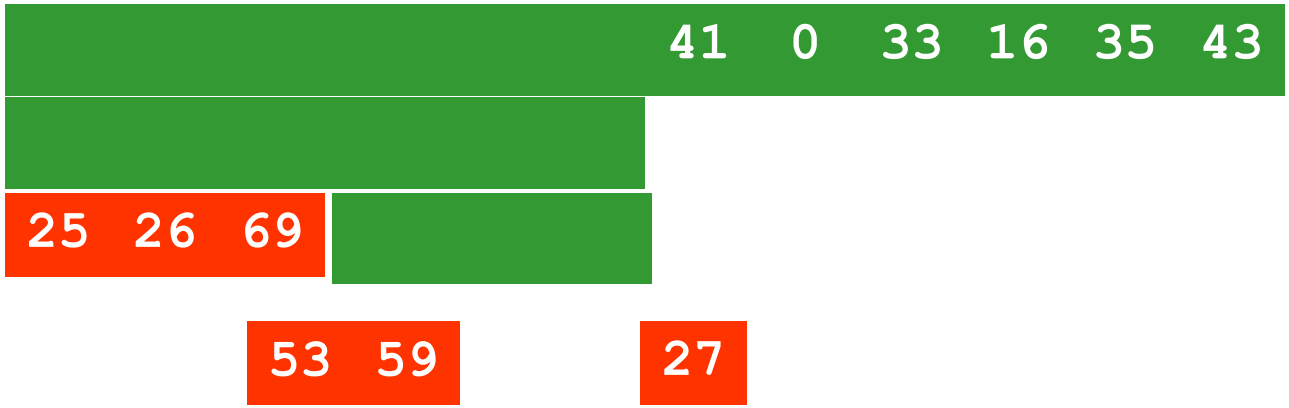



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

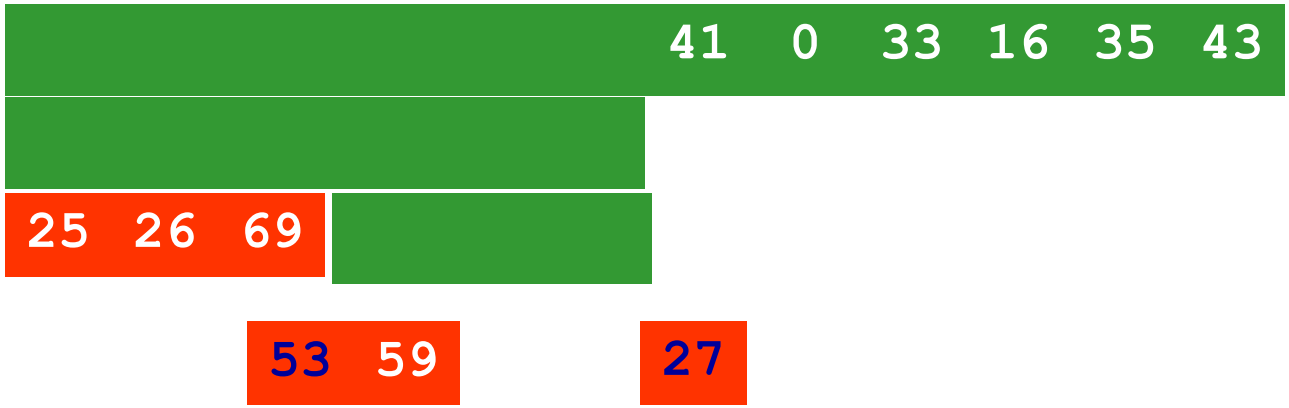



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

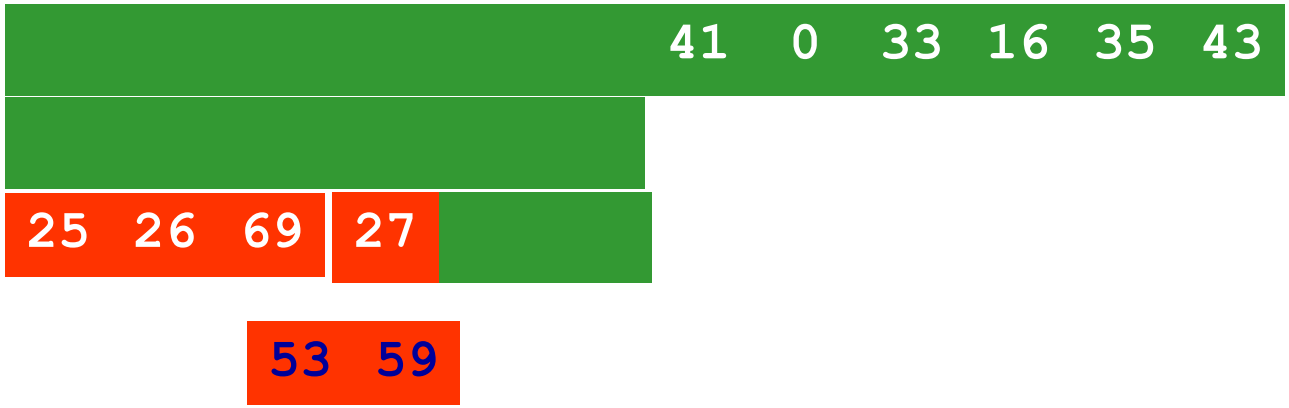



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

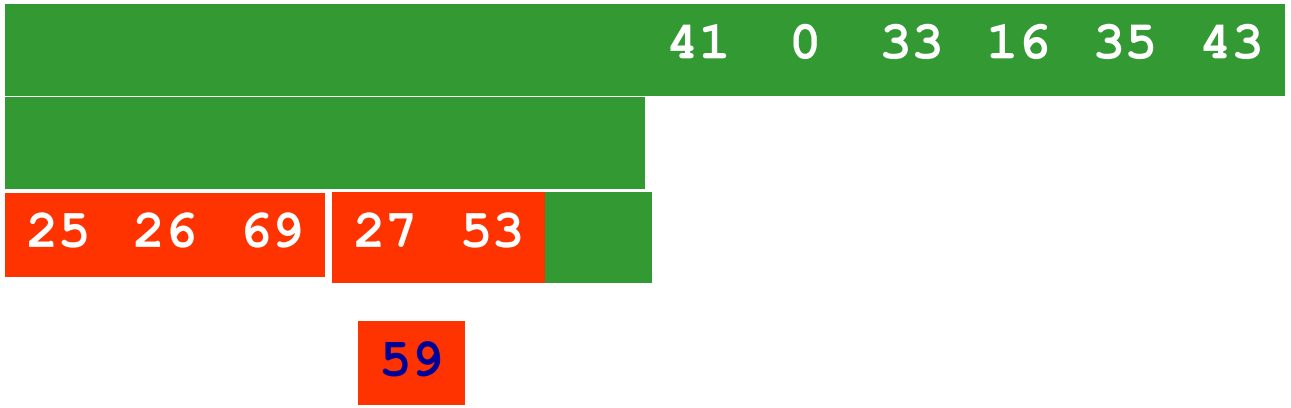



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

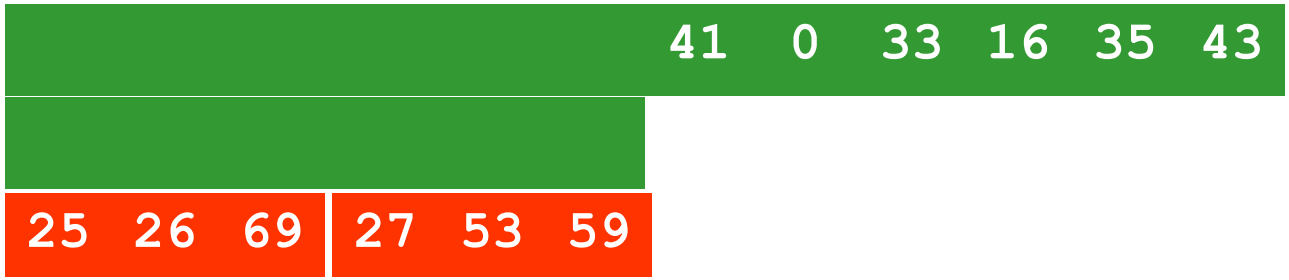



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

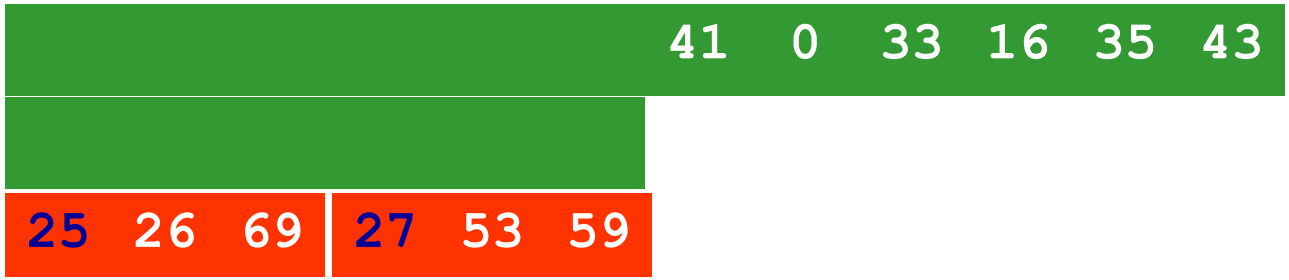



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

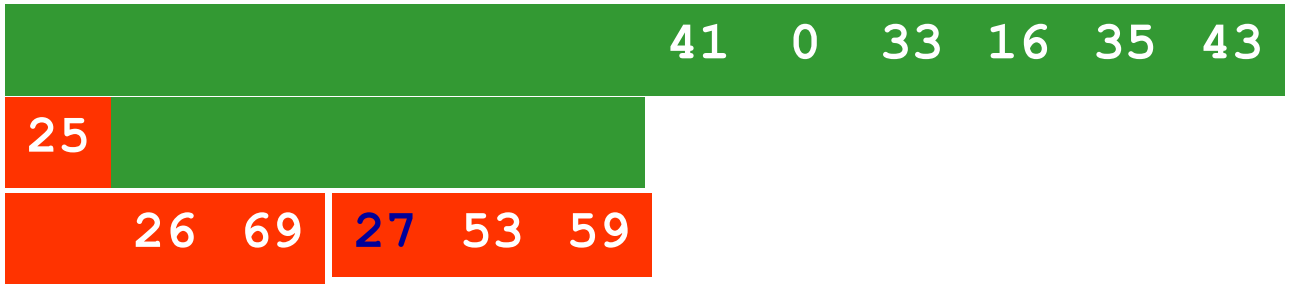



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

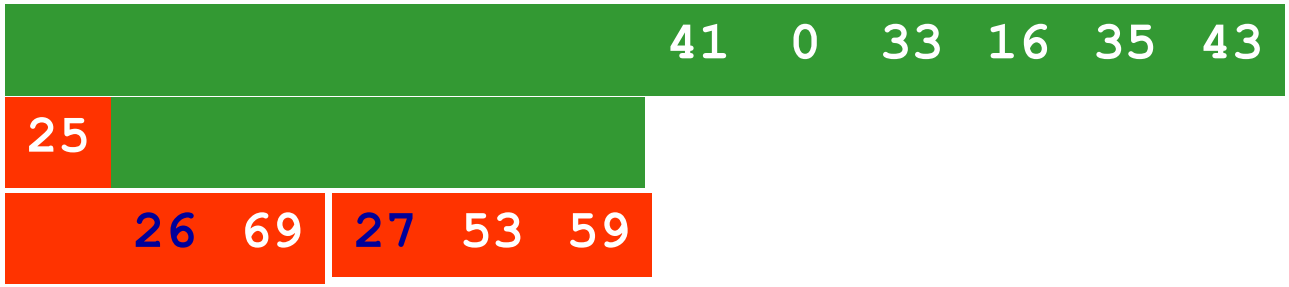



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

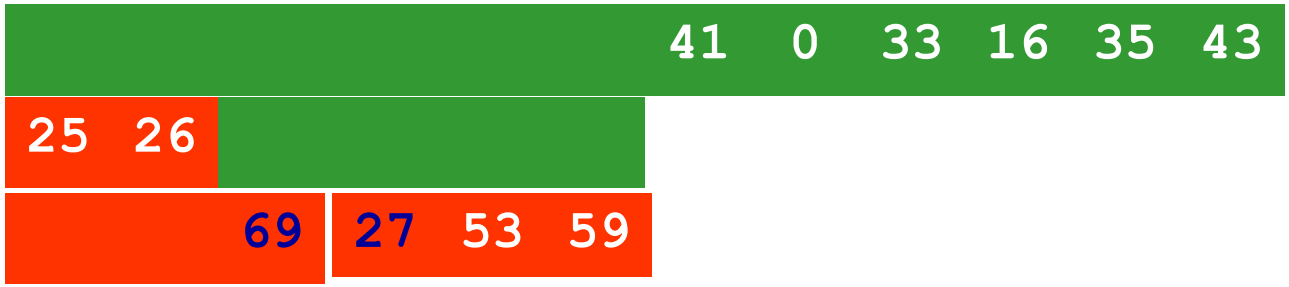



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

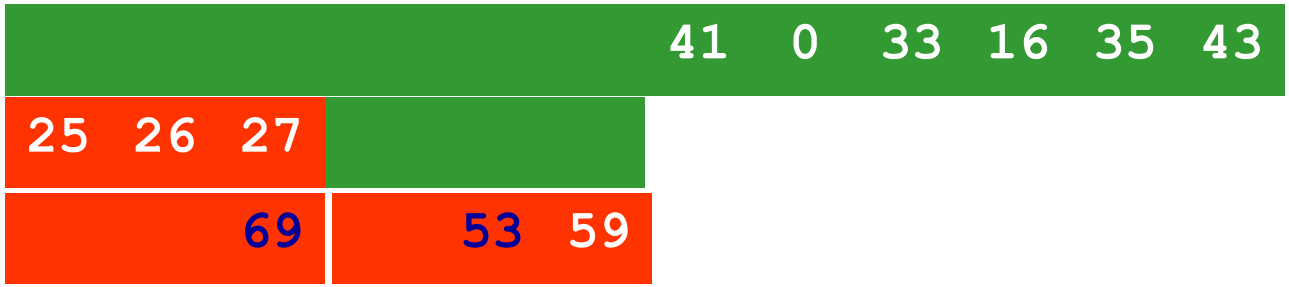



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

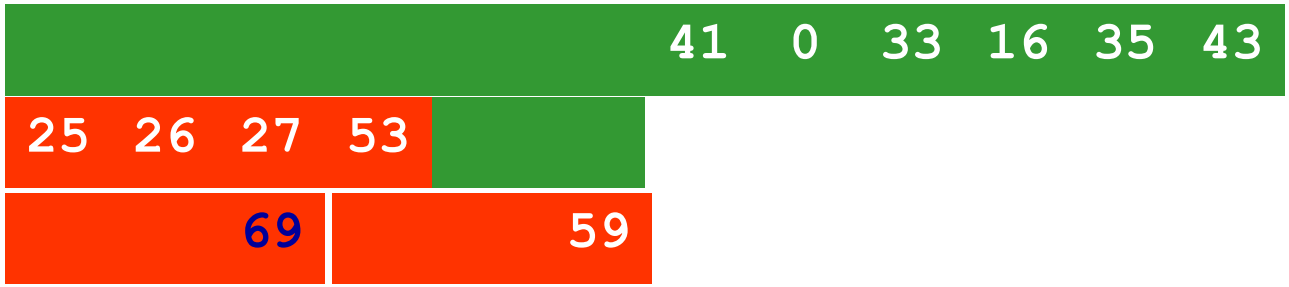



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

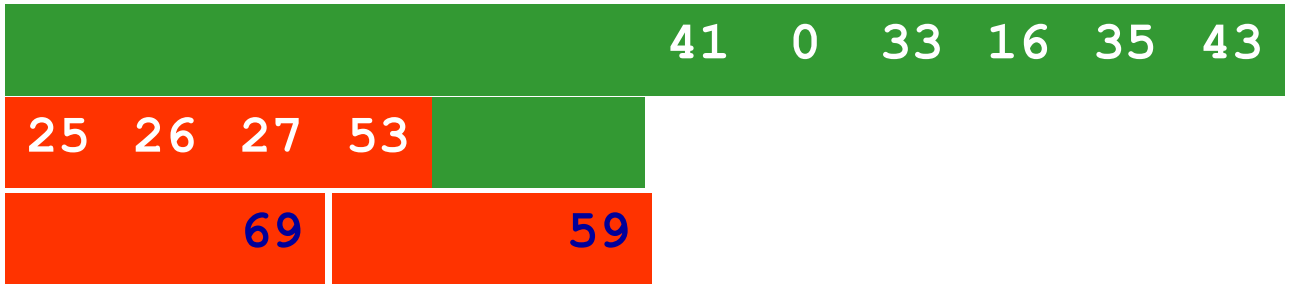



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

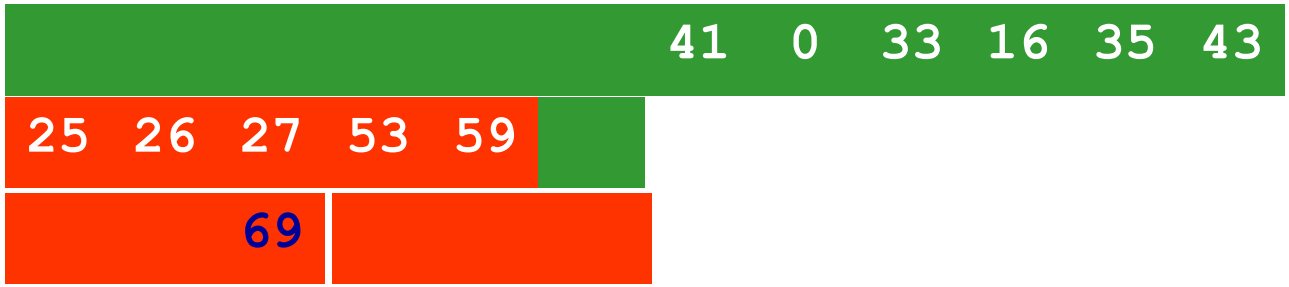



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

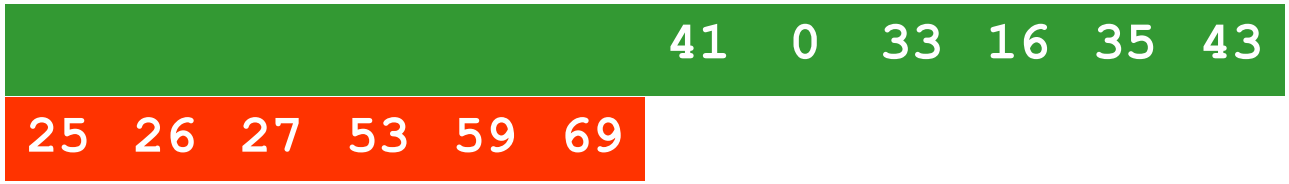



```
...
Se (inicio < fim) {

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

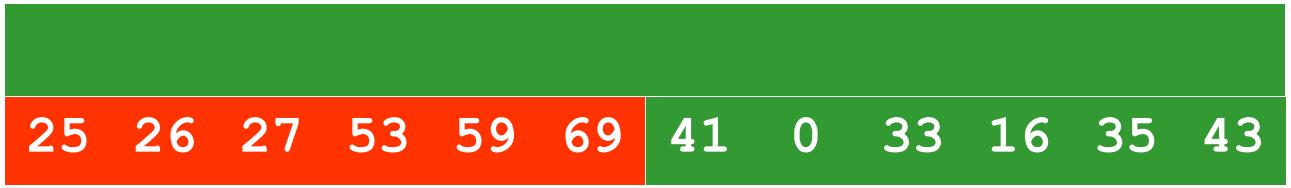
# O algoritmo MergeSort

- Execução:



# O algoritmo MergeSort

- Execução:

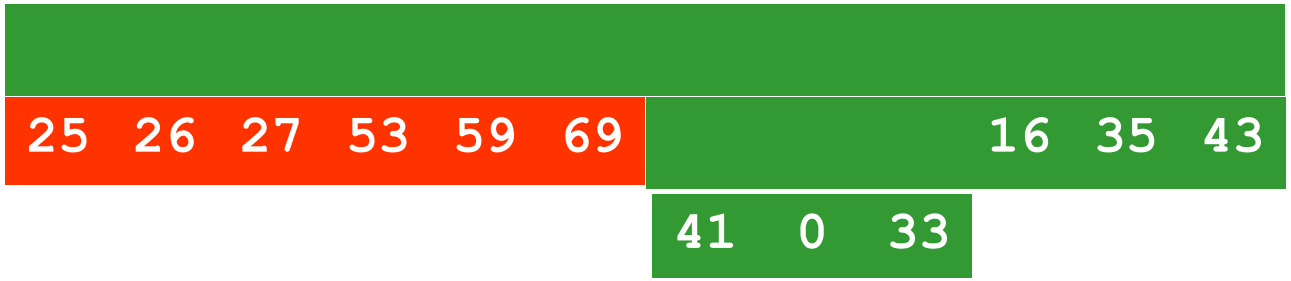



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

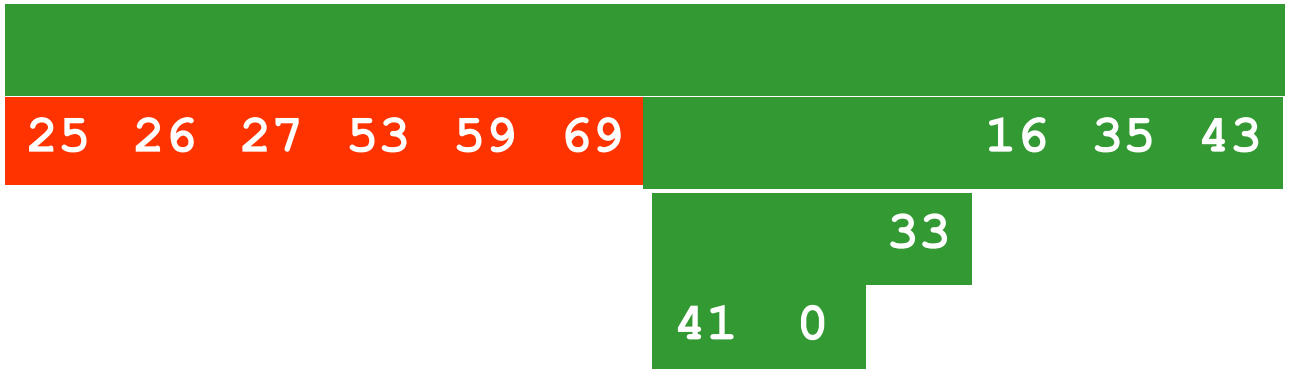



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

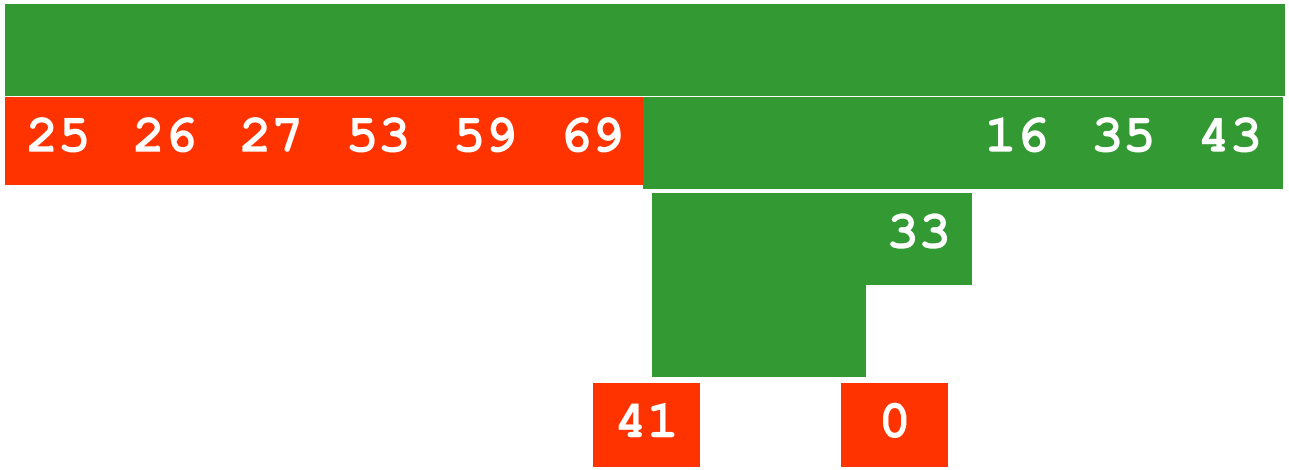



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

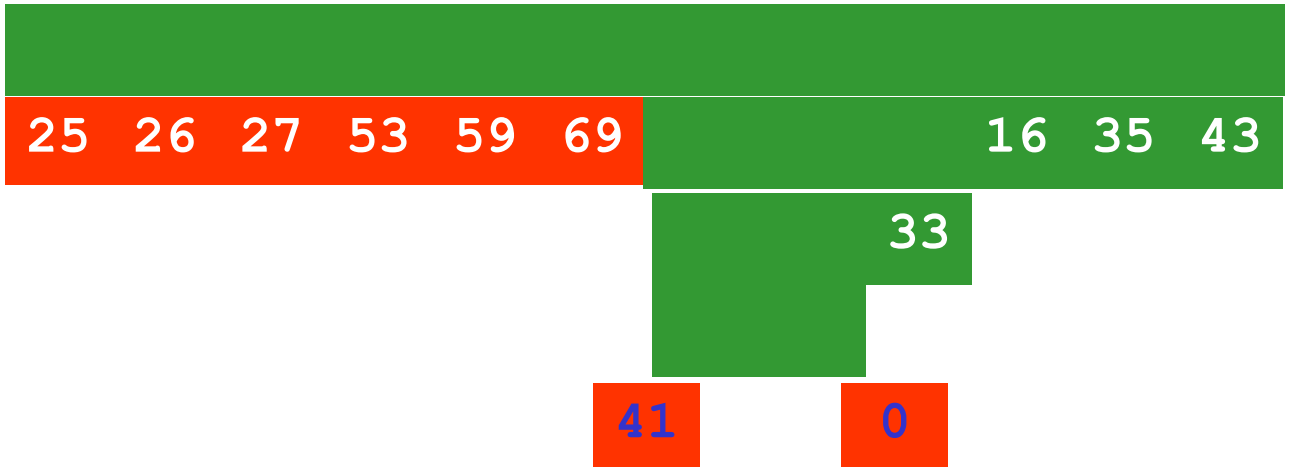



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

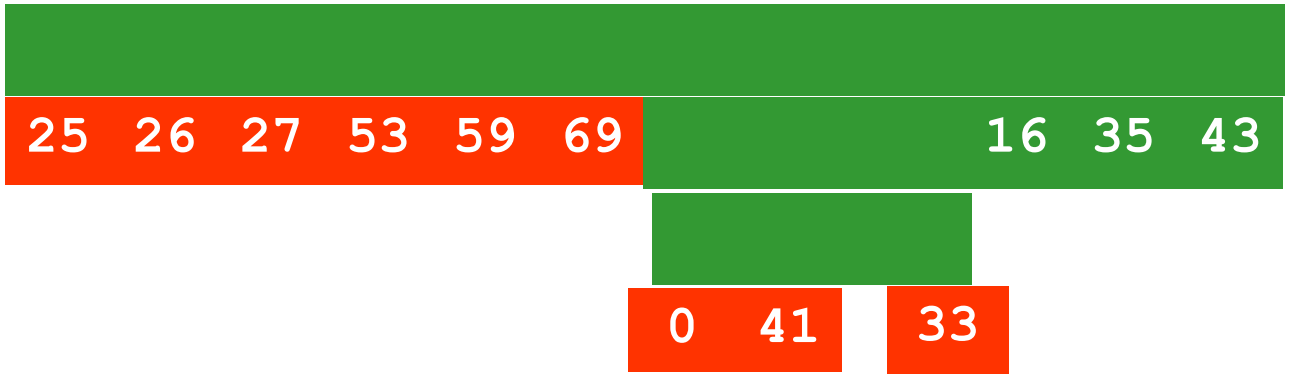



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

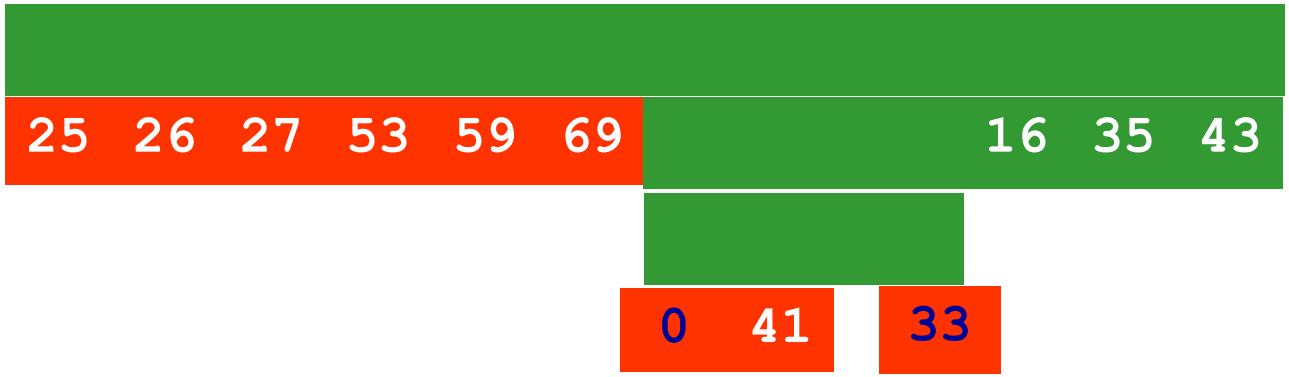



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

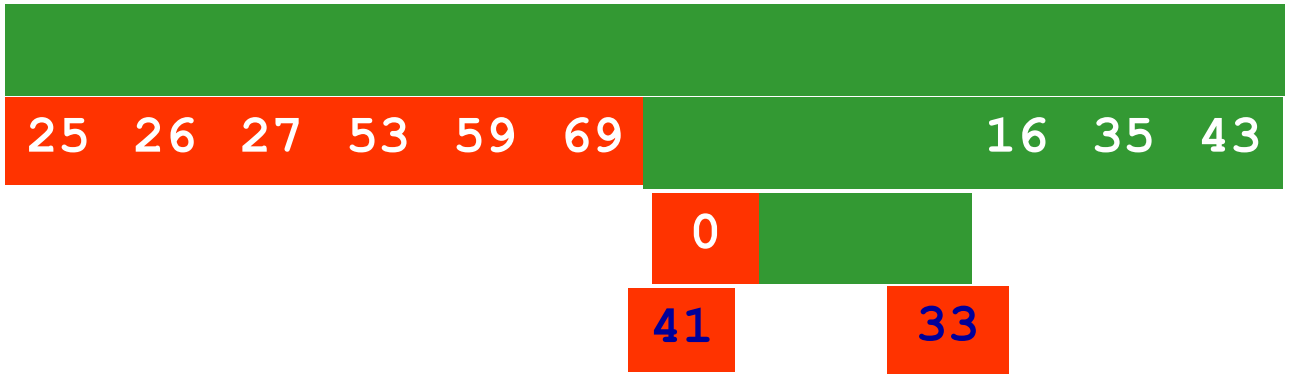



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

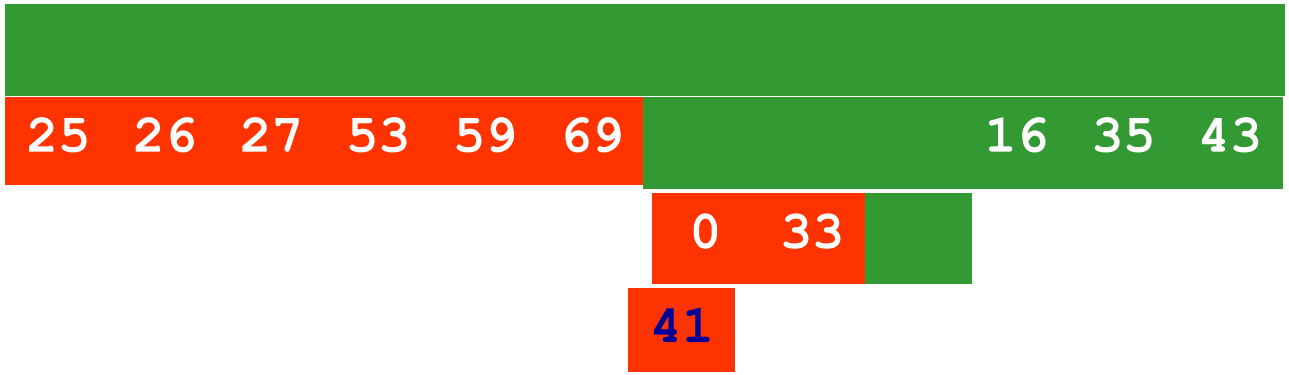



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

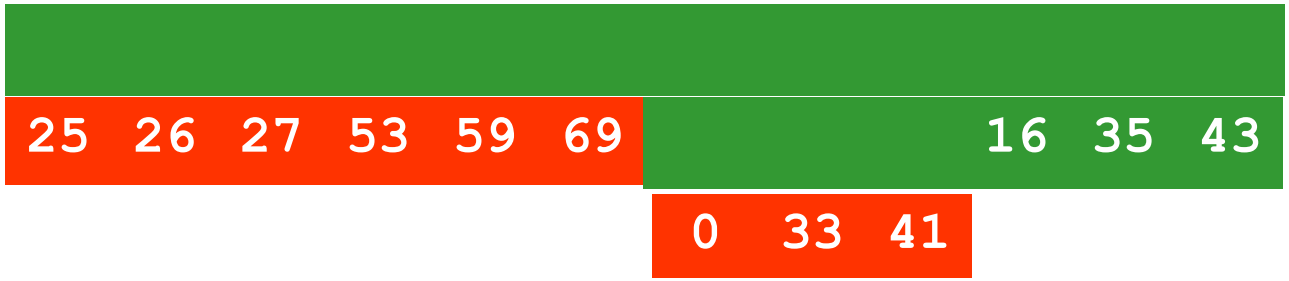



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

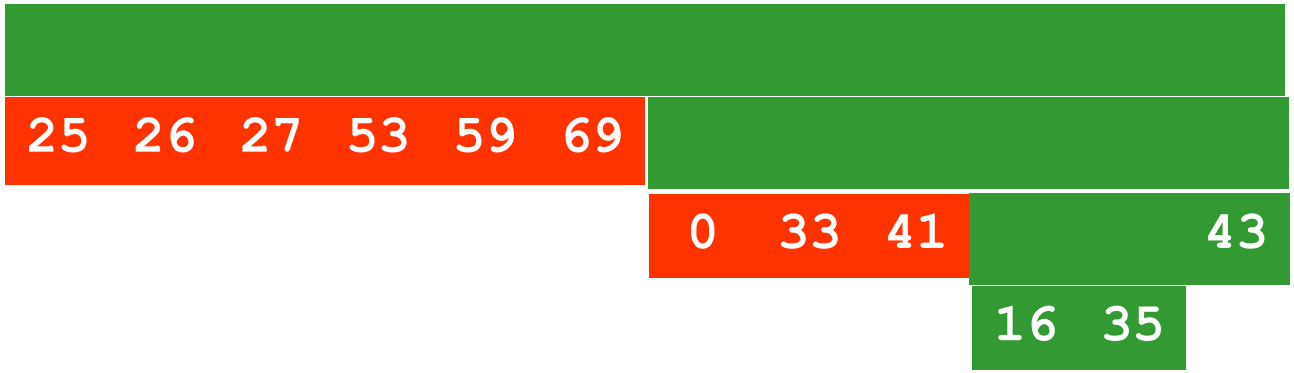



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

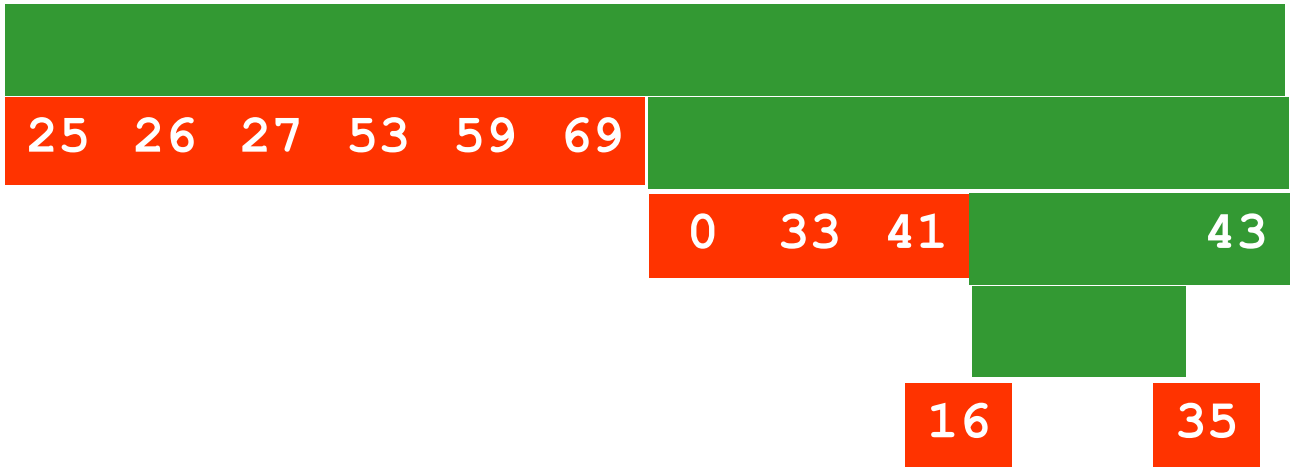



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

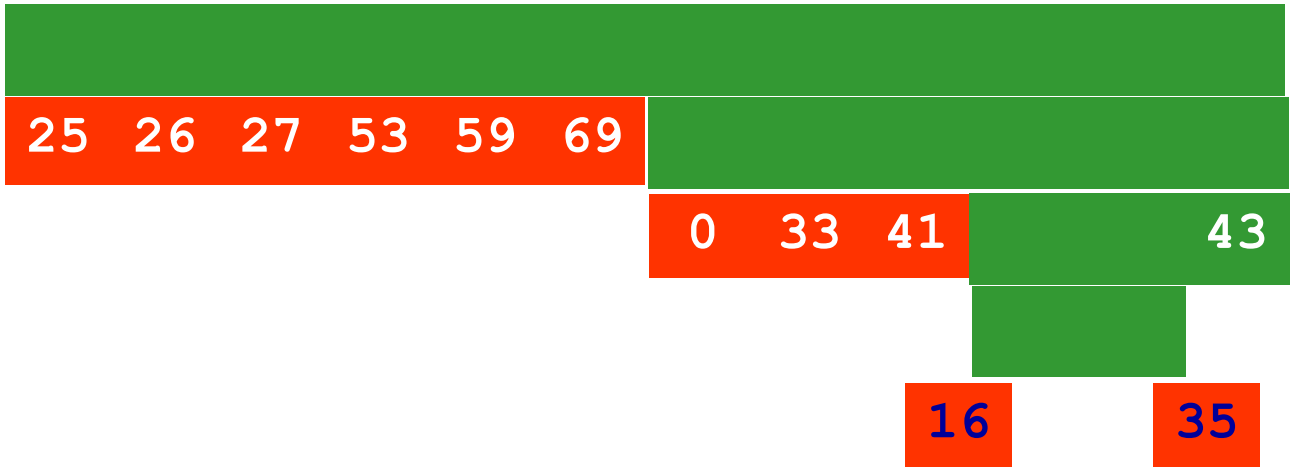



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

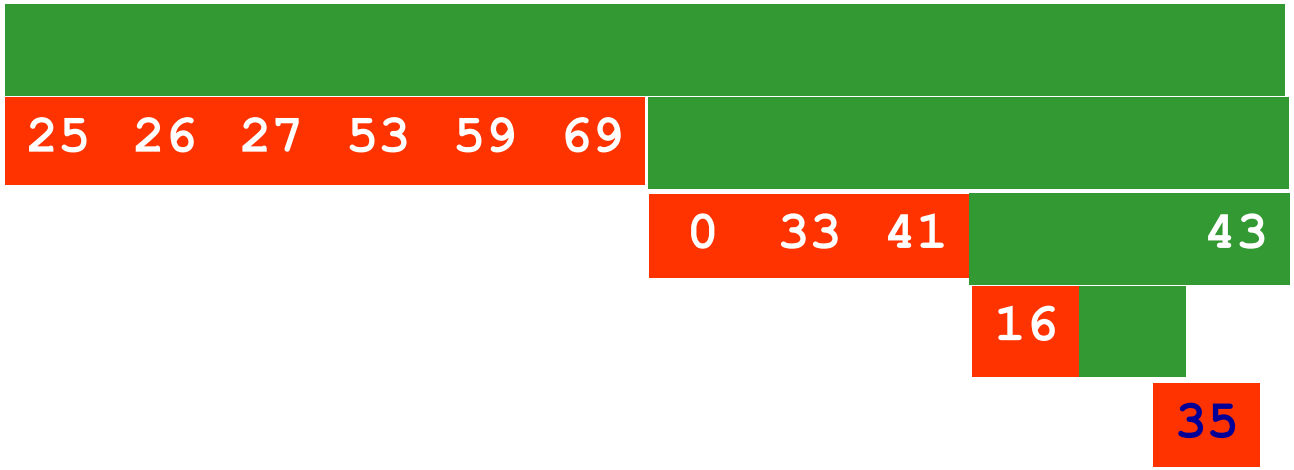



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

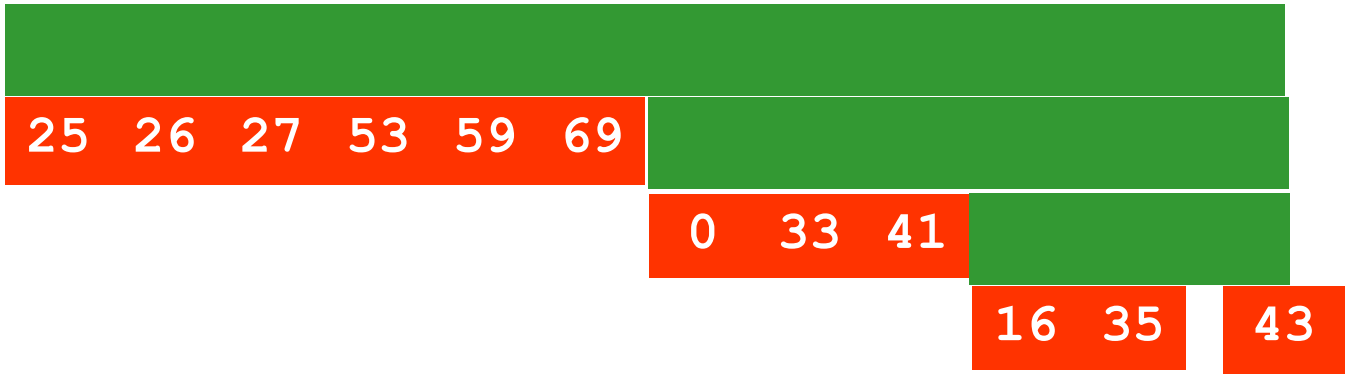



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

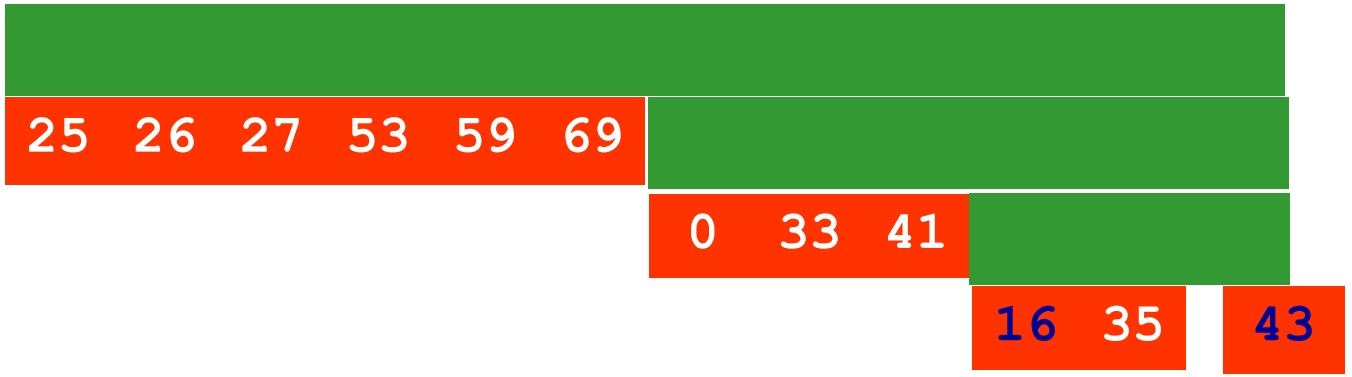



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

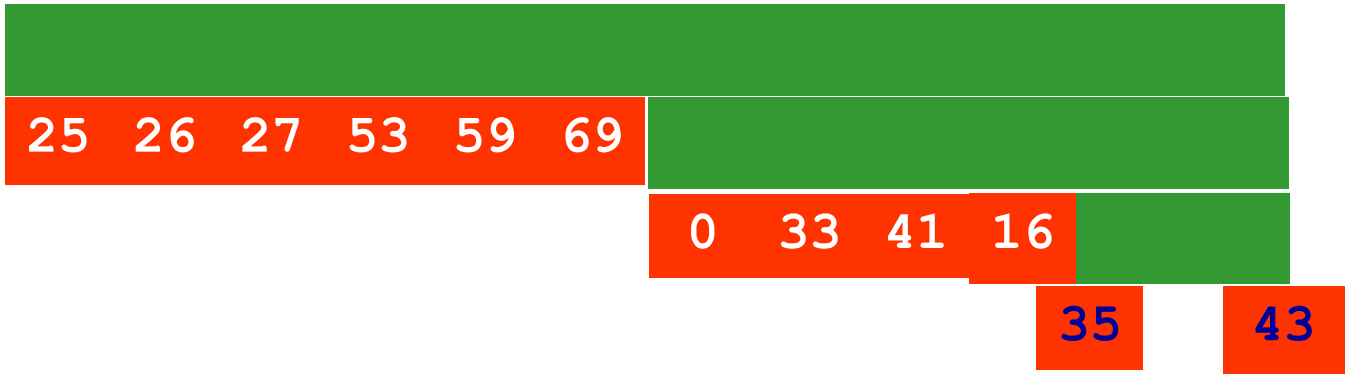



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

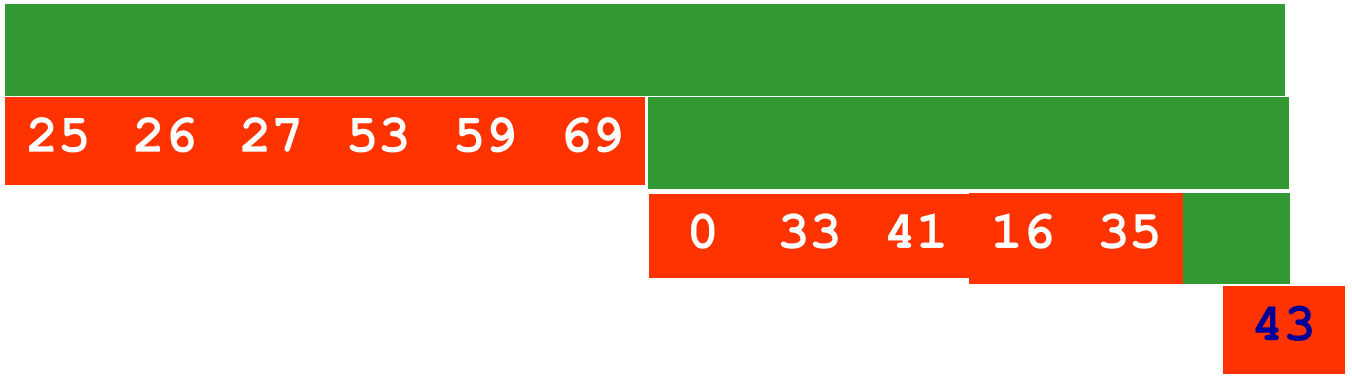



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

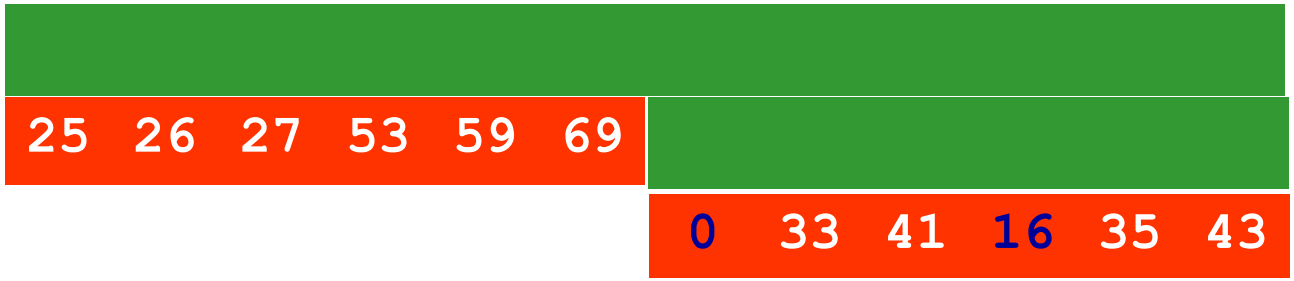



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

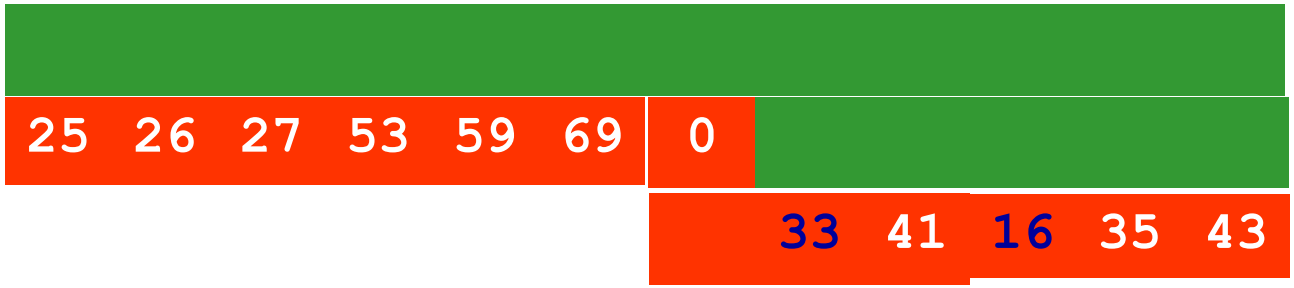



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

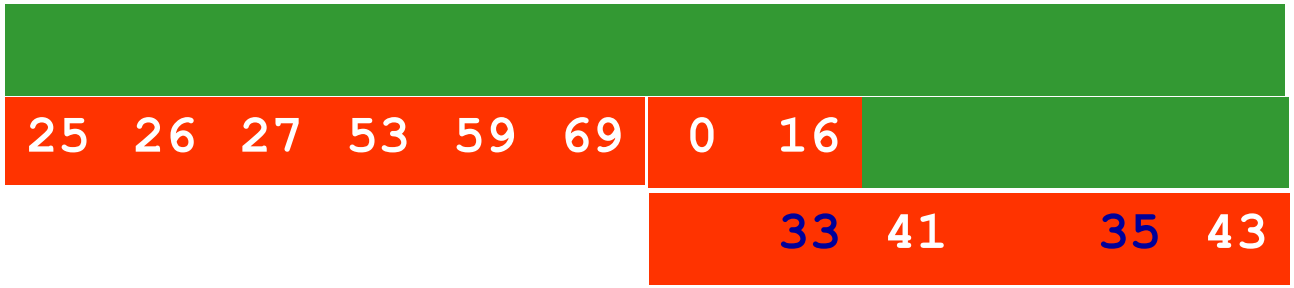



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

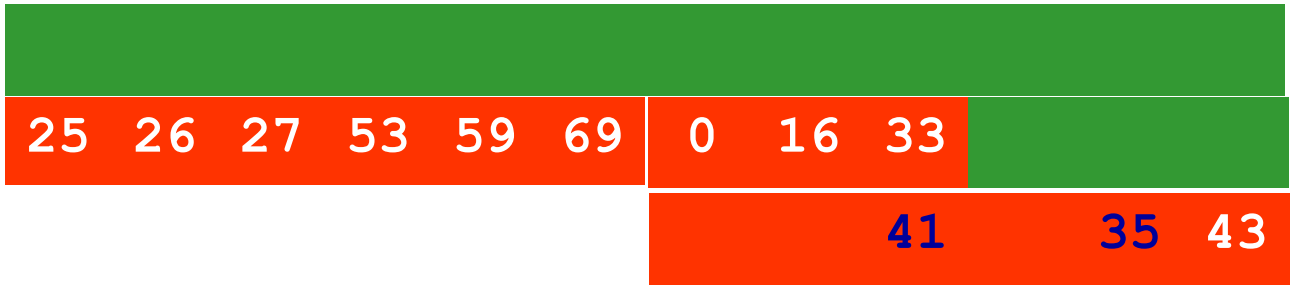



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

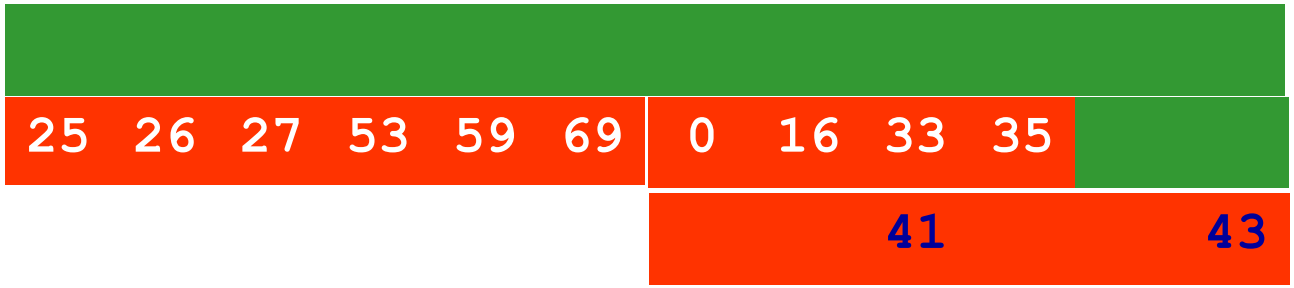



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

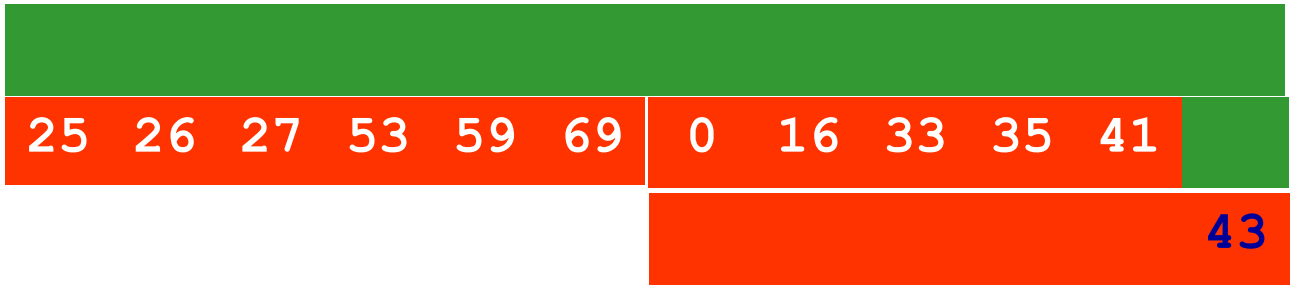



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

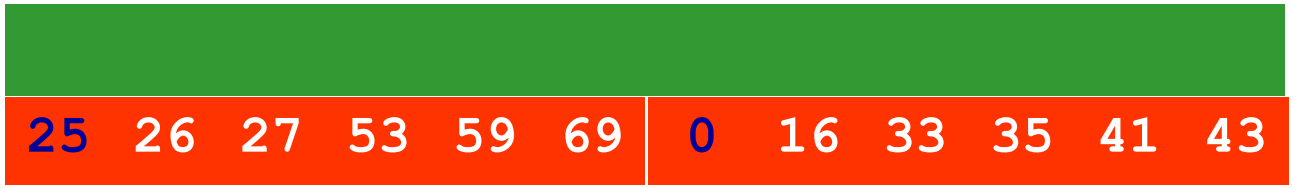



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

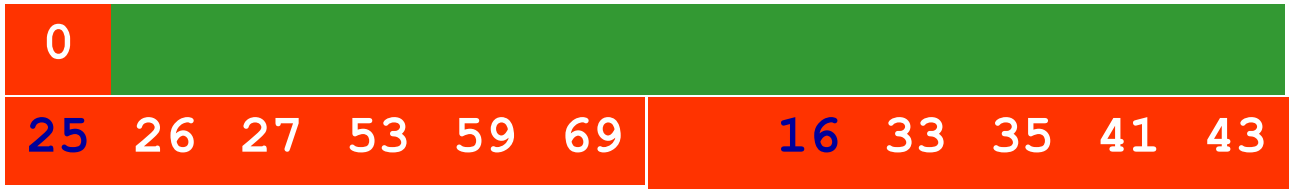



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

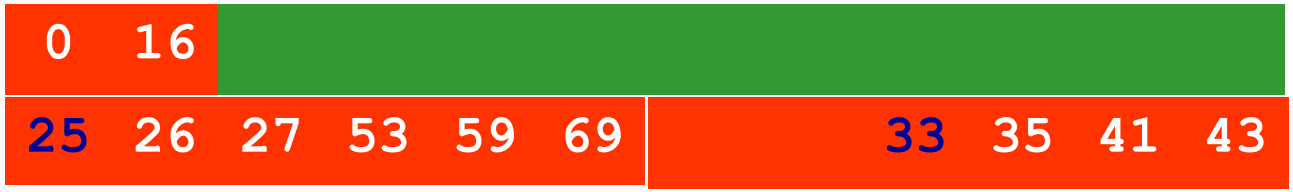



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

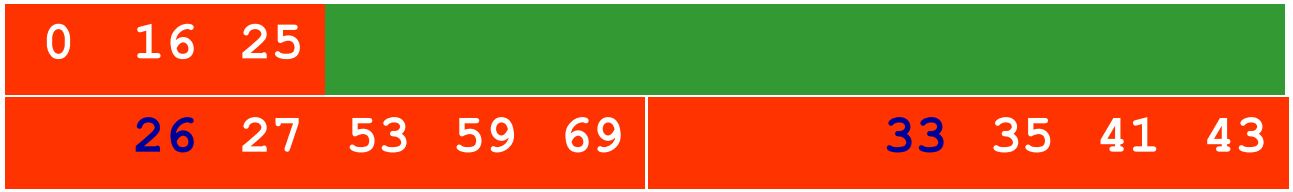



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

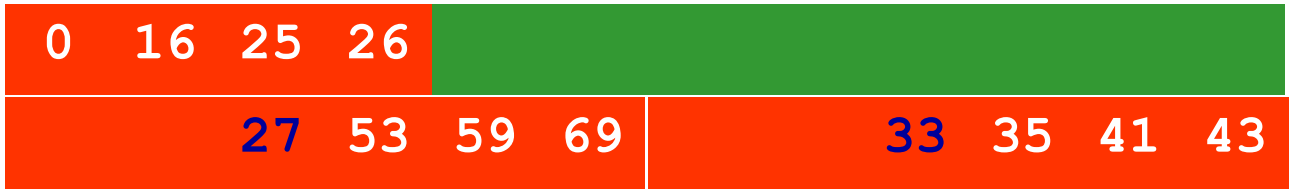



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

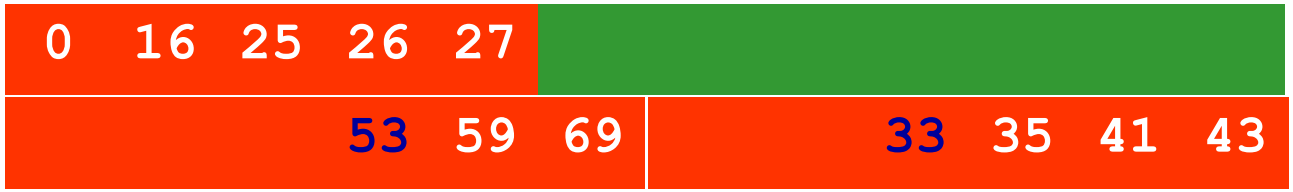



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

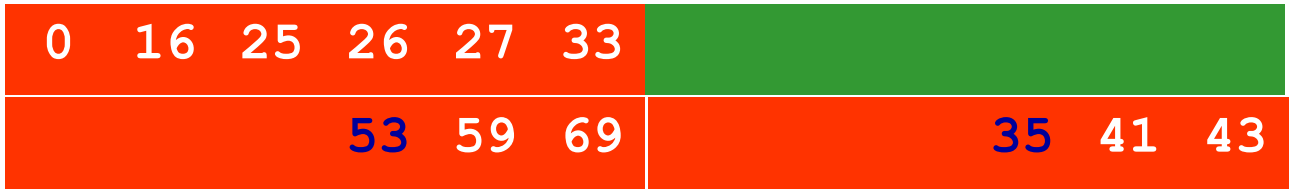



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

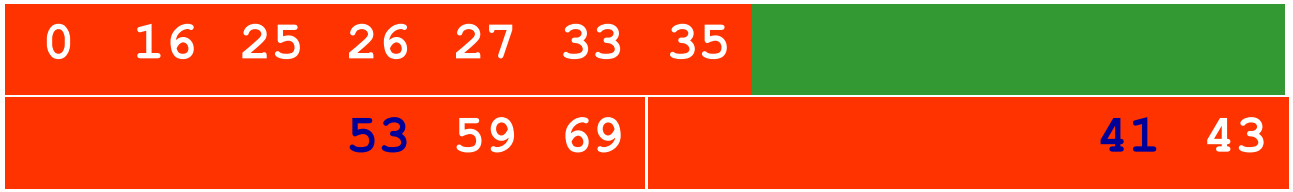



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

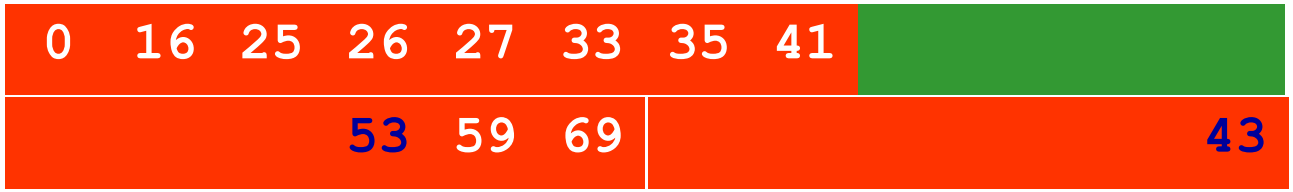



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

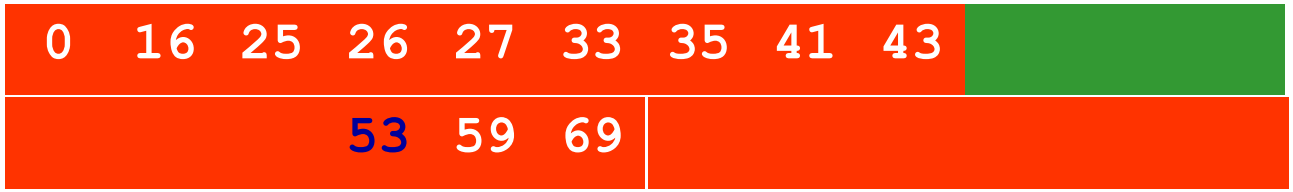



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

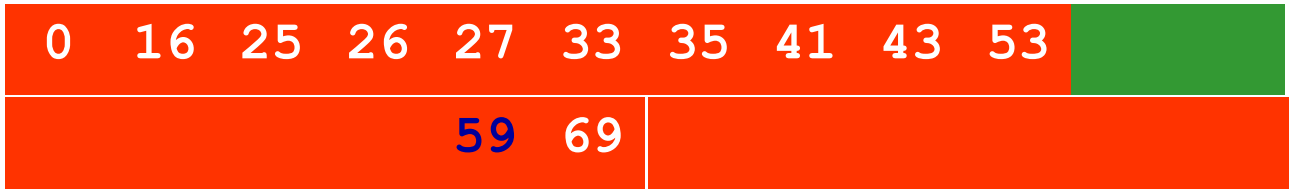



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:

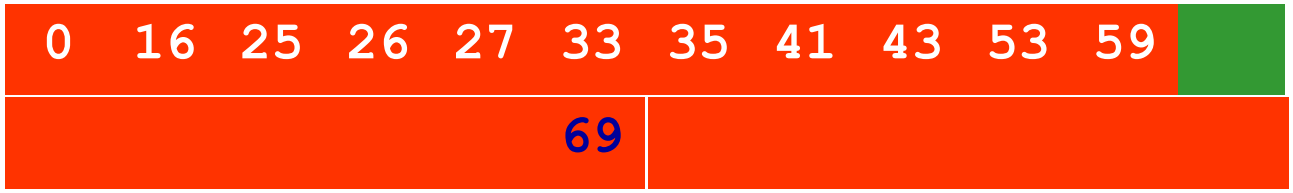



```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Execução:




```
...
Se (inicio < fim) {

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

# O algoritmo MergeSort

- Vetor Ordenado:

# O algoritmo MergeSort

- Implementação do algoritmo MergeSort:

```
principal
{
    inteiro vetor[] = {7, 6, 5, 4, 8, 22, 3, 55, 7, 99, 11, 3, 1, 88};
    inteiro tamanho;
    inteiro indice;

    //Pode-se usar a função sizeof()
    tamanho ← tamanho_vetor / tamanho_de_um_inteiro;

    mergesort(vetor, 0, tamanho);

}
```

# O algoritmo MergeSort

- Implementação do algoritmo MergeSort:

```
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);
        }
}
```

# O algoritmo MergeSort

- Implementação do algoritmo MergeSort:

```
intercala(inteiro *vetor, inteiro inicio, inteiro meio, inteiro fim)
{
    inteiro indice_inicio;
    inteiro indice_meio;
    inteiro indice_aux;
    inteiro *vetor_aux;

    indice_inicio ← inicio;
    indice_meio ← meio+1;
    indice_aux ← 0;

    vetor_aux ← alocacao_memoria(vetor);
    ...
    ...
    //Continua no slide posterior
```

# O algoritmo MergeSort

- Implementação do algoritmo MergeSort:

```
enquanto(indice_inicio < meio+1 ou indice_meio < fim+1) {
       se (indice_inicio = meio+1) {
           vetor_aux[indice_aux] ← vetor[indice_meio];
           indice_meio ← indice_meio + 1;
           indice_aux ← indice_aux + 1;

       } senao se (indice_meio = fim+1) {
           vetor_aux[indice_aux] ← vetor[indice_inicio];
           indice_inicio ← indice_inicio + 1;
           indice_aux ← indice_aux + 1;

       } senao se (vetor[indice_inicio] <= vetor[indice_meio]) {
           vetor_aux[indice_aux] ← vetor[indice_inicio];
           indice_inicio ← indice_inicio + 1;
           indice_aux ← indice_aux + 1;

       } senao {
           vetor_aux[indice_aux] ← vetor[indice_meio];
           indice_meio ← indice_meio + 1;
           indice_aux ← indice_aux + 1;

       }
  }
  ...
  //Continua no slide posterior
```

# O algoritmo MergeSort

- Implementação do algoritmo MergeSort:

```
 // copia vetor intercalado para o vetor original
  para(indice_inicio = inicio;indice_inicio <= fim;indice_inicio=indice_inicio+1)
      vetor[indice_inicio] ← vetor_aux[indice_inicio-inicio];

  libera_memoria(vetor_aux);

}
```

# Estudo da estabilidade

- O algoritmo é considerado estável, pois não há a possibilidade de elementos iguais mudar de posição no processo de ordenação
- A fase de divisão do algoritmo não altera a posição de nenhuma chave
- Ordenação é feita pelo algoritmo de intercalação
  - A intercalação é feita verificando, sequencialmente, os elementos de acordo com sua posição no array
  - Dessa forma, elementos com mesma chave não terão a sua posição relativa alterada