Aula 18 - MergeSort

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023

Projeto por Indução Forte

Segunda Alternativa

Projeto por Indução Forte

Segunda Alternativa

- **Base**: $n = 1$. Um conjunto de um único elemento está ordenado

Projeto por Indução Forte

Segunda Alternativa

- **Base**: $n = 1$. Um conjunto de um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $1 \leq k < n$ valores

Projeto por Indução Forte

Segunda Alternativa

- **Base**: $n = 1$. Um conjunto de um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $1 \leq k < n$ valores
- **Passo**:

Projeto por Indução Forte

Segunda Alternativa

- **Base**: $n = 1$. Um conjunto de um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $1 \leq k < n$ valores
- **Passo**:
  - Seja S um conjunto de $n \geq 2$ valores. Podemos particionar S em dois conjuntos, S_1 e S_2 , de tamanhos $n/2$

Projeto por Indução Forte

Segunda Alternativa

- **Base**: $n = 1$. Um conjunto de um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $1 \leq k < n$ valores
- **Passo**:
  - Seja S um conjunto de $n \geq 2$ valores. Podemos particionar S em dois conjuntos, S_1 e S_2 , de tamanhos $n/2$
  - Pela H.I., sabemos ordenar os conjuntos S_1 e S_2. Podemos então obter S ordenado intercalando os conjuntos ordenados S_1 e S_2

Projeto por Indução Forte

Segunda Alternativa

- **Base**: $n = 1$. Um conjunto de um único elemento está ordenado
- **H.I.**: Sei ordenar um conjunto de $1 \leq k < n$ valores
- **Passo**:
  - Seja S um conjunto de $n \geq 2$ valores. Podemos particionar S em dois conjuntos, S_1 e S_2 , de tamanhos $n/2$
  - Pela H.I., sabemos ordenar os conjuntos S_1 e S_2. Podemos então obter S ordenado intercalando os conjuntos ordenados S_1 e S_2
- *MergeSort*

MergeSort

Passos para ordenar um arranjo A

MergeSort

Passos para ordenar um arranjo A

- Dividir:

MergeSort

Passos para ordenar um arranjo A

- Dividir:
  - Divida o arranjo de n elementos em dois sub-arranjos de $n/2$ elementos cada

MergeSort

Passos para ordenar um arranjo A

- Dividir:
  - Divida o arranjo de n elementos em dois sub-arranjos de $n/2$ elementos cada
- Conquistar:

MergeSort

Passos para ordenar um arranjo A

- Dividir:
  - Divida o arranjo de n elementos em dois sub-arranjos de $n/2$ elementos cada
- Conquistar:
  - Ordene os dois sub-arranjos recursivamente

MergeSort

Passos para ordenar um arranjo A

- Dividir:
  - Divida o arranjo de n elementos em dois sub-arranjos de $n/2$ elementos cada
- Conquistar:
  - Ordene os dois sub-arranjos recursivamente
- Combinar:

MergeSort

Passos para ordenar um arranjo A

- Dividir:
  - Divida o arranjo de n elementos em dois sub-arranjos de $n/2$ elementos cada
- Conquistar:
  - Ordene os dois sub-arranjos recursivamente
- Combinar:
  - Intercale os dois sub-arranjos ordenados, de modo a produzir a resposta ordenada

MergeSort

Passos para ordenar um arranjo A

- Podemos ver que a divisão é imediata

MergeSort

Passos para ordenar um arranjo A

- Podemos ver que a divisão é imediata
  - O arranjo é dividido em dois sub-arranjos com metade do tamanho do original, que são ordenados recursivamente

MergeSort

Passos para ordenar um arranjo A

- Podemos ver que a divisão é imediata
  - O arranjo é dividido em dois sub-arranjos com metade do tamanho do original, que são ordenados recursivamente
- A conquista também não é difícil

MergeSort

Passos para ordenar um arranjo A

- Podemos ver que a divisão é imediata
  - O arranjo é dividido em dois sub-arranjos com metade do tamanho do original, que são ordenados recursivamente
- A conquista também não é difícil
  - Inevitavelmente vamos parar no caso base, que é trivial

MergeSort

Passos para ordenar um arranjo A

- Podemos ver que a divisão é imediata
  - O arranjo é dividido em dois sub-arranjos com metade do tamanho do original, que são ordenados recursivamente
- A conquista também não é difícil
  - Inevitavelmente vamos parar no caso base, que é trivial
- E a combinação?

MergeSort

Passos para ordenar um arranjo A

- Podemos ver que a divisão é imediata
  - O arranjo é dividido em dois sub-arranjos com metade do tamanho do original, que são ordenados recursivamente
- A conquista também não é difícil
  - Inevitavelmente vamos parar no caso base, que é trivial
- E a combinação?
  - Como intercalar os 2 sub-arranjos ordenados?

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

Arranjo a ser ordenado

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

índices no arranjo, sendo $p \leq q < r$

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

O procedimento assume que os sub-arranjos `A[p..q]` e `A[q+1..r]` já estão ordenados

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

Os sub-arranjos serão então intercalados para formar um único sub-arranjo ordenado, substituindo o sub-arranjo `A[p..r]` atual

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

Tamanho da sub-sequência 1

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

Tamanho da sub-sequência 2

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

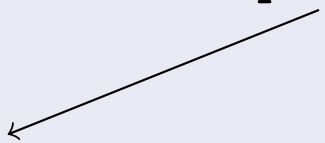
Copia A[p..q] em seq1

MergeSort

Intercalação

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];

  ...
```

E A[q+1..r] em seq2



MergeSort

Intercalação (cont.)

```
...

k = p; i = 0; j = 0;
while (i < tamseq1 && j < tamseq2) {
  if(seq2[j] < seq1[i]) {
    A[k] = seq2[j];
    j++;
  }
  else {
    A[k] = seq1[i];
    i++;
  }
  k++;
}

...
```

Juntamos agora as subsequências

MergeSort

Intercalação (cont.)

```
...

k = p; i = 0; j = 0;
while (i < tamseq1 && j < tamseq2) {
  if(seq2[j] < seq1[i]) {
    A[k] = seq2[j];
    j++;
  }
  else {
    A[k] = seq1[i];
    i++;
  }
  k++;
}

...
```

O laço vai até terminar a menor subsequência

MergeSort

Intercalação (cont.)

```
...

k = p; i = 0; j = 0;
while (i < tamseq1 && j < tamseq2) {
  if(seq2[j] < seq1[i]) {
    A[k] = seq2[j];
    j++;
  }
  else {
    A[k] = seq1[i];
    i++;
  }
  k++;
}

...
```

A[p...] recebe o menor elemento dentre as 2 subsequências

Intercalação (cont.)

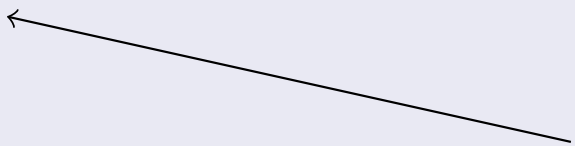
```
  ...

  while (i < tamseq1) {
    A[k] = seq1[i];
    k++;
    i++;
  }

  while (j < tamseq2) {
    A[k] = seq2[j];
    k++;
    j++;
  }
}
```


Completa com a subsequência que ainda não acabou

MergeSort

Intercalação (cont.)

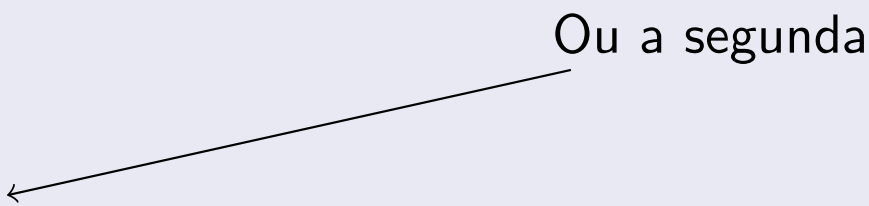
```
  ...

  while (i < tamseq1) {
    A[k] = seq1[i];
    k++;
    i++;
  }

  while (j < tamseq2) {
    A[k] = seq2[j];
    k++;
    j++;
  }
}
```



MergeSort

Intercalação (cont.)

```
  ...

  while (i < tamseq1) {
    A[k] = seq1[i];
    k++;
    i++;
  }

  while (j < tamseq2) {
    A[k] = seq2[j];
    k++;
    j++;
  }
}
```



MergeSort

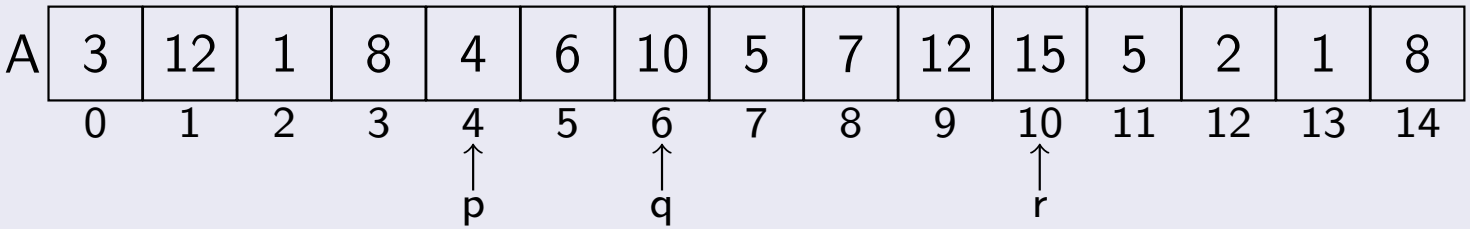
Intercalação (cont.)

```
  ...

  while (i < tamseq1) {
    A[k] = seq1[i];
    k++;
    i++;
  }

  while (j < tamseq2) {
    A[k] = seq2[j];
    k++;
    j++;
  }
}
```

No final, a sub-sequência `A[p..r]` está ordenada

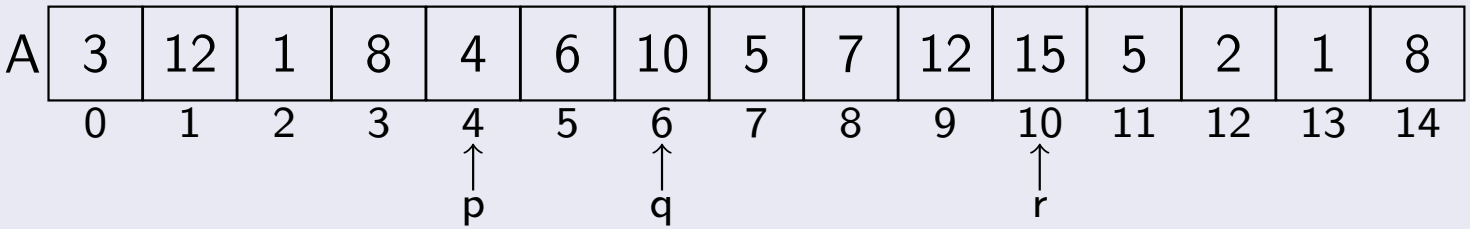
Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:



Intercalação: Exemplo

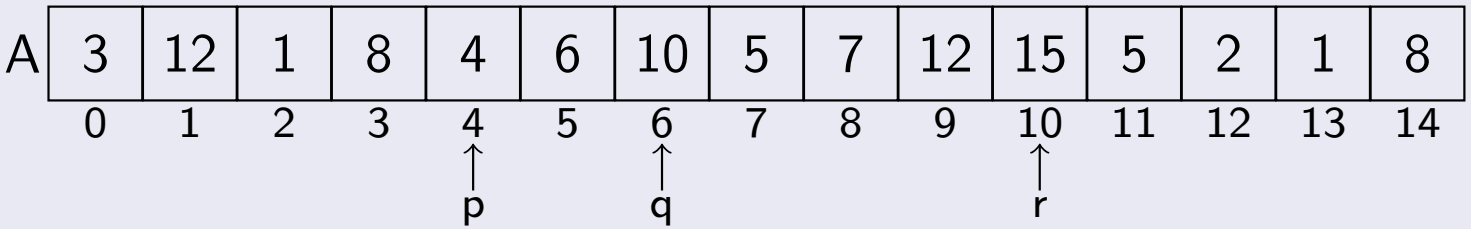
Considere o arranjo A e os índices p, q e r:



Note que `A[p..q]` e `A[q+1..r]` estão ordenados

Intercalação: Exemplo

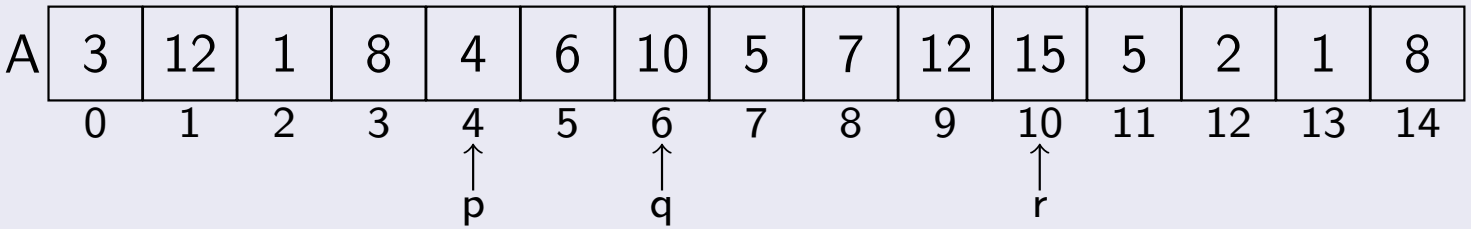
Considere o arranjo A e os índices p, q e r:



Copiamos `A[p..q]` em `seq1` e `A[q+1..r]` em `seq2`

Intercalação: Exemplo

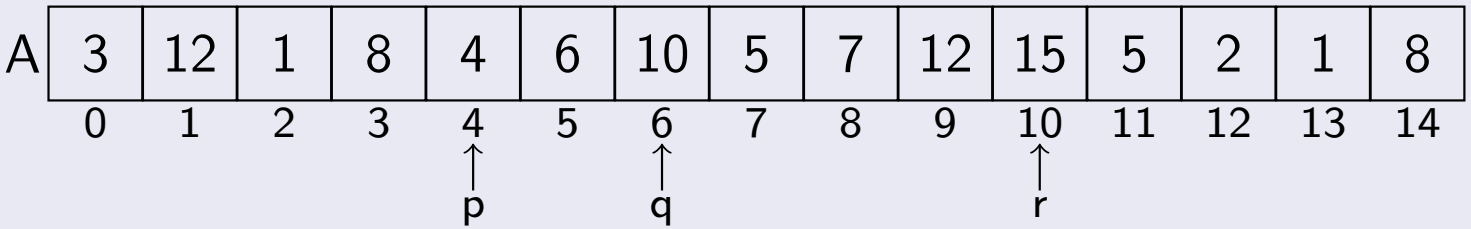
Considere o arranjo A e os índices p, q e r:



`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

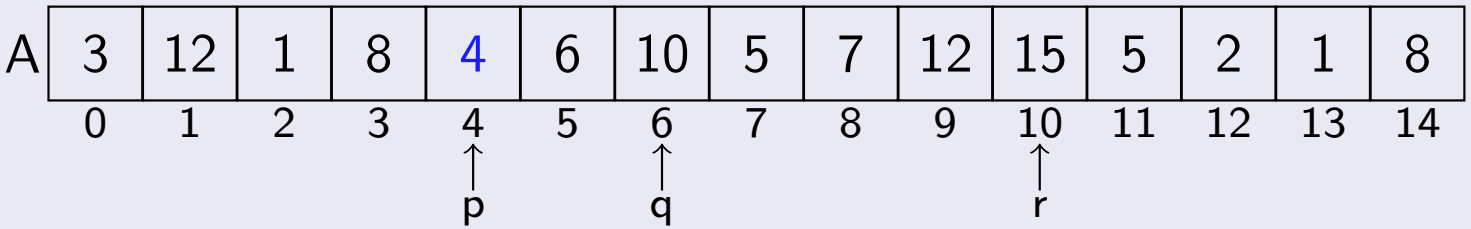
Considere o arranjo A e os índices p, q e r:



`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

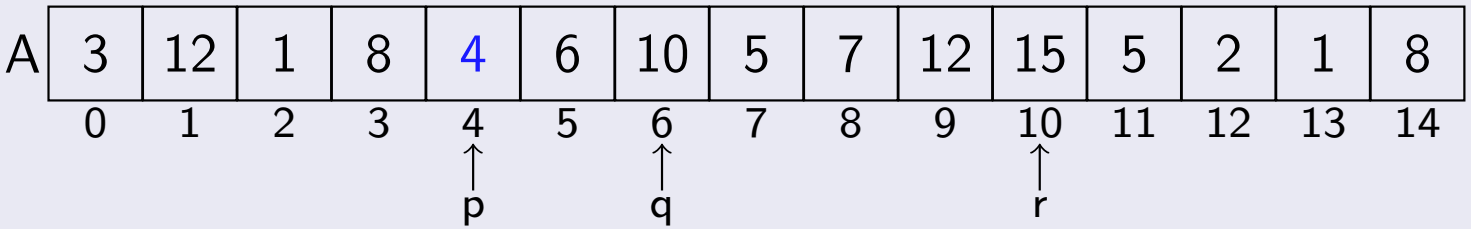
Considere o arranjo A e os índices p, q e r:



`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

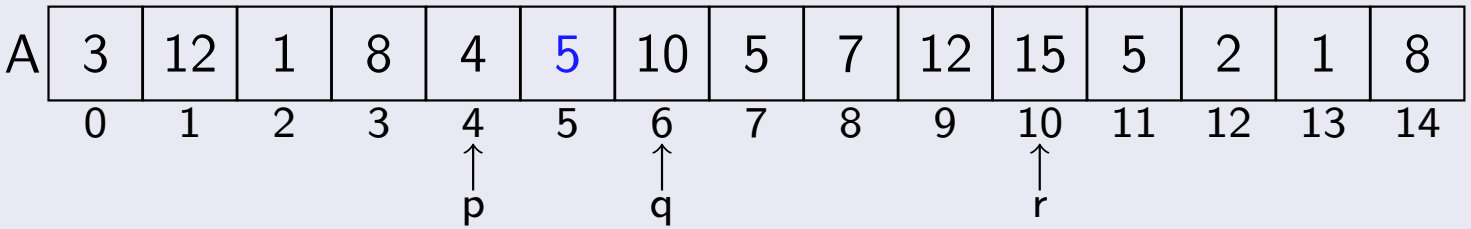
Considere o arranjo A e os índices p, q e r:



`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

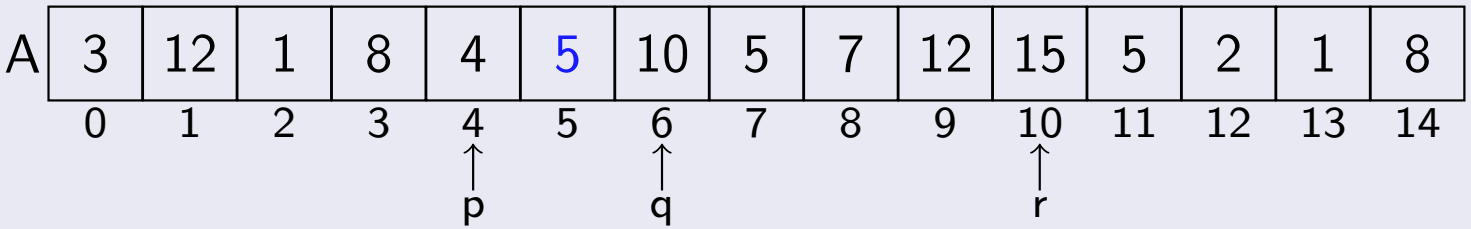
Considere o arranjo A e os índices p, q e r:

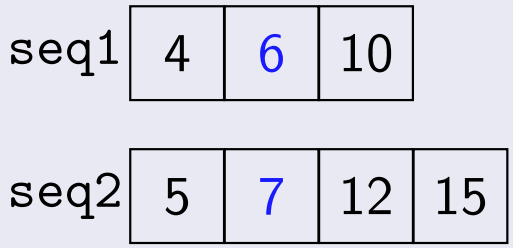


`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:

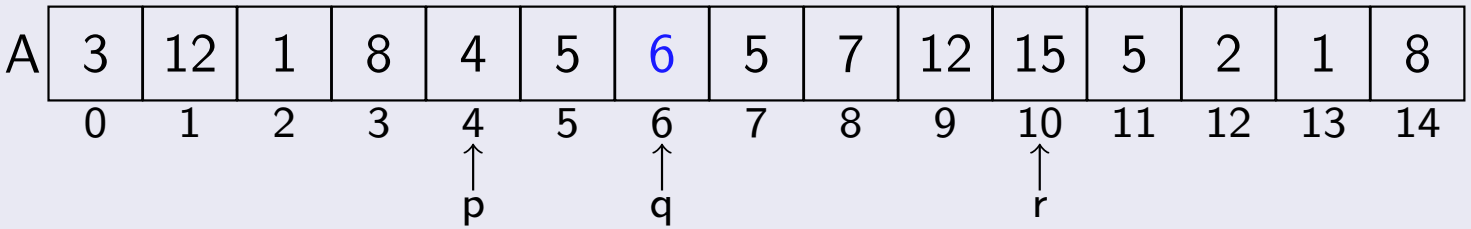


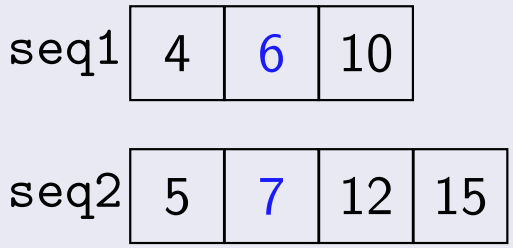


`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:

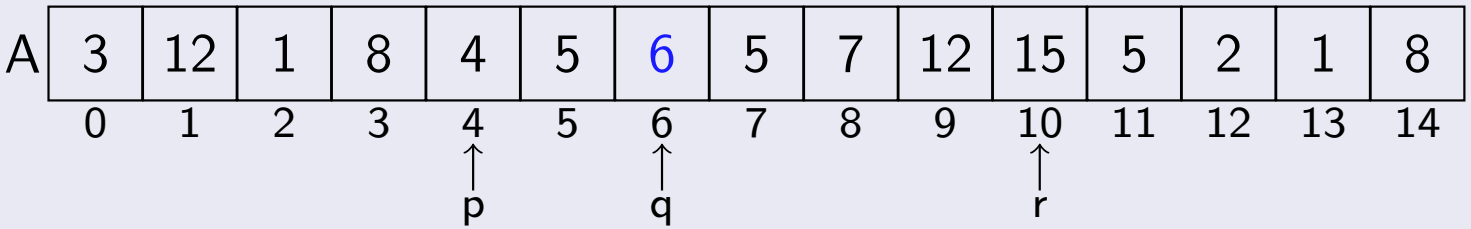


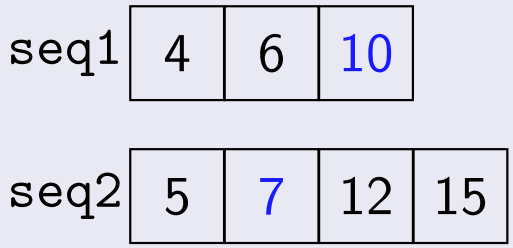


`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:

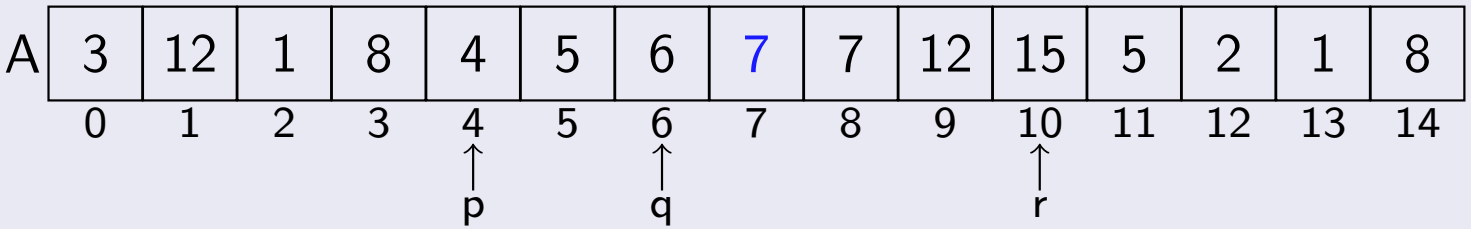


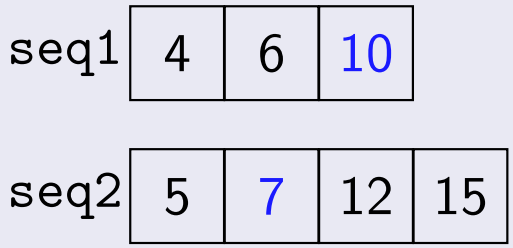


`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:

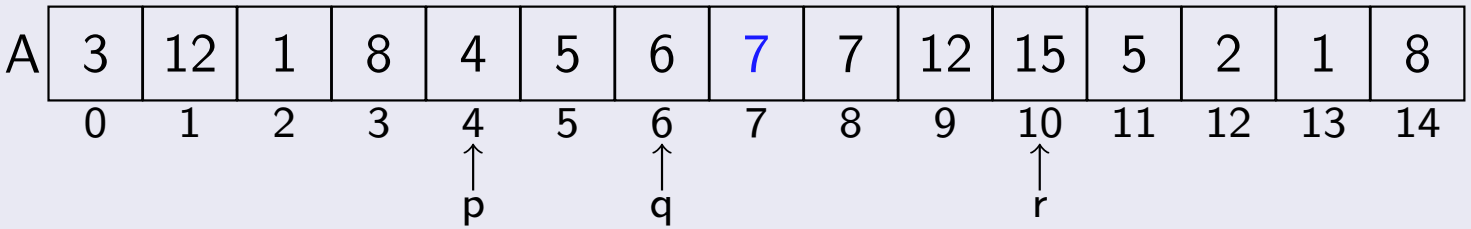




`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

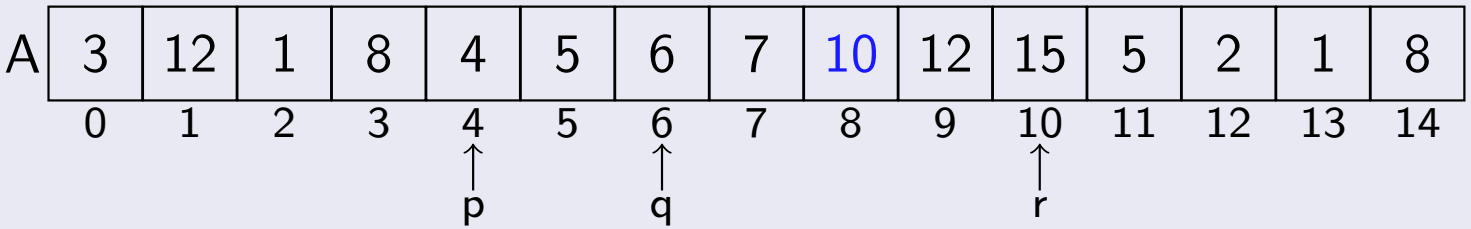
Considere o arranjo A e os índices p, q e r:



`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

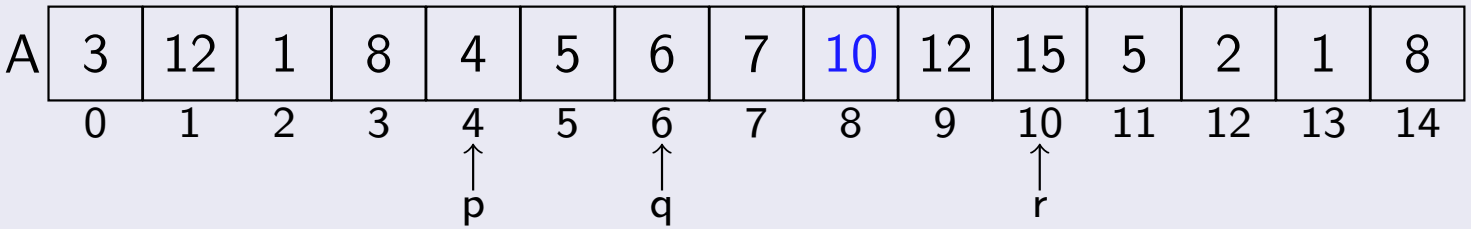
Considere o arranjo A e os índices p, q e r:



`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

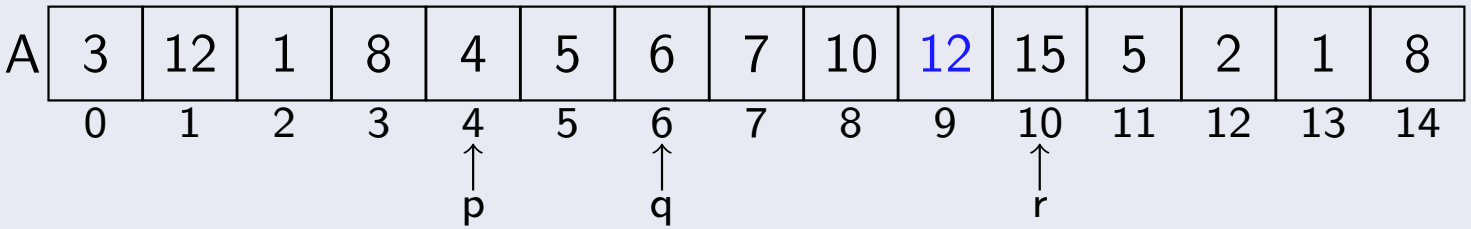
Considere o arranjo A e os índices p, q e r:

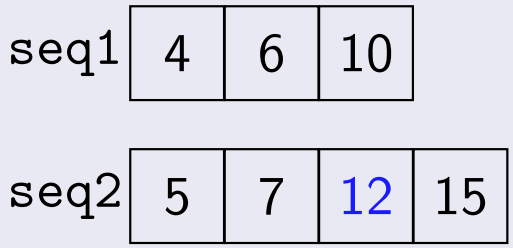


`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:

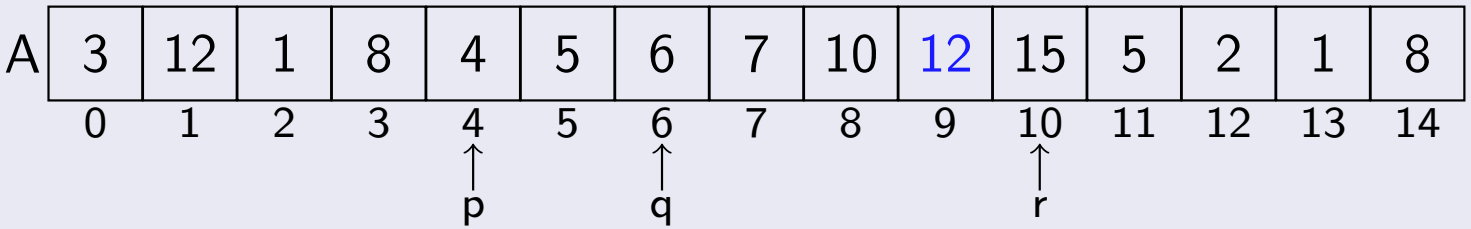


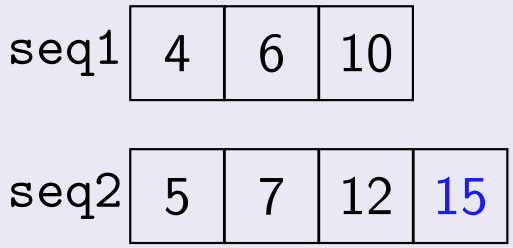


`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:

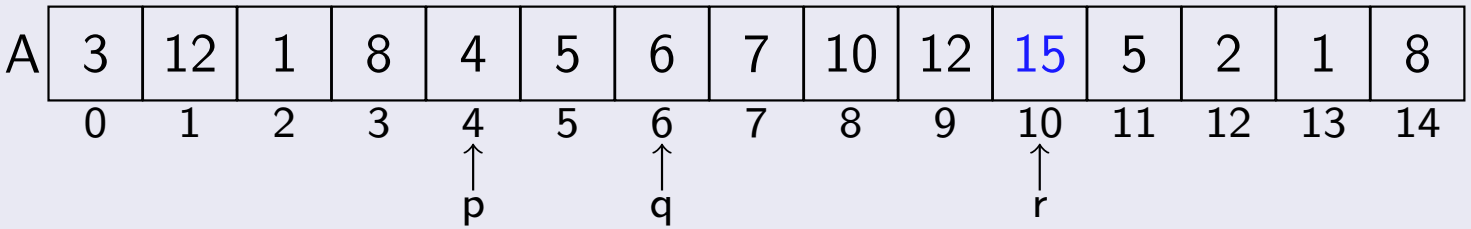


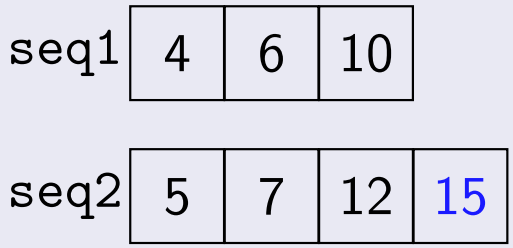


`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:

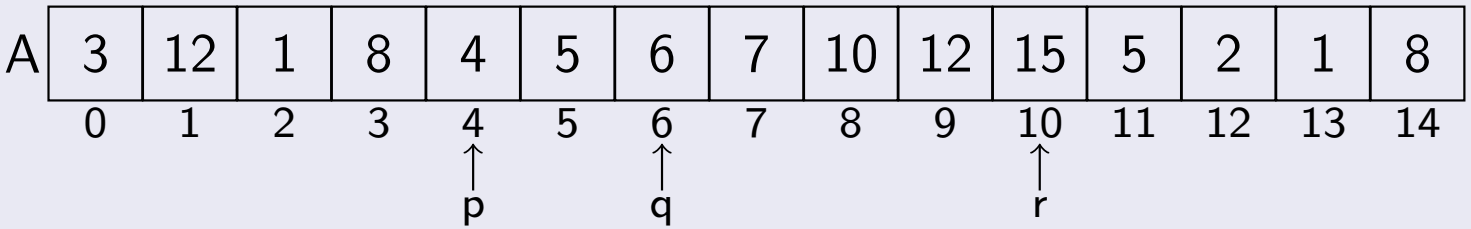




`A[k]` recebe o menor elemento dentre os 2 sub-arranjos

Intercalação: Exemplo

Considere o arranjo A e os índices p, q e r:



E `A[p..r]` está ordenado

MergeSort

Vejamos agora o MergeSort completo

```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```

MergeSort

Vejamos agora o MergeSort completo

```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```

MergeSort

Vejamos agora o MergeSort completo

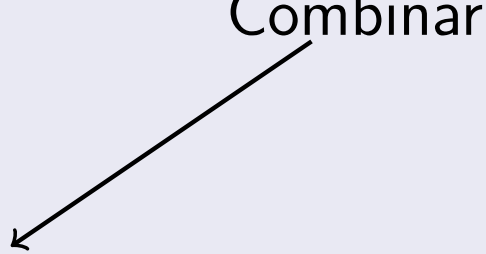
```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```

Conquistar

MergeSort

Vejamos agora o MergeSort completo

```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```



MergeSort

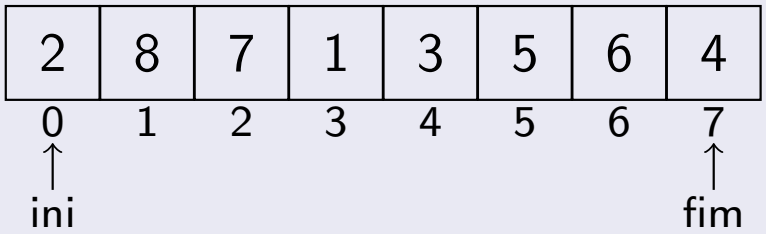
Código

```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                        fim);

    merge(numeros, ini, meio,
                        fim);
  }
}
```

MergeSort

Código

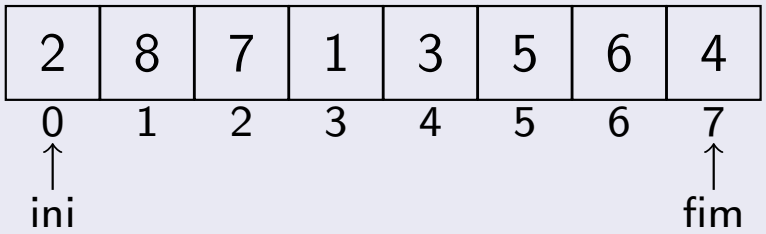
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

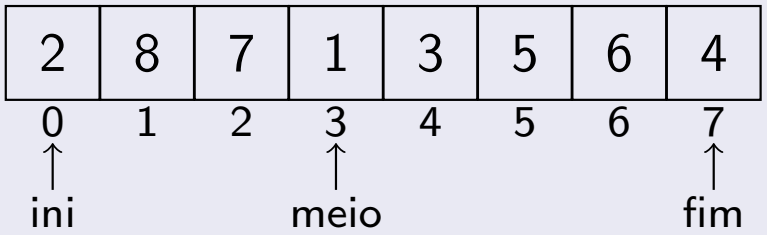
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

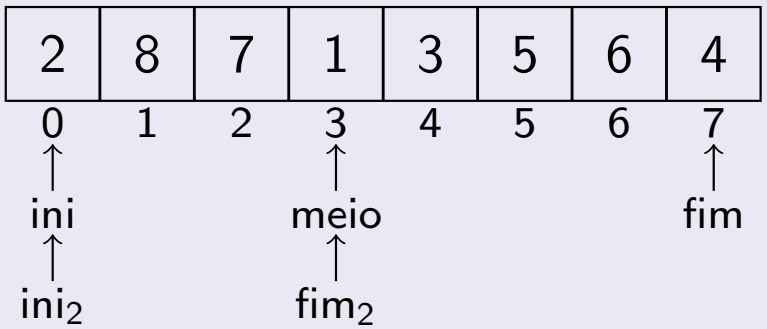
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

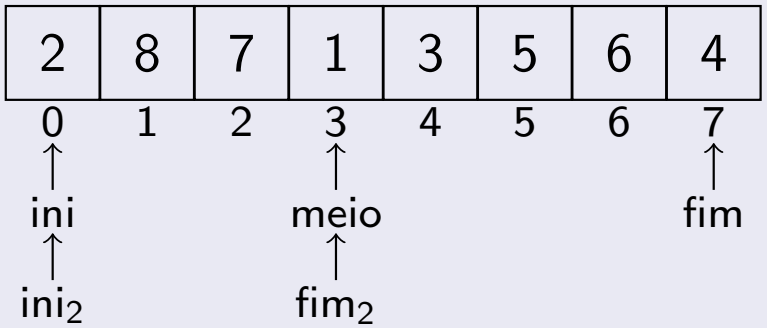
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                           fim);

    merge(numeros, ini, meio,
                           fim);
  }
}
```



MergeSort

Código

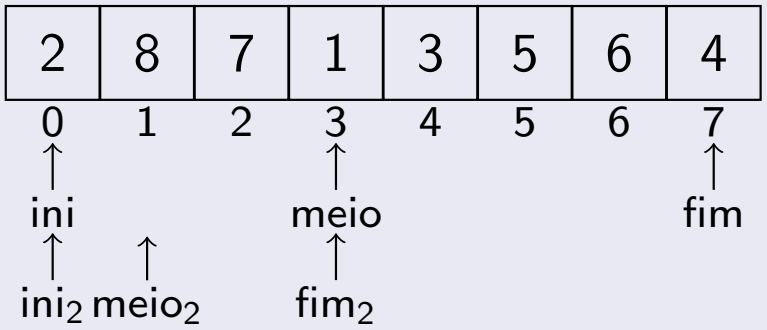
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

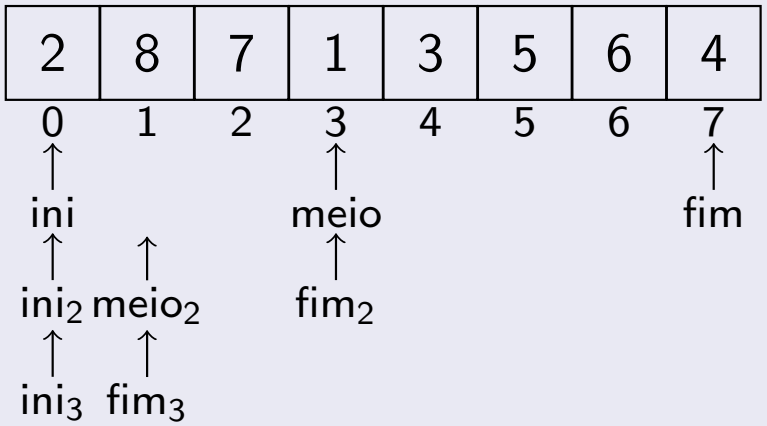
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

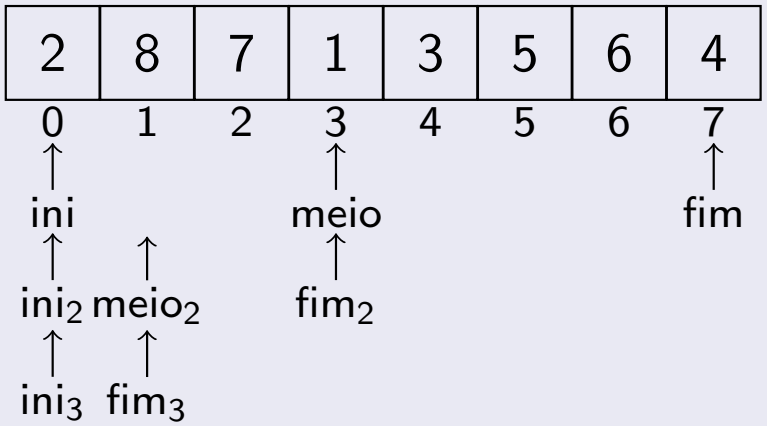
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

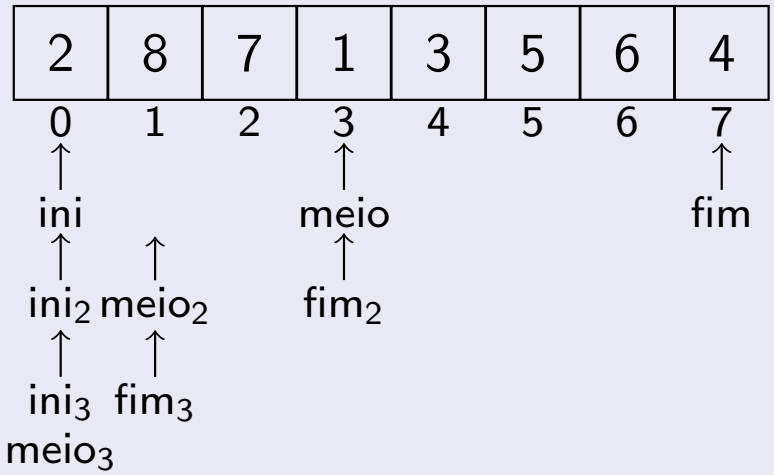
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

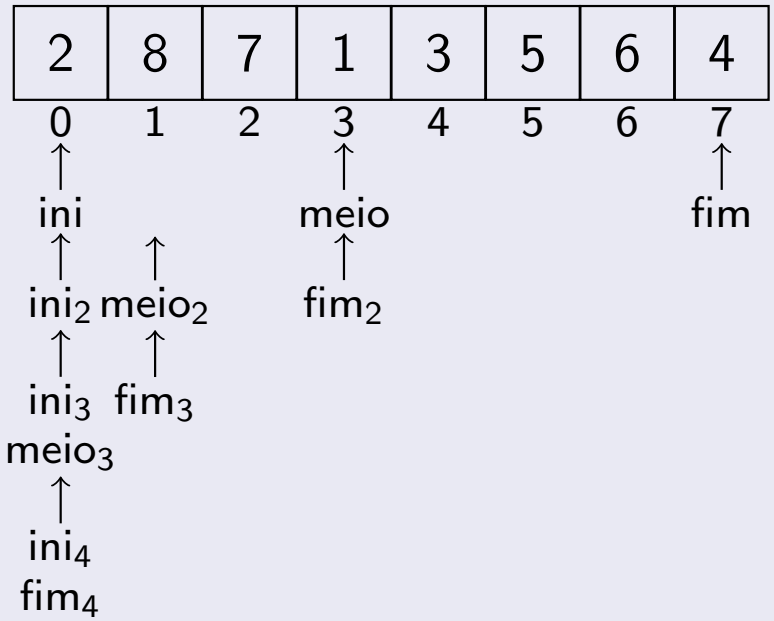
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

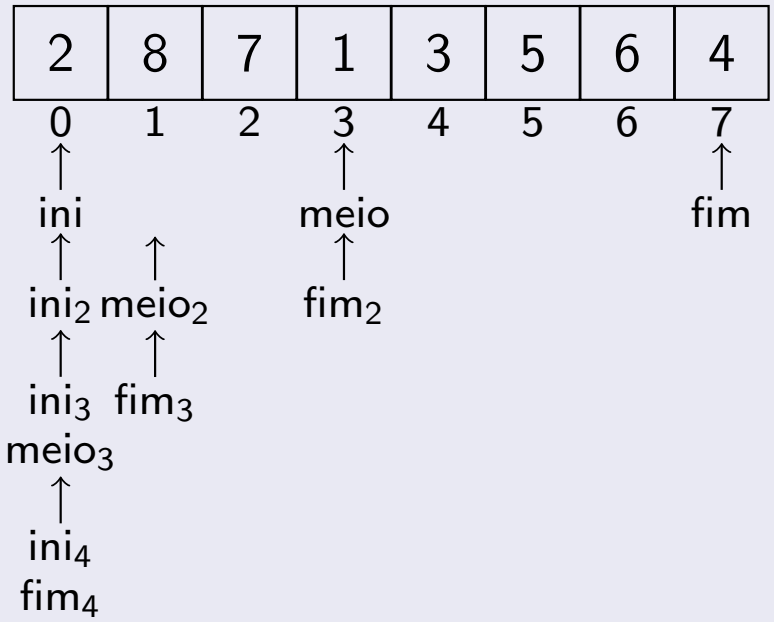
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

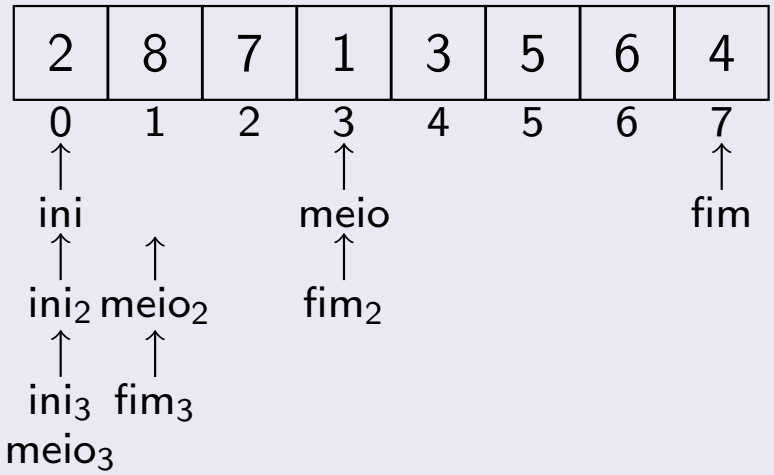
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

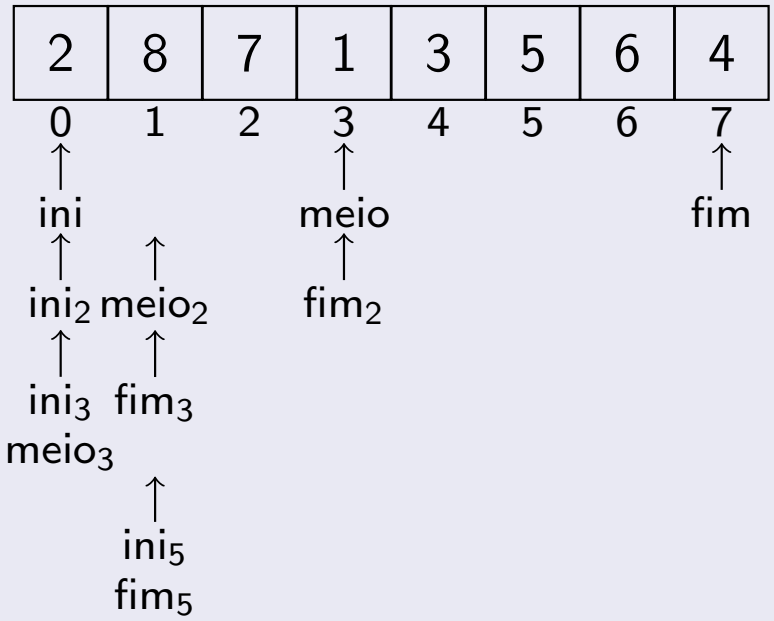
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

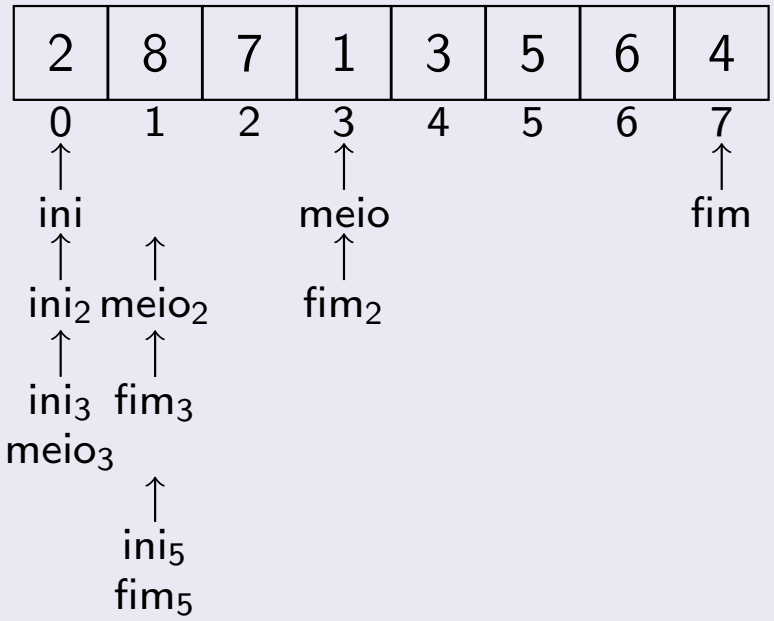
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

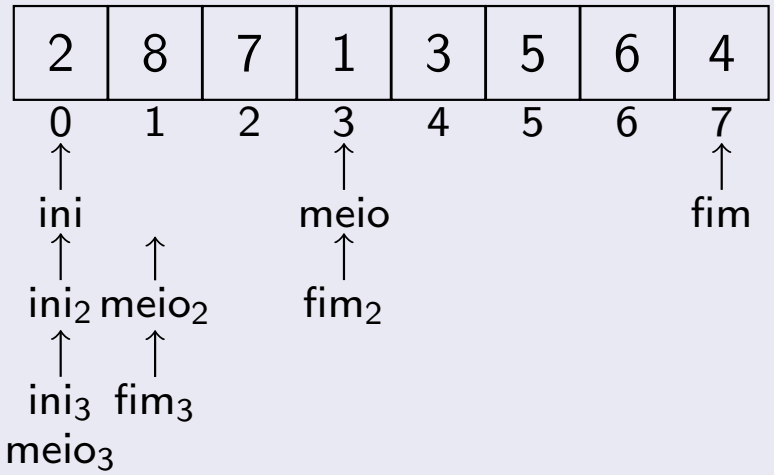
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

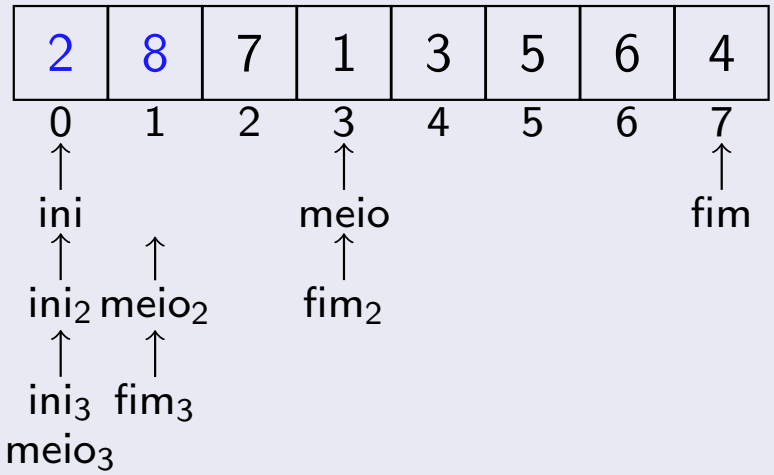
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

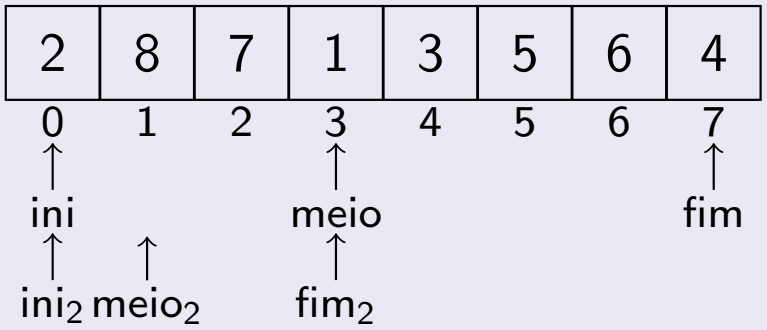
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

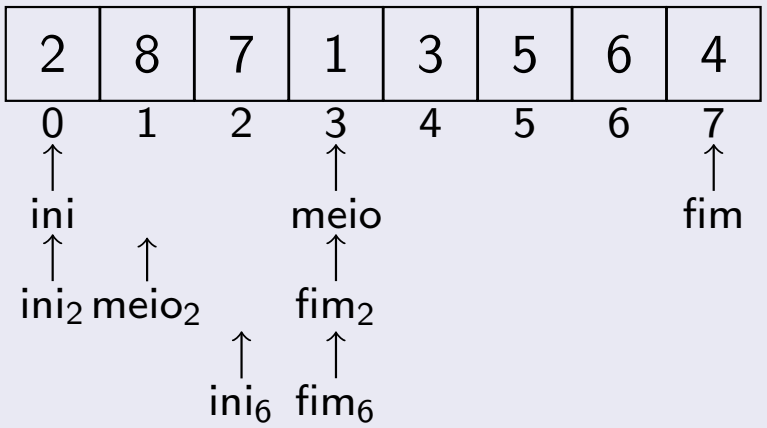
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

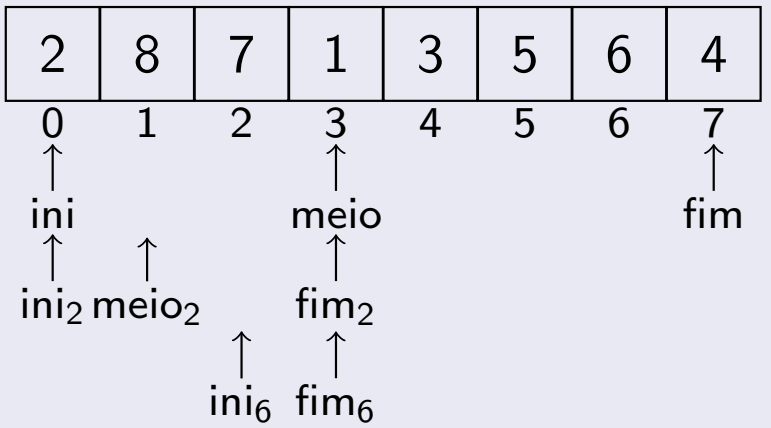
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

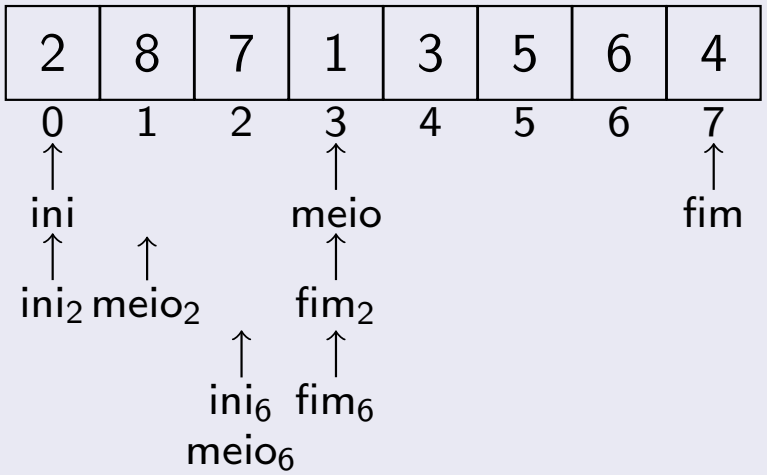
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

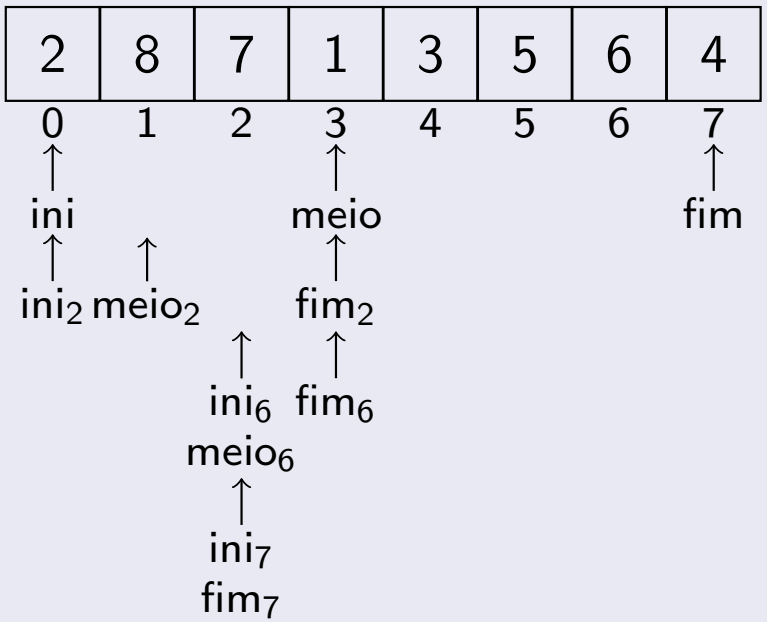
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

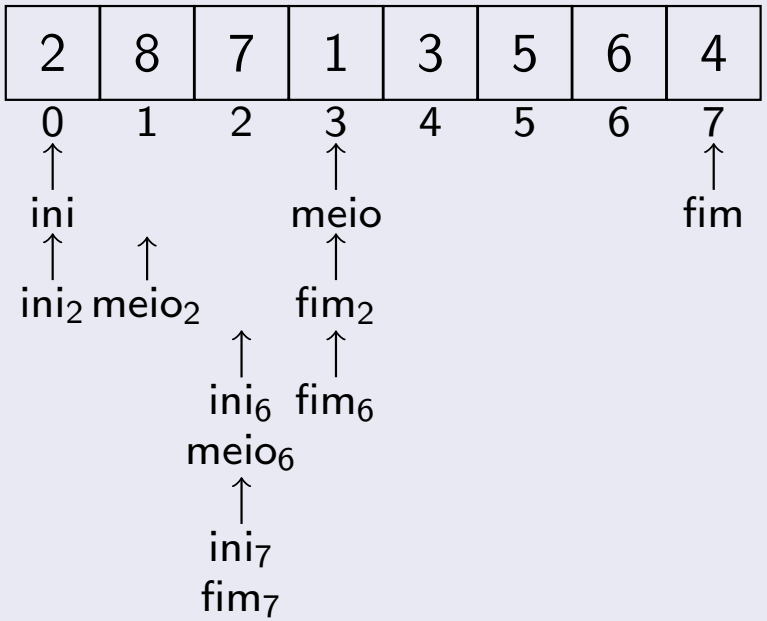
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

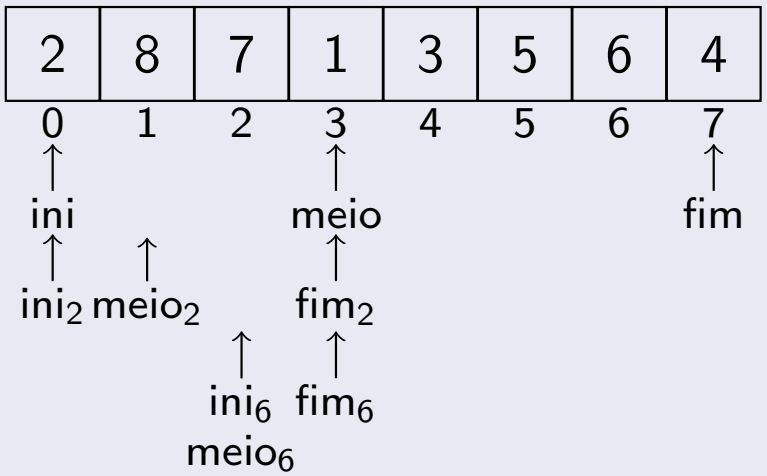
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

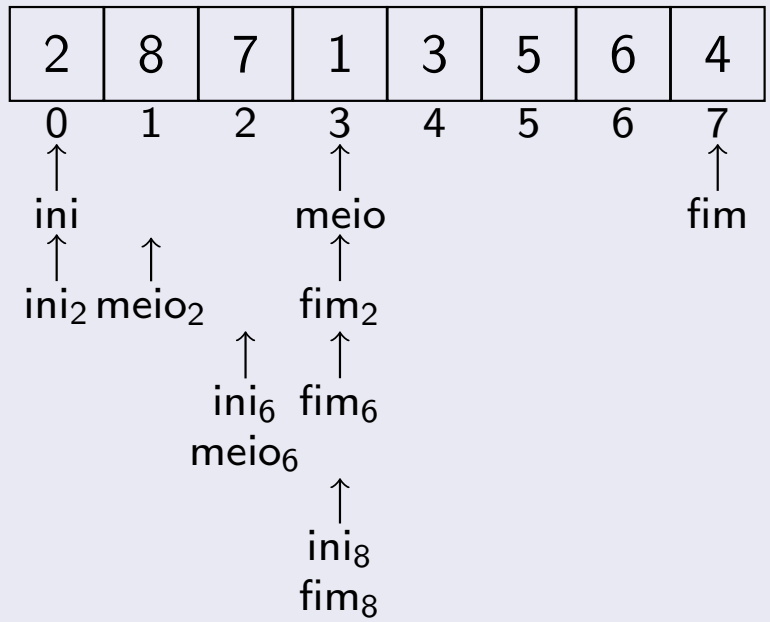
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

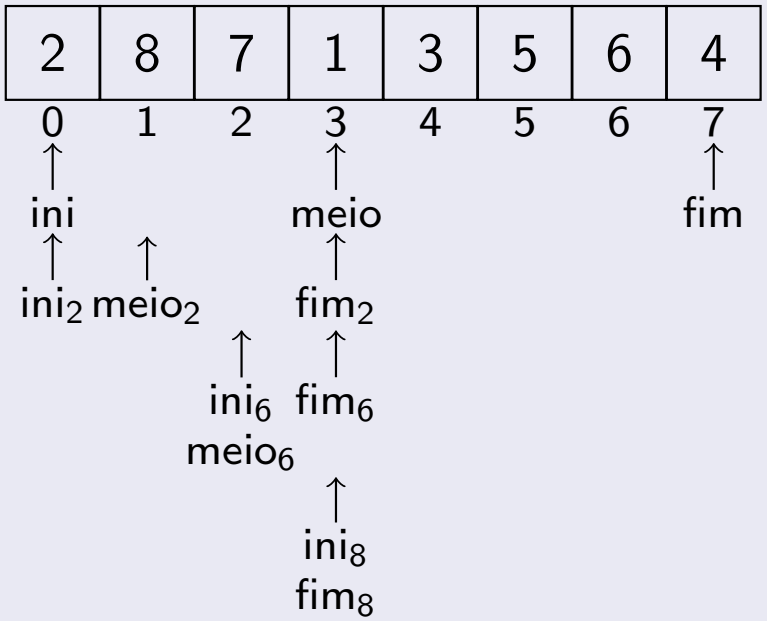
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

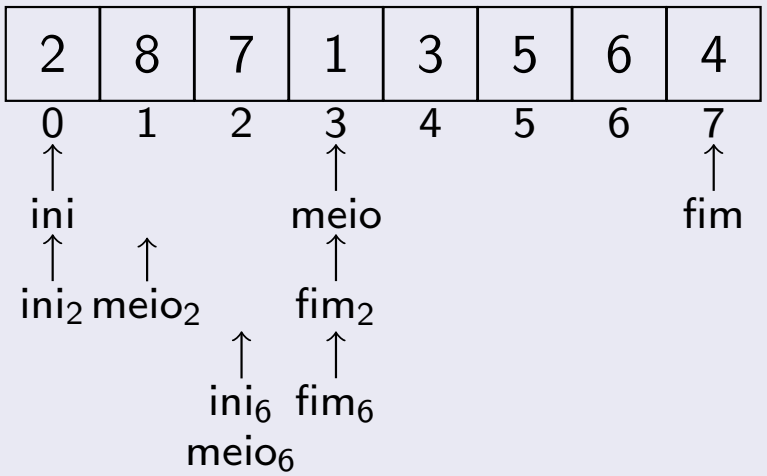
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

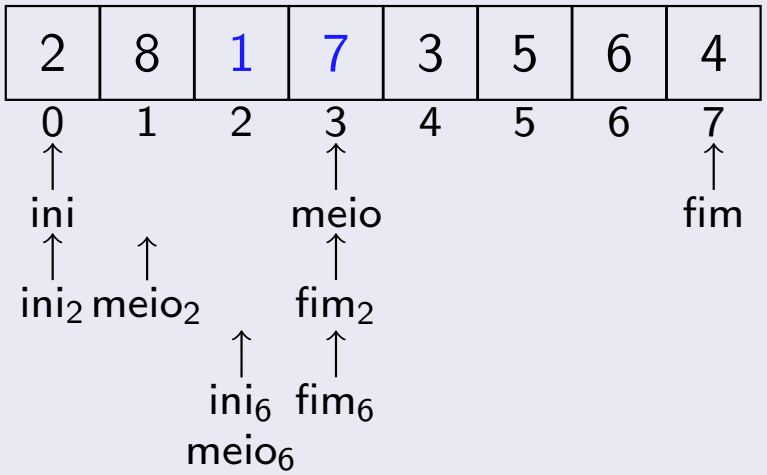
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

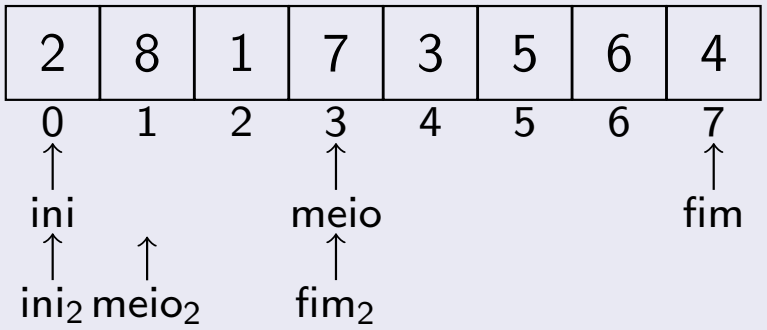
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

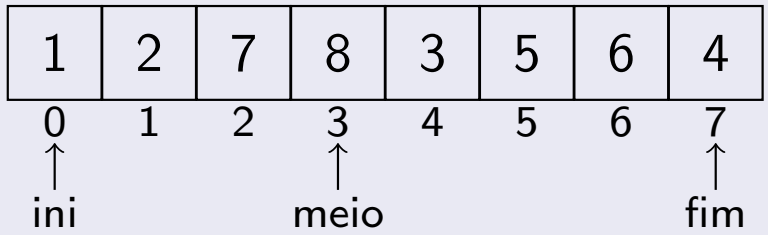
Código

```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```

MergeSort

Código

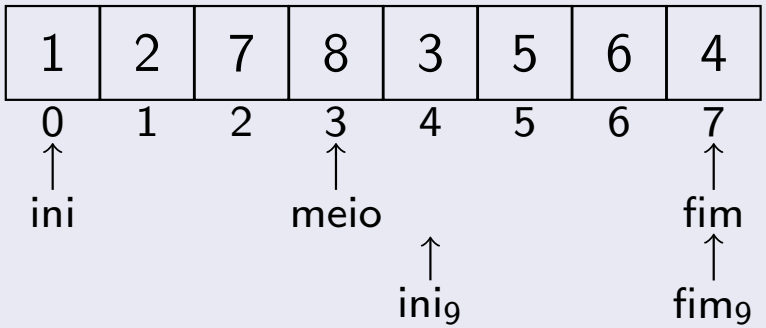
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

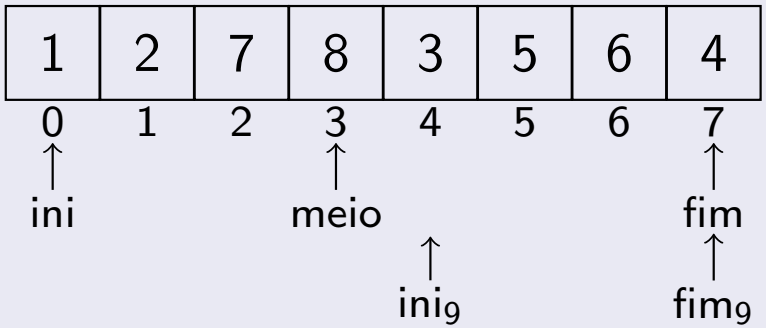
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

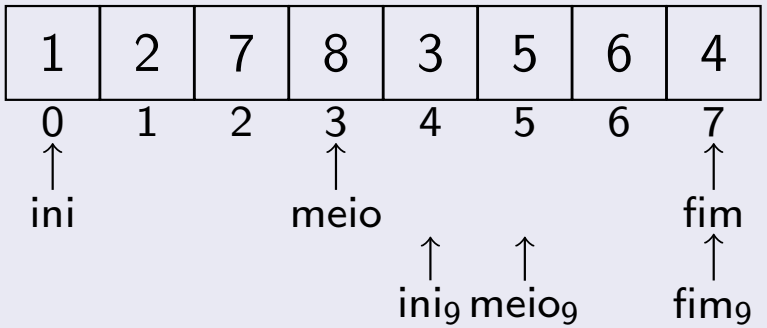
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

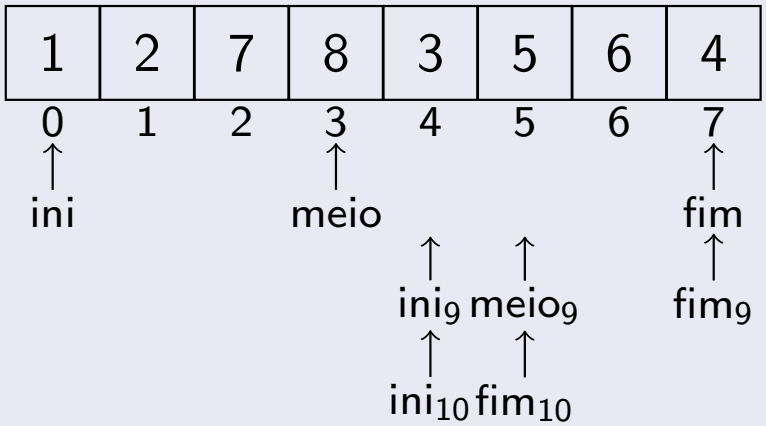
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

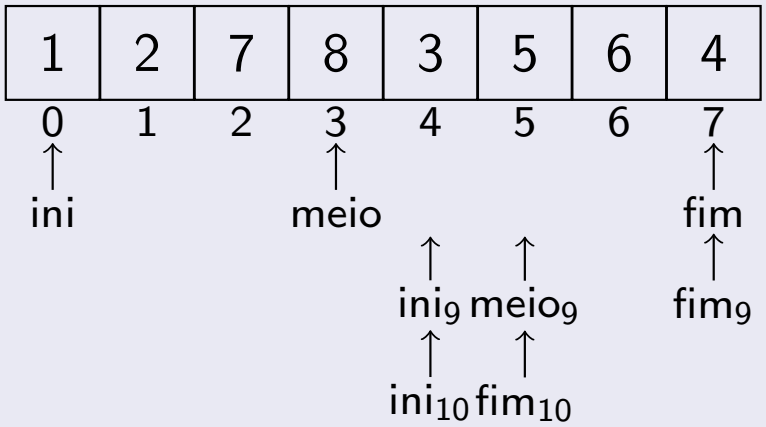
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                           fim);

    merge(numeros, ini, meio,
                           fim);
  }
}
```



MergeSort

Código

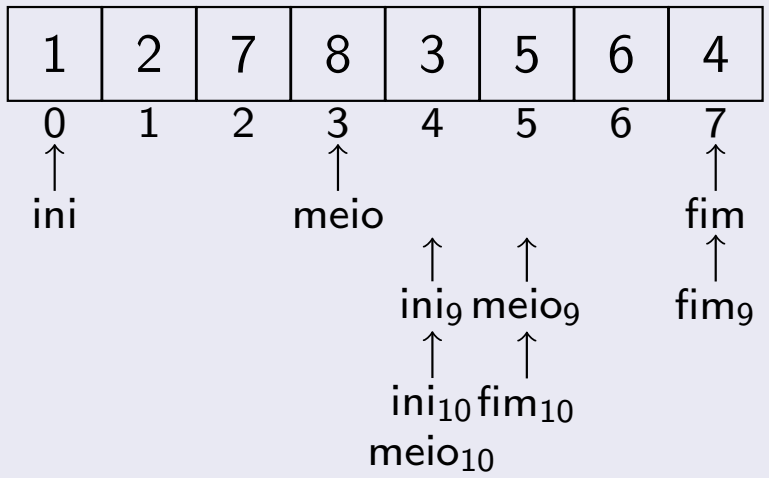
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

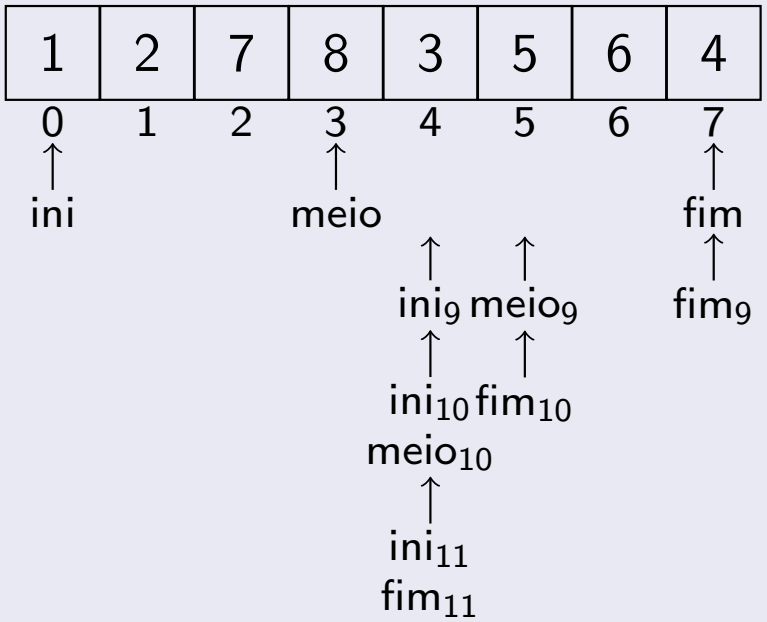
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

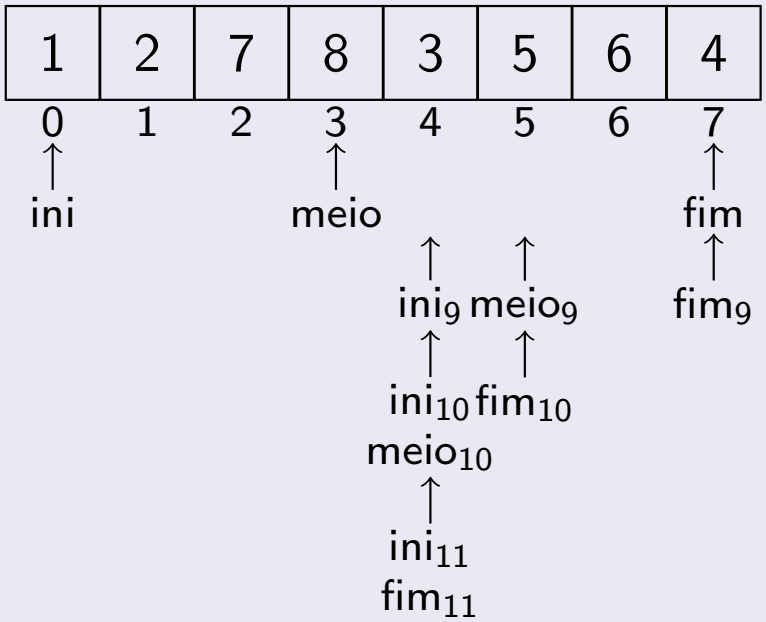
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

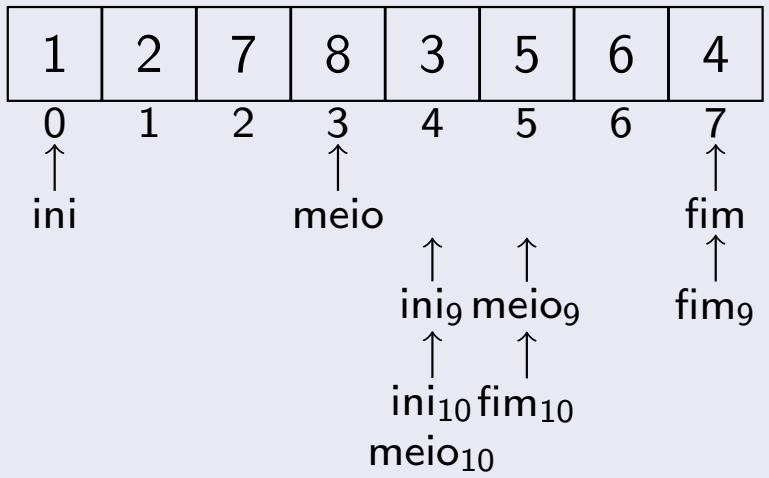
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

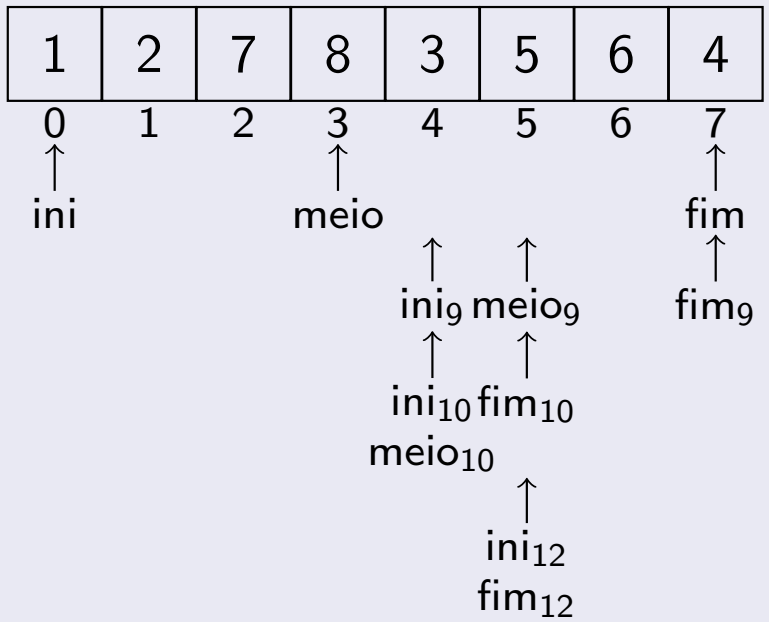
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

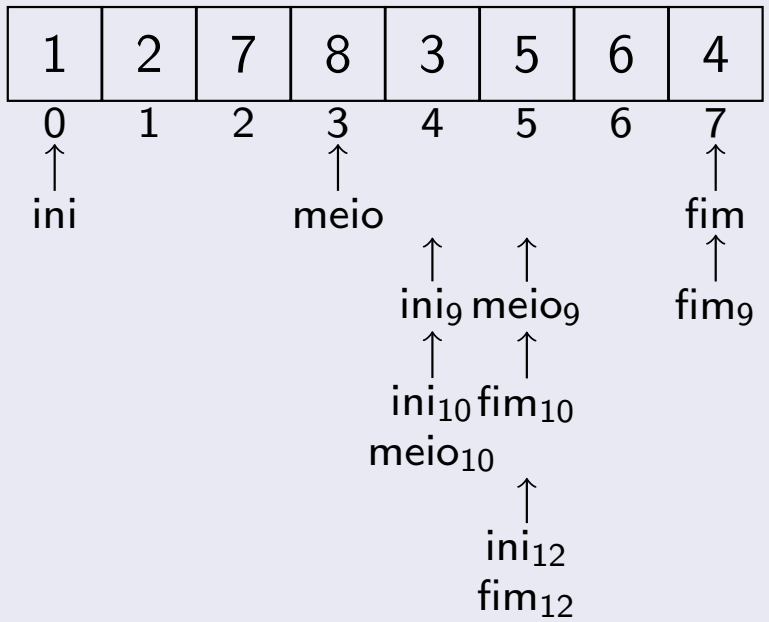
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

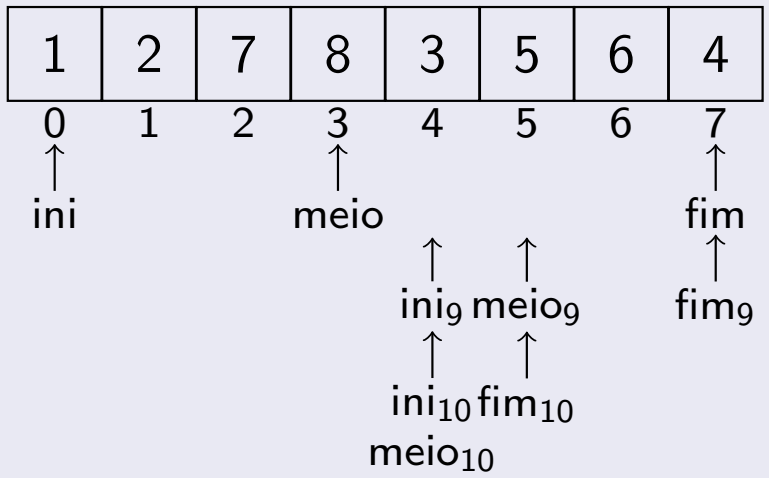
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

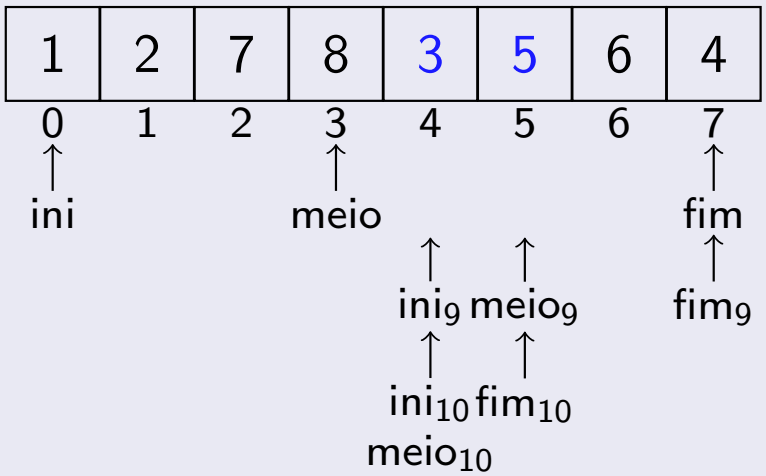
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

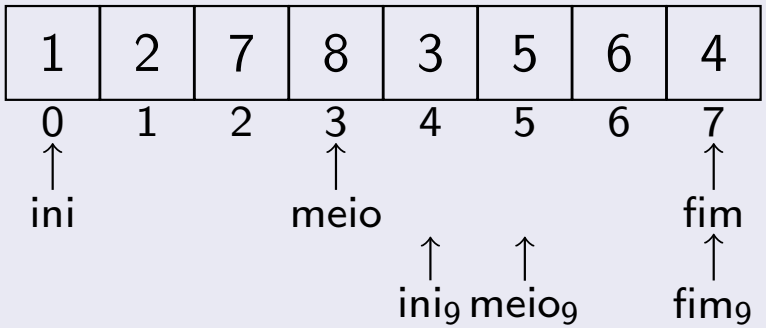
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

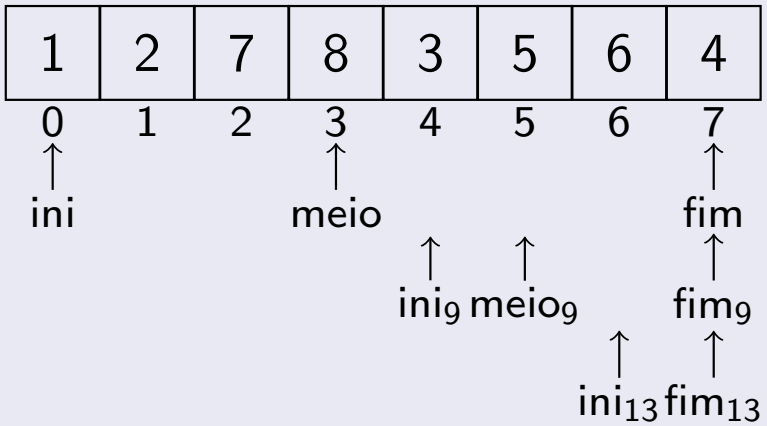
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

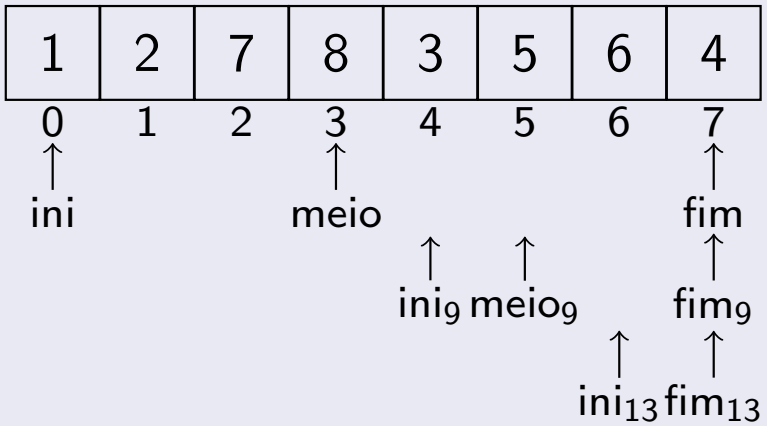
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

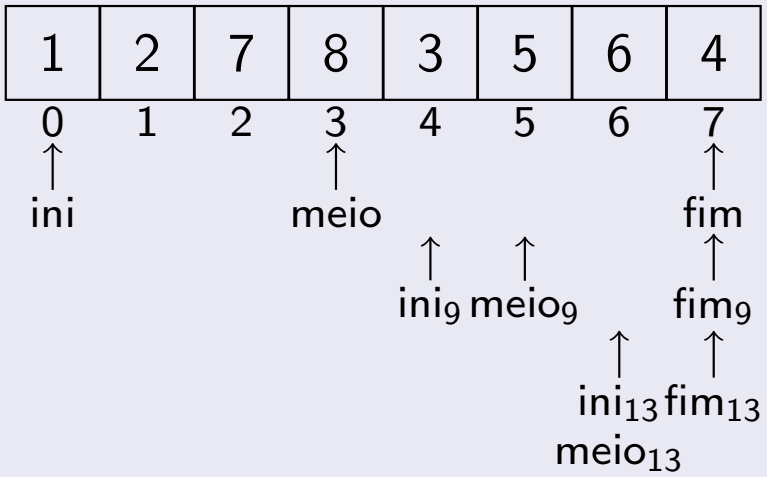
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

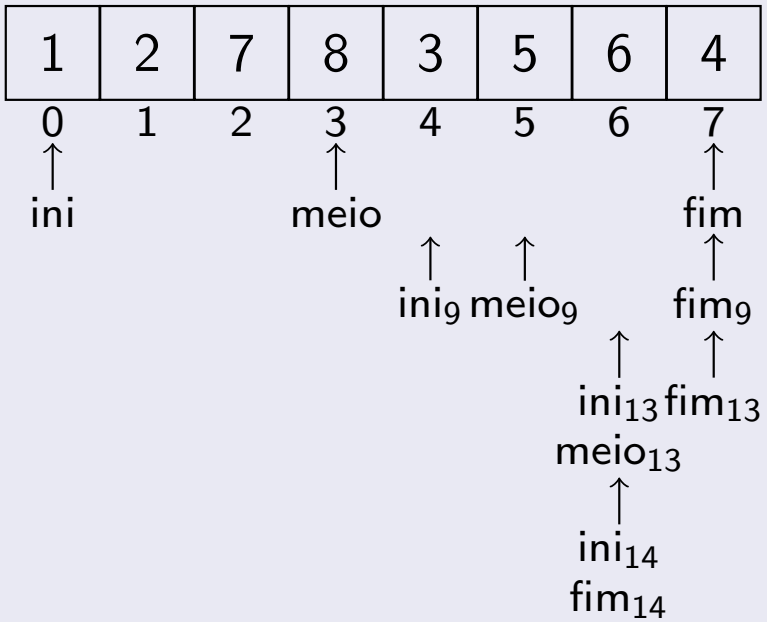
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

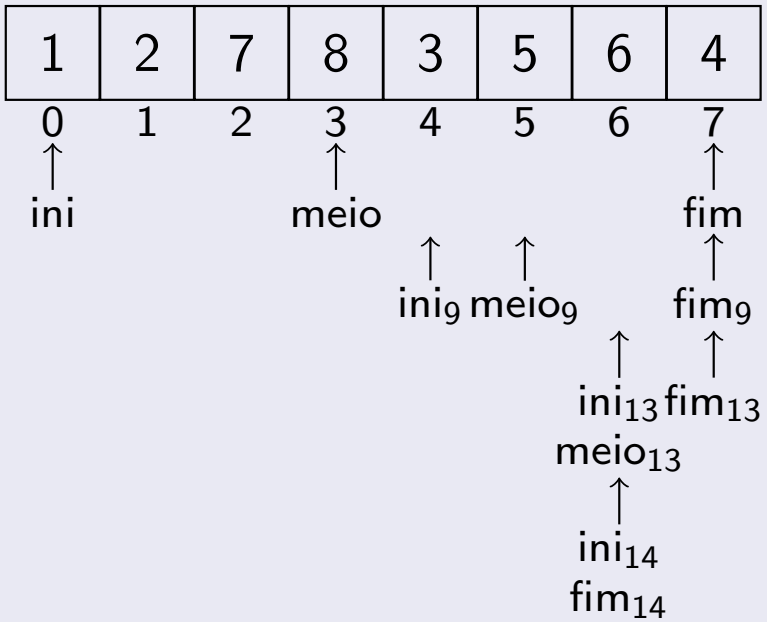
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

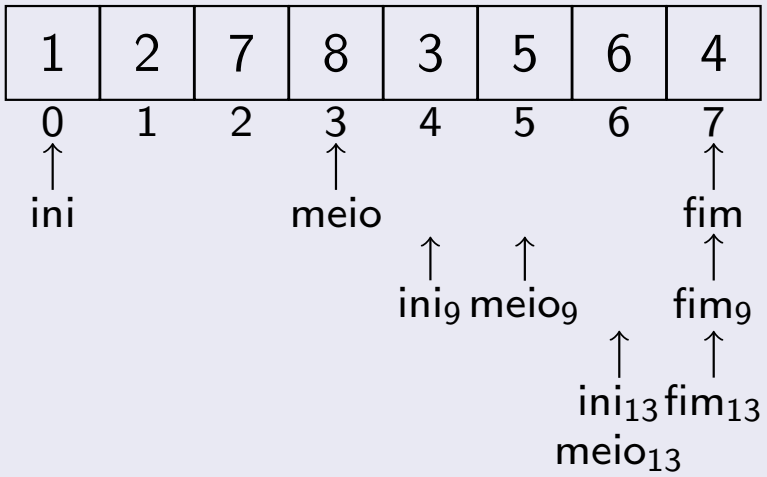
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

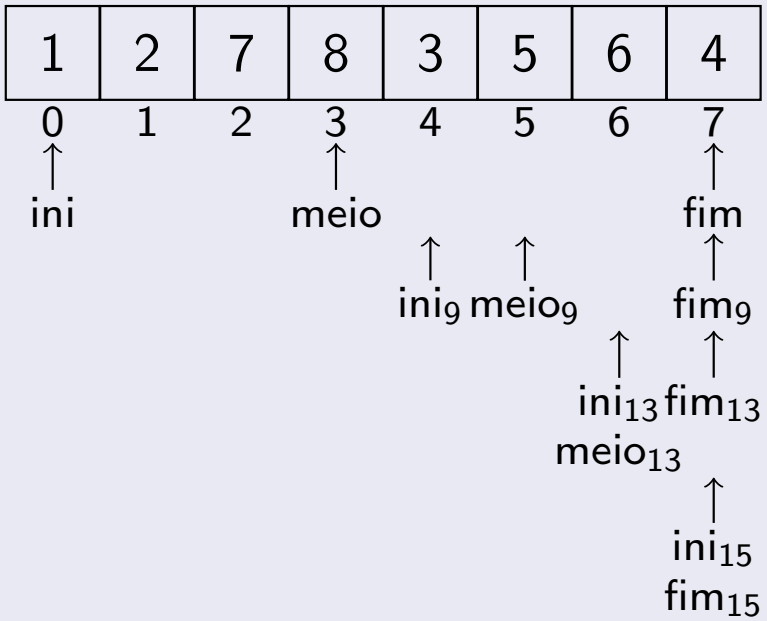
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

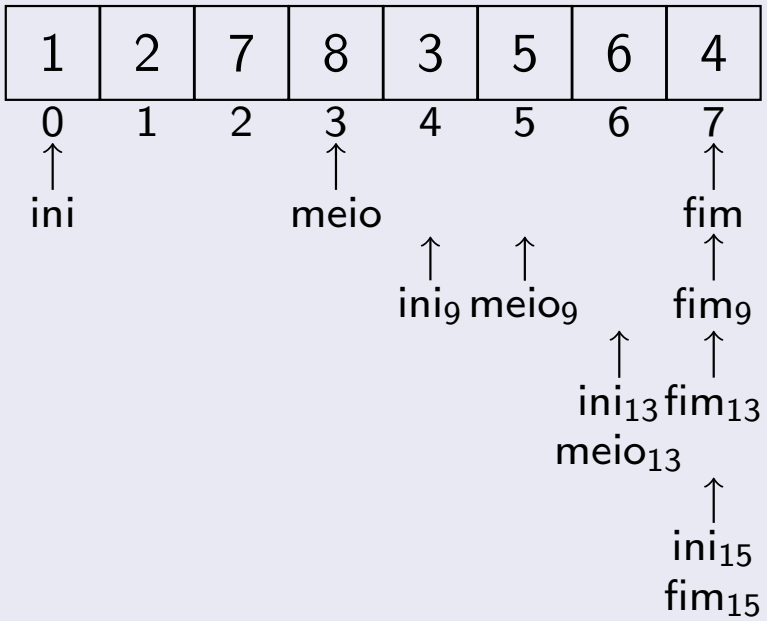
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

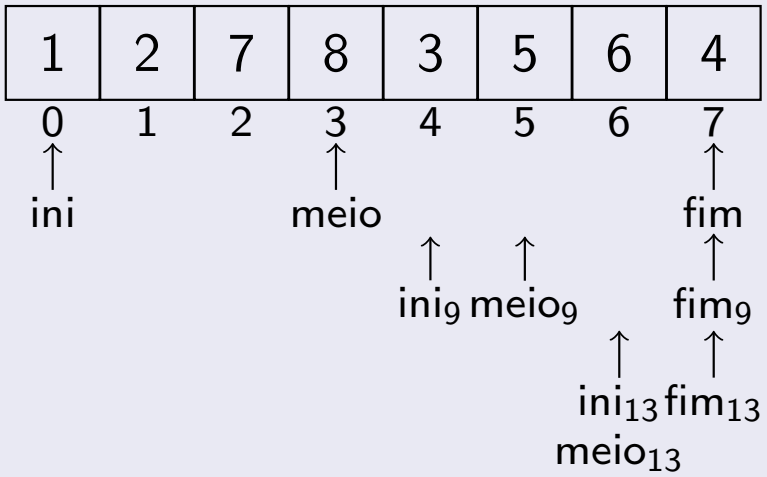
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

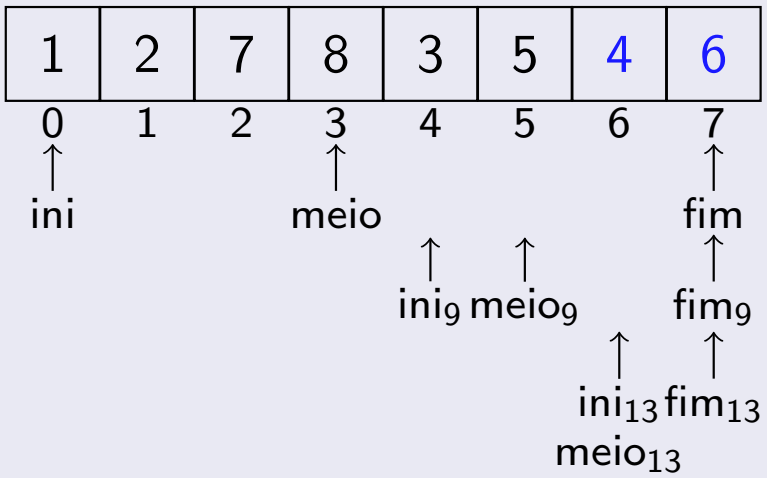
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

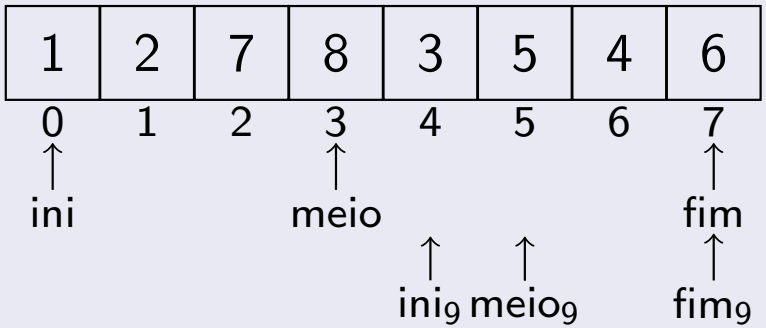
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

Código

```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                          meio);
    mergeSort(numeros, meio+1,
                          fim);

    merge(numeros, ini, meio,
                          fim);
  }
}
```



MergeSort

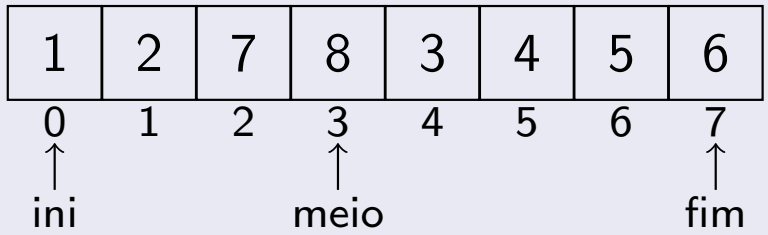
Código

```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```

MergeSort

Código

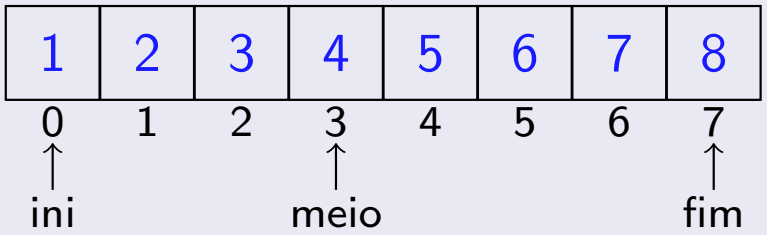
```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Código

```
void mergeSort(int numeros[],
               int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini,
                       meio);
    mergeSort(numeros, meio+1,
                       fim);

    merge(numeros, ini, meio,
                       fim);
  }
}
```



MergeSort

Complexidade da Fusão

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];
```

- Qual a complexidade do `merge`?

MergeSort

Complexidade da Fusão

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];
```

- Qual a complexidade do `merge`?
  - $\Theta(n)$

MergeSort

Complexidade da Fusão

```
void merge(int A[], int p, int q, int r) {
  int i, j, k;
  int tamseq1 = q - p + 1;
  int tamseq2 = r - q;

  int seq1[tamseq1];
  for(i=0; i < tamseq1; i++)
    seq1[i] = A[p+i];

  int seq2[tamseq2];
  for(j=0; j < tamseq2; j++)
    seq2[j] = A[q+1+j];
```

- Qual a complexidade do `merge`?
  - $\Theta(n)$
  - $+\Theta(n)$

Complexidade da Fusão

```
k = p; i = 0; j = 0;
while (i < tamseq1 && j < tamseq2) {
  if(seq2[j] < seq1[i]) {
    A[k] = seq2[j];
    j++;
  }
  else {
    A[k] = seq1[i];
    i++;
  }
  k++;
}
```

Complexidade da Fusão

```
k = p; i = 0; j = 0;
while (i < tamseq1 && j < tamseq2) {
  if(seq2[j] < seq1[i]) {
    A[k] = seq2[j];
    j++;
  }
  else {
    A[k] = seq1[i];
    i++;
  }
  k++;
}
```

- $+\Theta(n)$

Complexidade da Fusão

```
  while (i < tamseq1) {
    A[k] = seq1[i];
    k++;
    i++;
  }

  while (j < tamseq2) {
    A[k] = seq2[j];
    k++;
    j++;
  }
}
```

Complexidade da Fusão

```
  while (i < tamseq1) {
    A[k] = seq1[i];
    k++;
    i++;
  }

  while (j < tamseq2) {
    A[k] = seq2[j];
    k++;
    j++;
  }
}
```

- $+\Theta(n)$

MergeSort

Complexidade da Fusão

```
  while (i < tamseq1) {
    A[k] = seq1[i];
    k++;
    i++;
  }

  while (j < tamseq2) {
    A[k] = seq2[j];
    k++;
    j++;
  }
}
```

- $+\Theta(n)$
- Ou seja, `merge`, no total, é $\Theta(n)$

Complexidade do MergeSort

```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```

- E do MergeSort?

MergeSort

Complexidade do MergeSort

```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```

- E do MergeSort?
  - $T(n/2)$

MergeSort

Complexidade do MergeSort

```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```

- E do MergeSort?
  - $T(n/2)$
  - $+T(n/2)$

MergeSort

Complexidade do MergeSort

```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```

- E do MergeSort?
  - $T(n/2)$
  - $+T(n/2)$
  - $+\Theta(n)$

MergeSort

Complexidade do MergeSort

```
void mergeSort(int numeros[], int ini, int fim) {
  int meio;
  if(ini < fim) {
    meio = (ini + fim)/2;

    mergeSort(numeros, ini, meio);
    mergeSort(numeros, meio+1, fim);

    merge(numeros, ini, meio, fim);
  }
}
```

- E do MergeSort?
  - $T(n/2)$
  - $+T(n/2)$
  - $+\Theta(n)$

- Então

$$T(n) = \begin{cases} O(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n) & \text{para } n \geq 2 \end{cases}$$

MergeSort

Complexidade do MergeSort

- Pelo Teorema Mestre,

$$T(n) = \begin{cases} O(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n) & \text{para } n \geq 2 \end{cases}$$

cai no caso 2, fazendo com que $T(n) = \Theta(n \log n)$

Complexidade do MergeSort

- É possível fazer a intercalação dos sub-arranjos ordenados sem o uso de arranjos auxiliares?

Complexidade do MergeSort

- É possível fazer a intercalação dos sub-arranjos ordenados sem o uso de arranjos auxiliares?
  - Sim! Basta deslocarmos os elementos de um dos sub-arranjos, quando necessário, para dar lugar ao mínimo dos dois sub-arranjos

Complexidade do MergeSort

- É possível fazer a intercalação dos sub-arranjos ordenados sem o uso de arranjos auxiliares?
  - Sim! Basta deslocarmos os elementos de um dos sub-arranjos, quando necessário, para dar lugar ao mínimo dos dois sub-arranjos
  - No entanto, a etapa de intercalação passa a ter complexidade $\Theta(n^2)$, resultando na recorrência

$$T(n) = \begin{cases} O(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n^2) & \text{para } n \geq 2 \end{cases}$$

Complexidade do MergeSort

$$T(n) = \begin{cases} O(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n^2) & \text{para } n \geq 2 \end{cases}$$

Complexidade do MergeSort

$$T(n) = \begin{cases} O(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n^2) & \text{para } n \geq 2 \end{cases}$$

- Pelo Teorema Mestre (caso 3), $T(n) \in \Theta(n^2)$

Complexidade do MergeSort

$$T(n) = \begin{cases} O(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n^2) & \text{para } n \geq 2 \end{cases}$$

- Pelo Teorema Mestre (caso 3), $T(n) \in \Theta(n^2)$
- Ou seja, a complexidade do MergeSort passa a ser $\Theta(n^2)$

MergeSort

Complexidade do MergeSort

$$T(n) = \begin{cases} O(1) & \text{se } n = 1 \\ 2T(n/2) + \Theta(n^2) & \text{para } n \geq 2 \end{cases}$$

- Pelo Teorema Mestre (caso 3), $T(n) \in \Theta(n^2)$
- Ou seja, a complexidade do MergeSort passa a ser $\Theta(n^2)$
- Dessa forma, a eficiência da etapa de intercalação é crucial para a eficiência do MergeSort

MergeSort × Quicksort

Mergesort

- Melhor caso:
  - $O(n \log n)$
- Pior caso:
  - $O(n \log n)$
- Caso médio:
  - $O(n \log n)$
- Ordenação:
  - Não é local (*in place*). Necessita de arranjos auxiliares

Quicksort

- Melhor caso:
  - $O(n \log n)$
- Pior caso:
  - $O(n^2)$
- Caso médio:
  - $O(n \log n)$
- Ordenação:
  - Local (*in place*). Não necessita de arranjos auxiliares

Referências

- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms. 2a ed. MIT Press, 2001.

- Material baseado em slides dos professores Delano Beder e Marcos Chain

Aula 18 - MergeSort

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023