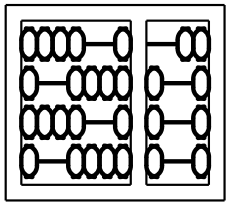


Universidade Estadual de Campinas - UNICAMP
Instituto de Computação - IC

Ordenação

Tópicos

1. Ordenação por intercalação (*Mergesort*)
2. Ordenação por *Quicksort*

- O algoritmo de ordenação por intercalação (mergesort)

– Intercalação:

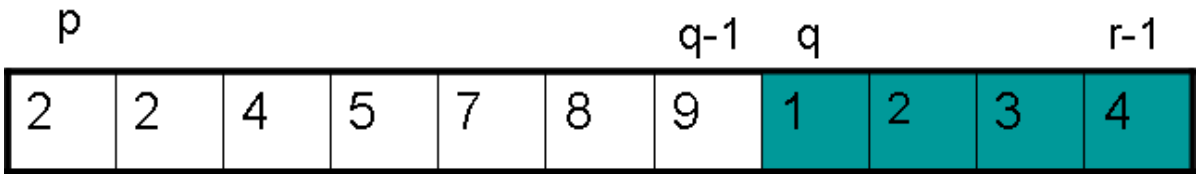
Problema: Dados dois vetores com elementos em ordem crescente, $v[p..q-1]$ e $v[q..r-1]$, rearranjá-los de modo que $v[p..r-1]$ esteja em ordem crescente.

- O algoritmo de ordenação por intercalação (mergesort)

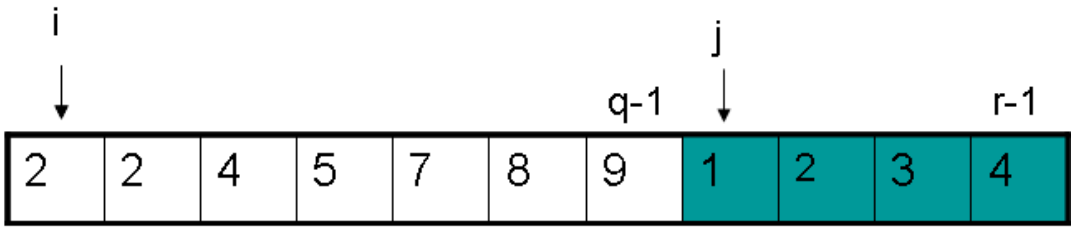
– Intercalação:

Problema: Dados dois vetores com elementos em ordem crescente, $v[p..q-1]$ e $v[q..r-1]$, rearranjá-los de modo que $v[p..r-1]$ esteja em ordem crescente.

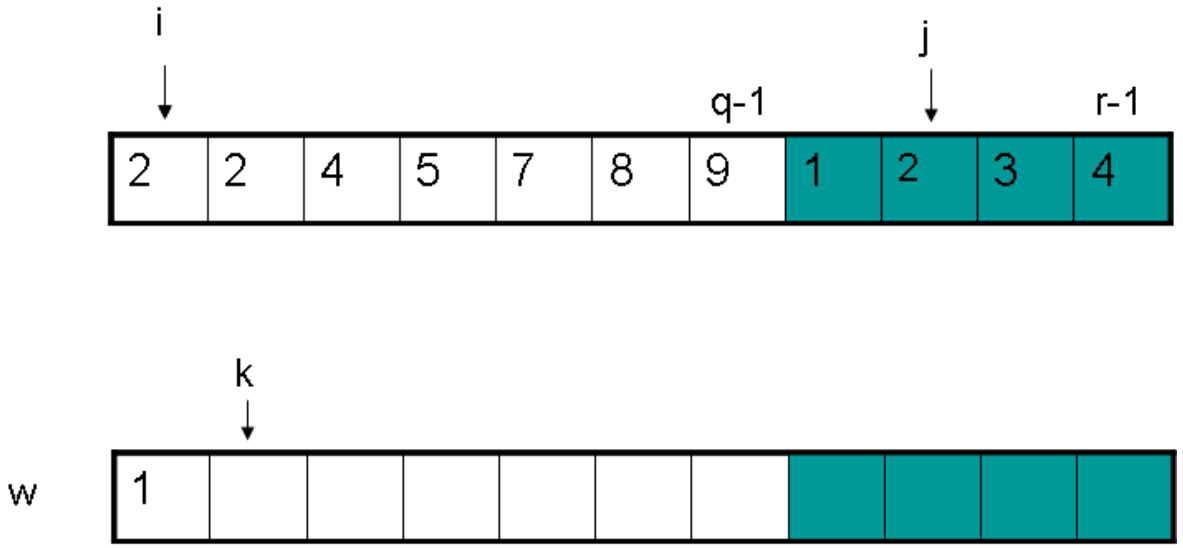
$\Rightarrow$ Duas metades estão ordenadas:



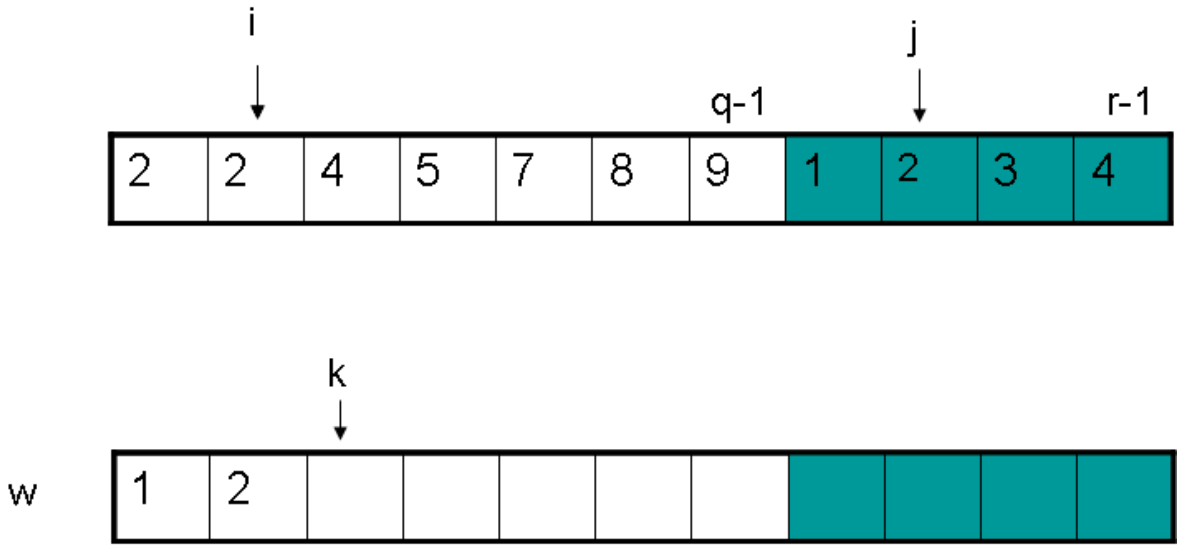
- Exemplo:



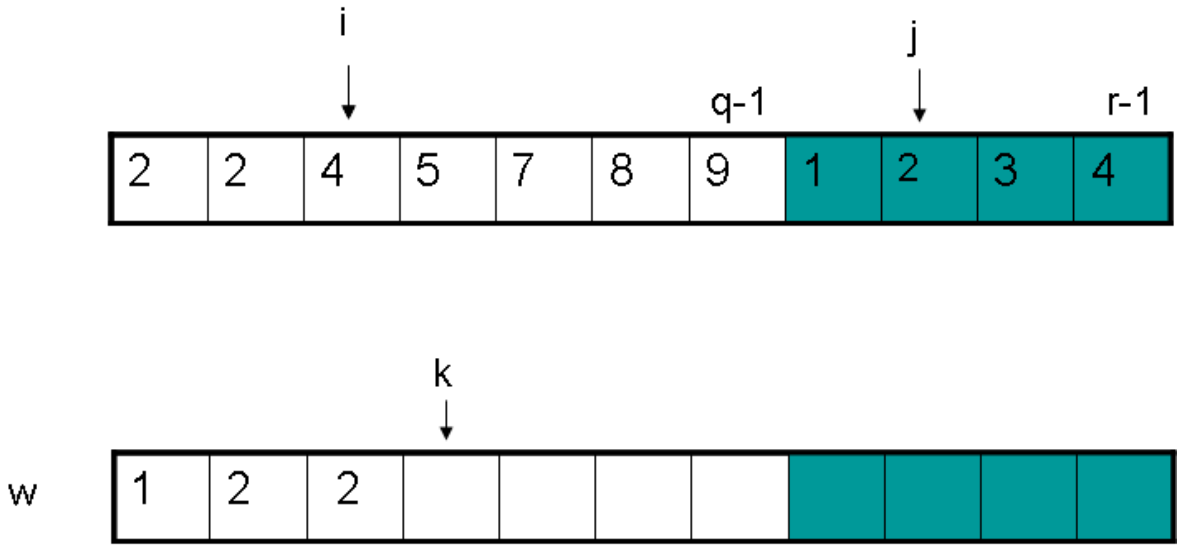
- Exemplo:



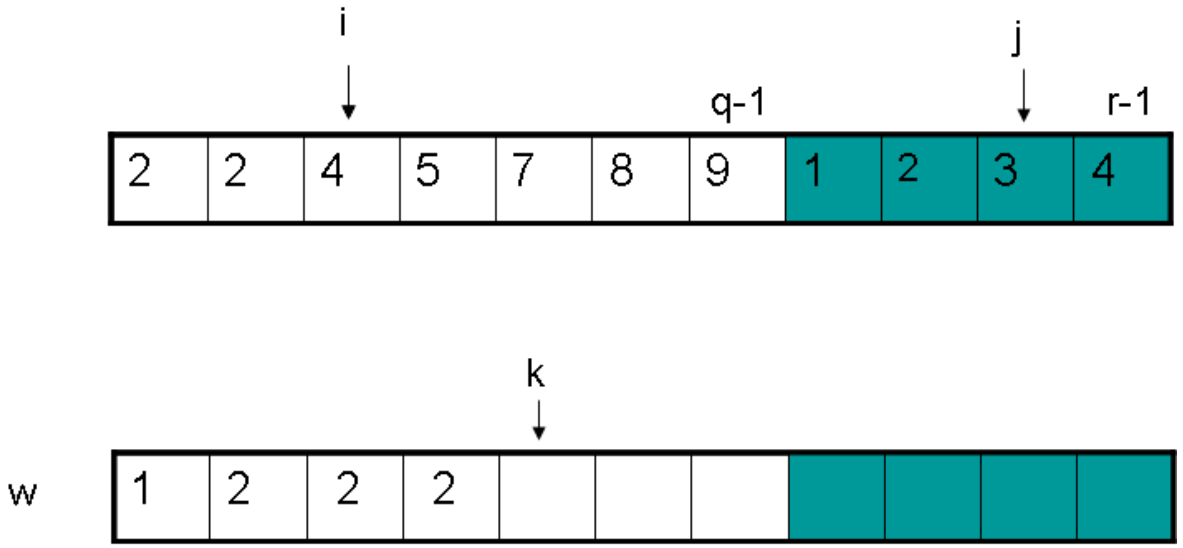
- Exemplo:



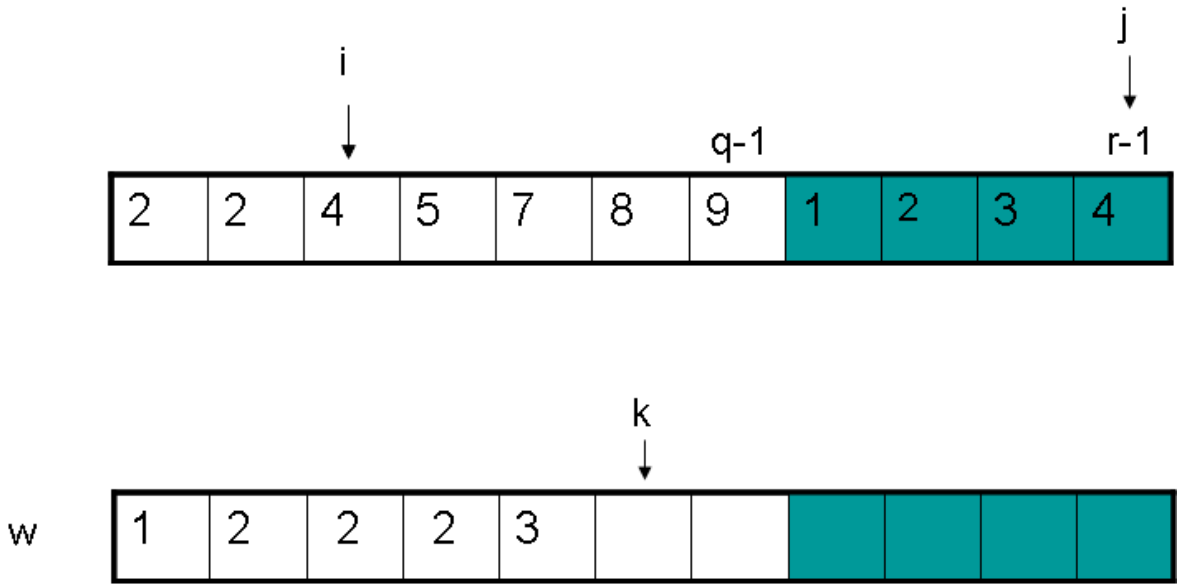
- Exemplo:



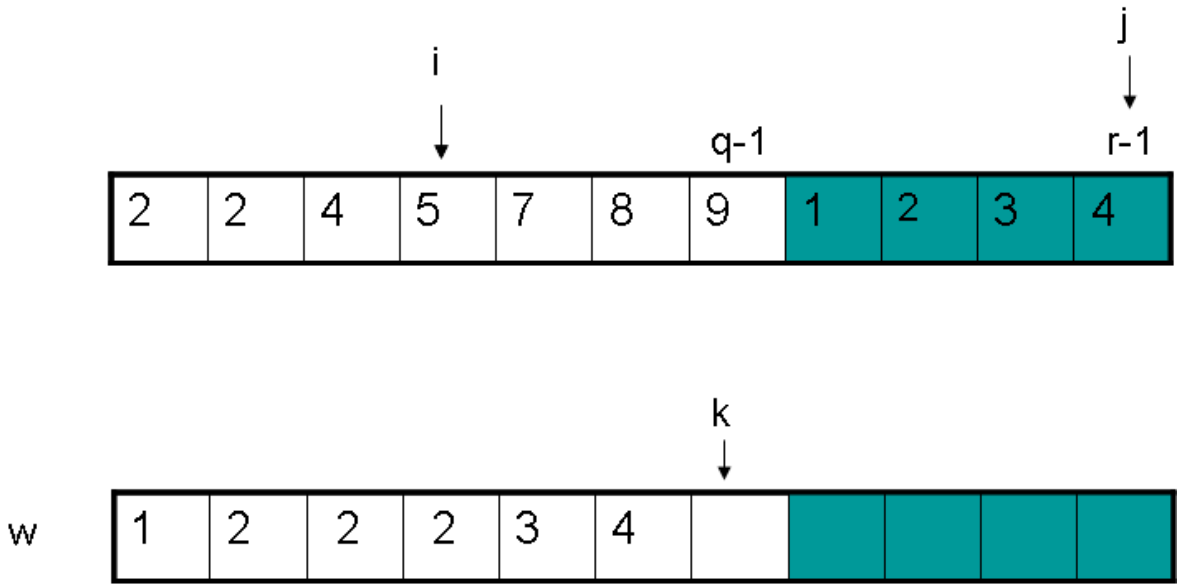
- Exemplo:



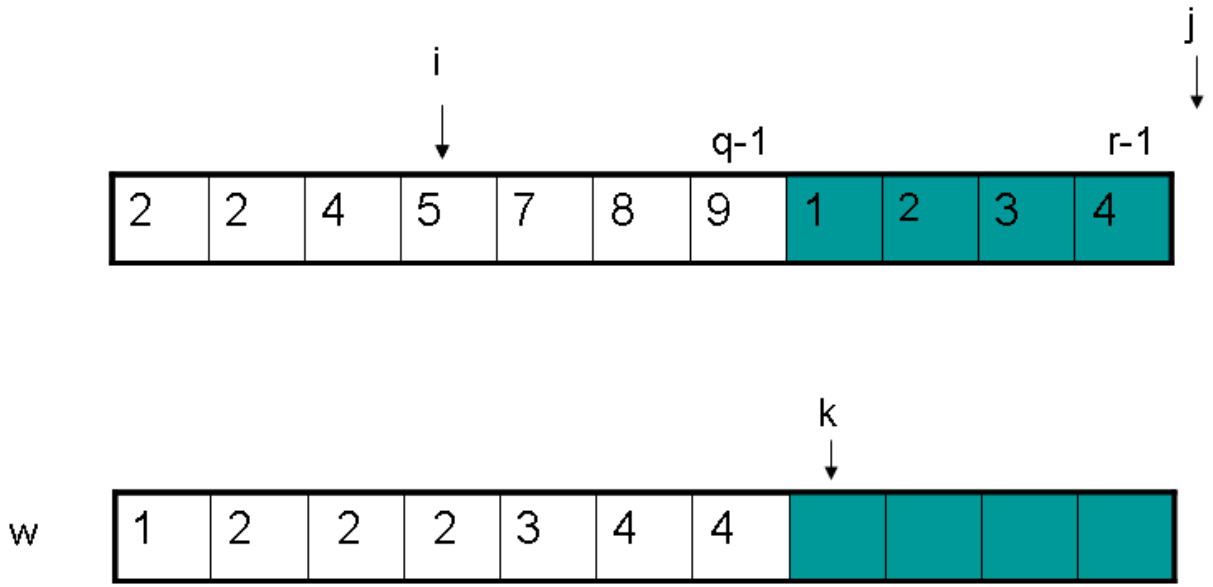
- Exemplo:



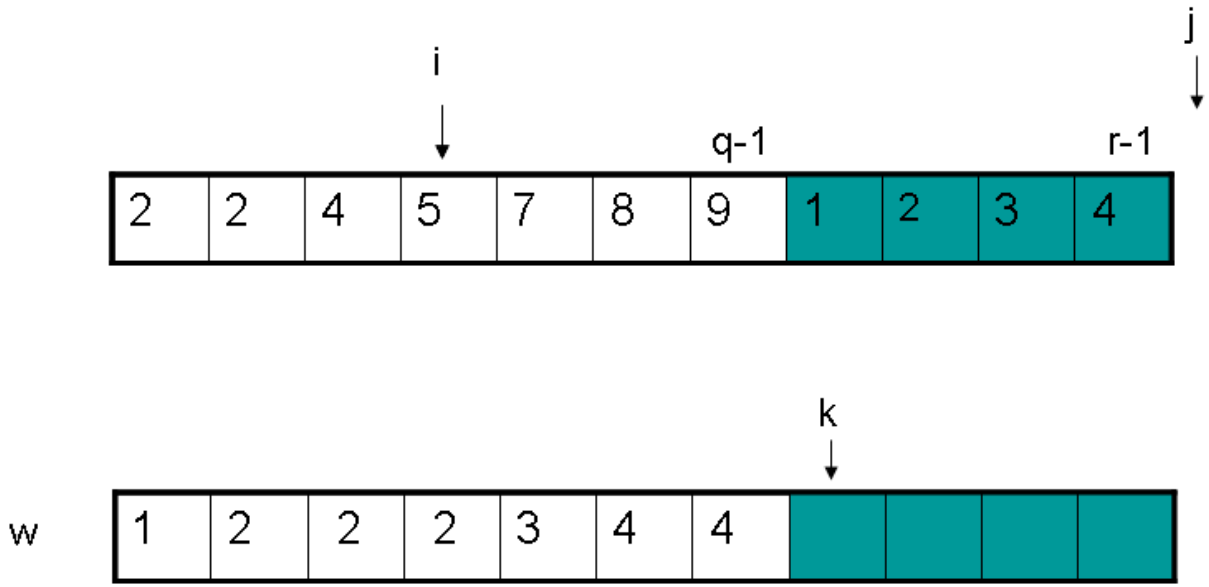
- Exemplo:



- Exemplo:

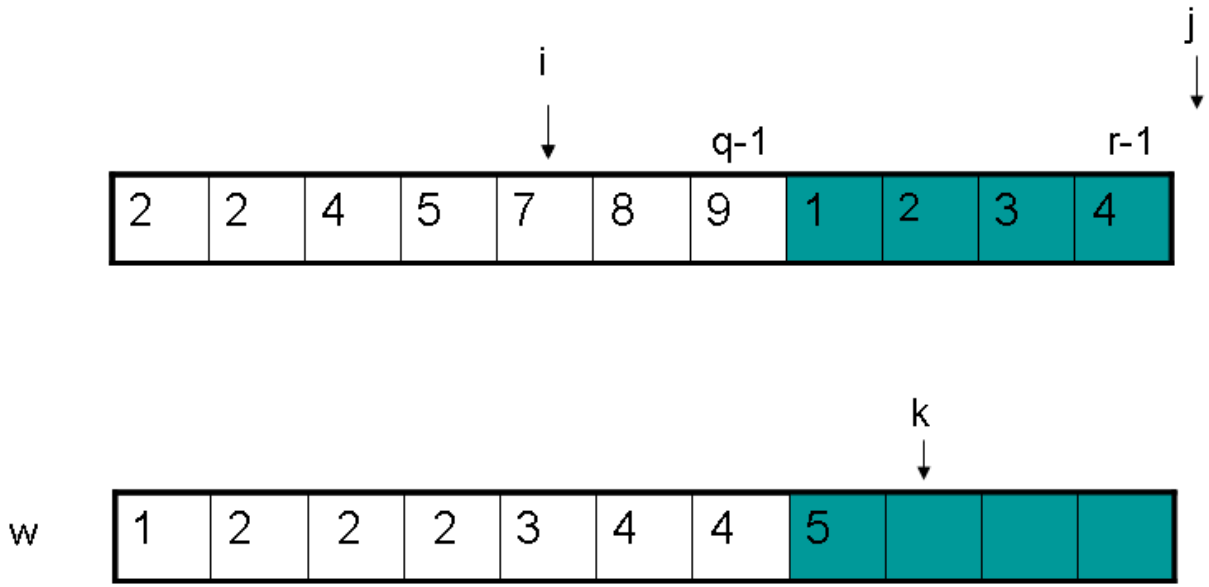


- Exemplo:



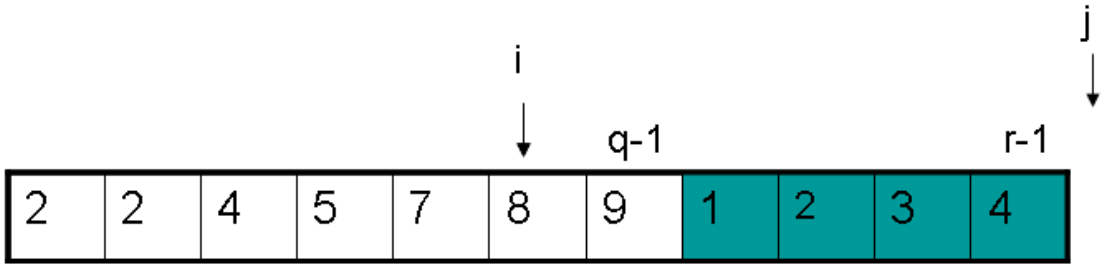
j == r → lista 2 terminou → transferir elementos restantes da lista 1 para o vetor w.

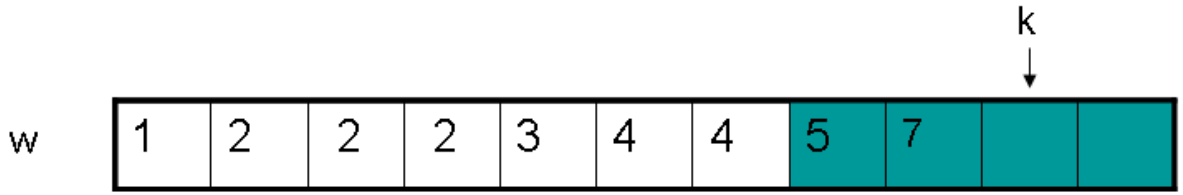
- Exemplo:



j == r → lista 2 terminou → transferir elementos restantes da lista 1 para o vetor w.

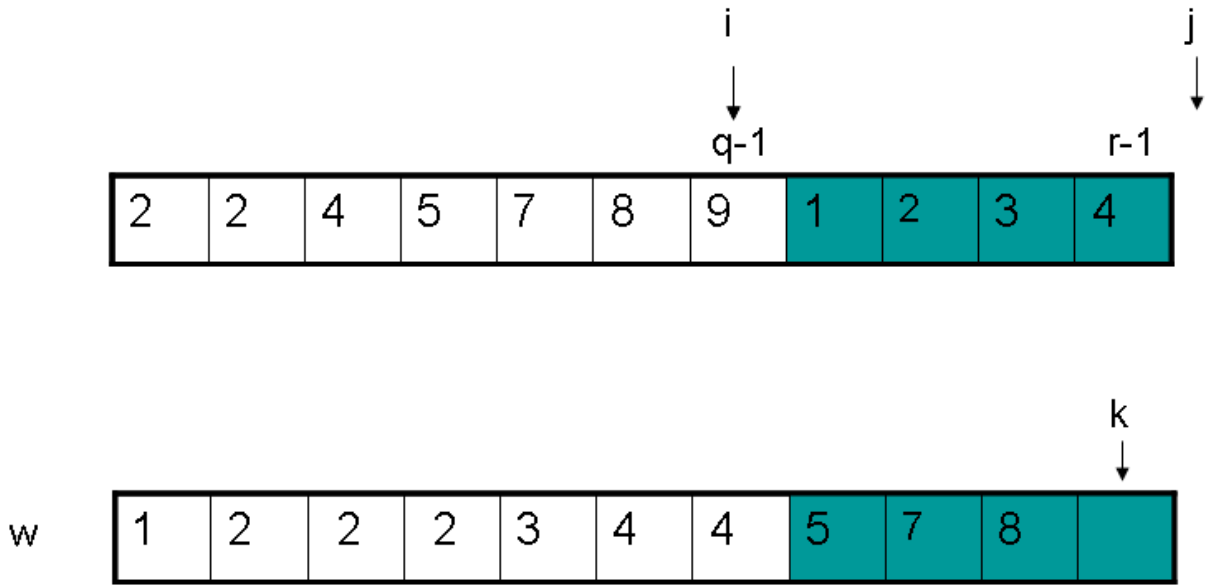
- Exemplo:





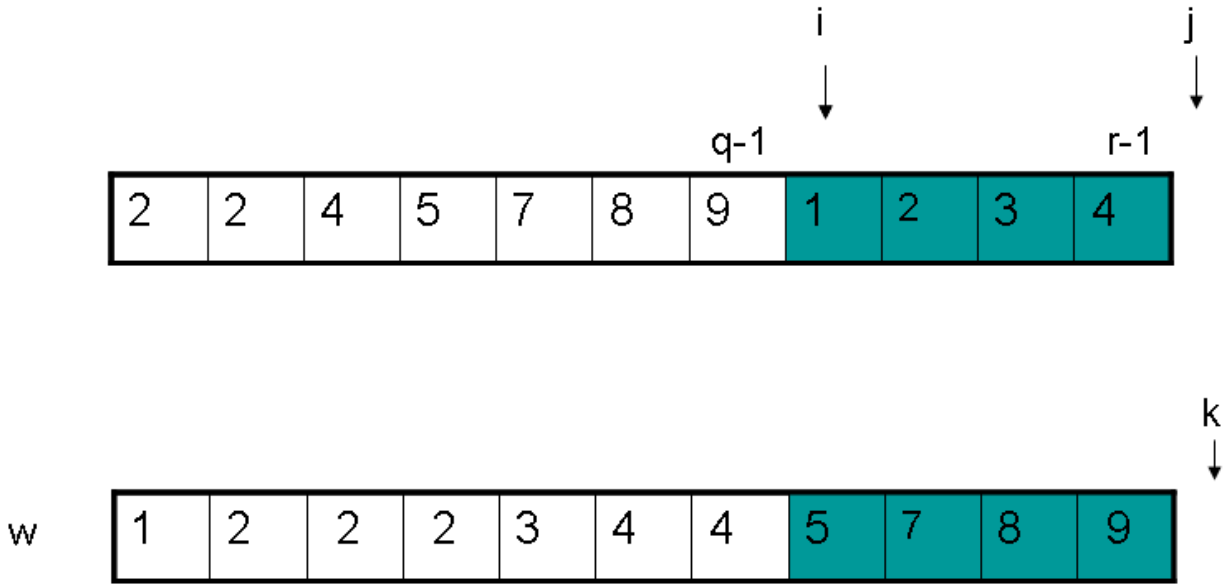
j == r → lista 2 terminou → transferir elementos restantes da lista 1 para o vetor w.

- Exemplo:



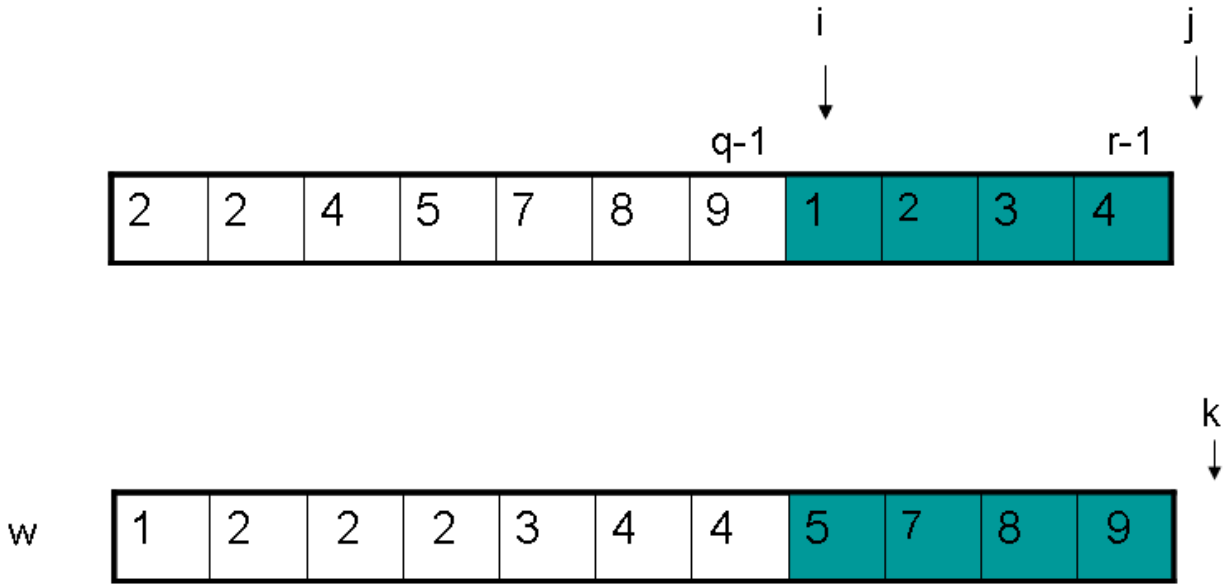
j == r → lista 2 terminou → transferir elementos restantes da lista 1 para o vetor w.

- Exemplo:



j == r → lista 2 terminou → transferir elementos restantes da lista 1 para o vetor w.

- Exemplo:



j == r → lista 2 terminou → transferir elementos restantes da lista 1 para o vetor w.

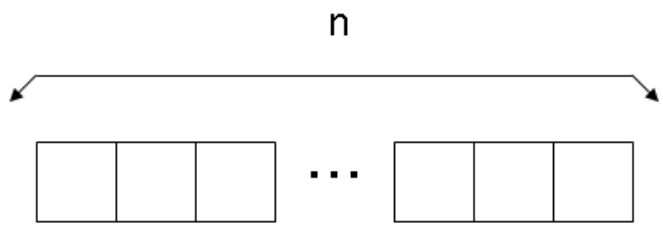
Complexidade: *O*(n)

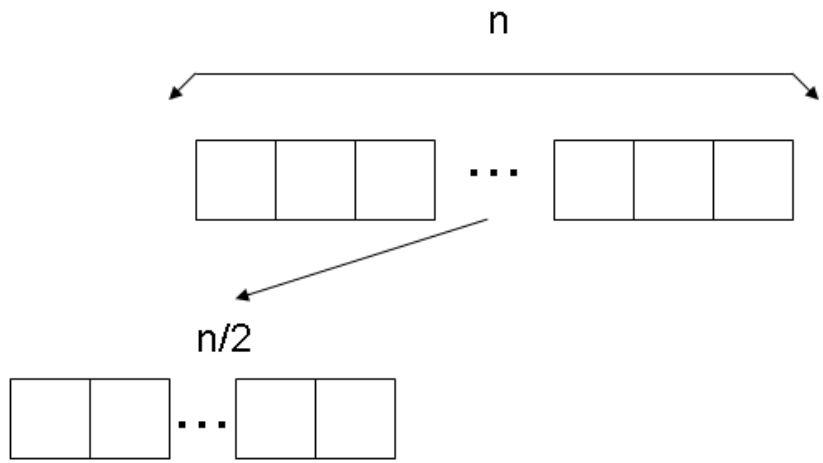
- Programa:

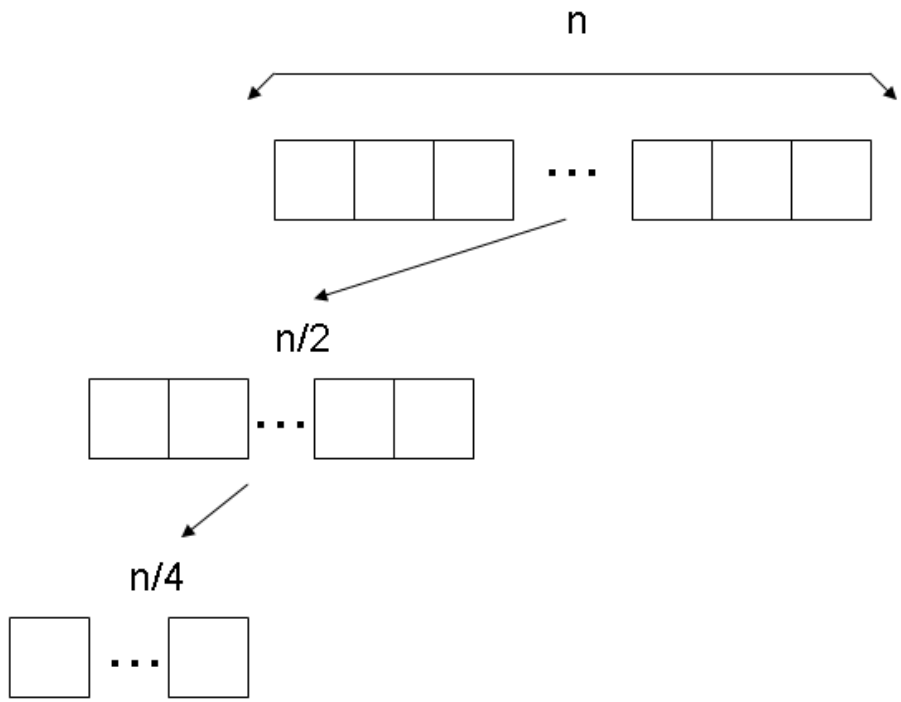
```
/* recebe vetores crescentes v[p..q-1] e v[q..r-1] e
rearranja v[p..r-1] em ordem crescente */
void Intercala(int p, int q, int r, int v[]) {
  int i, j, k, *w ;
  w = (int *)malloc((r-p)*sizeof(int)) ;
  i = p; j = q; k = 0;      /* i e j apontam para inicio...
  while (i<q && j<r) {          dos subvetores */
    if (v[i] <= v[j]) w[k++] = v[i++] ;
    else w[k++] = v[j++] ;
  }
  /* transferir elementos restantes de uma das listas */
  while (i<q) w[k++] = v[i++] ;
  while (j<r) w[k++] = v[j++] ;

  for (i=p; i<r; i++) v[i] = w[i-p] ;
  free(w) ;
}
```

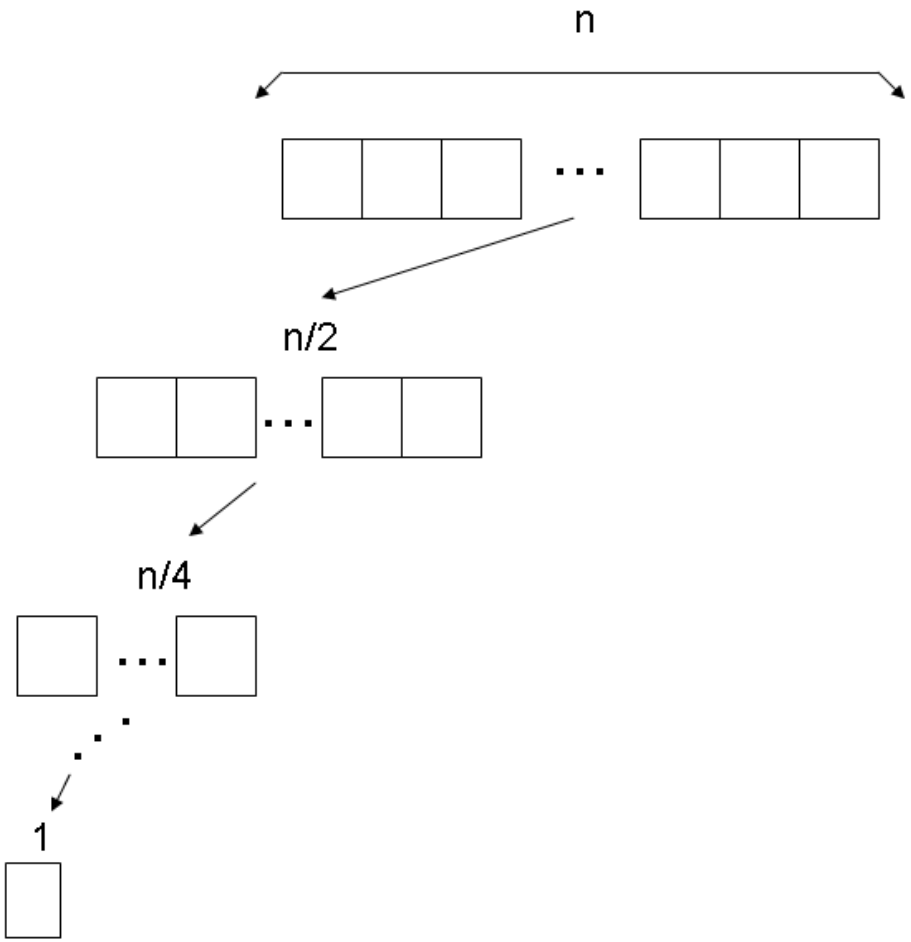
- O Mergesort:

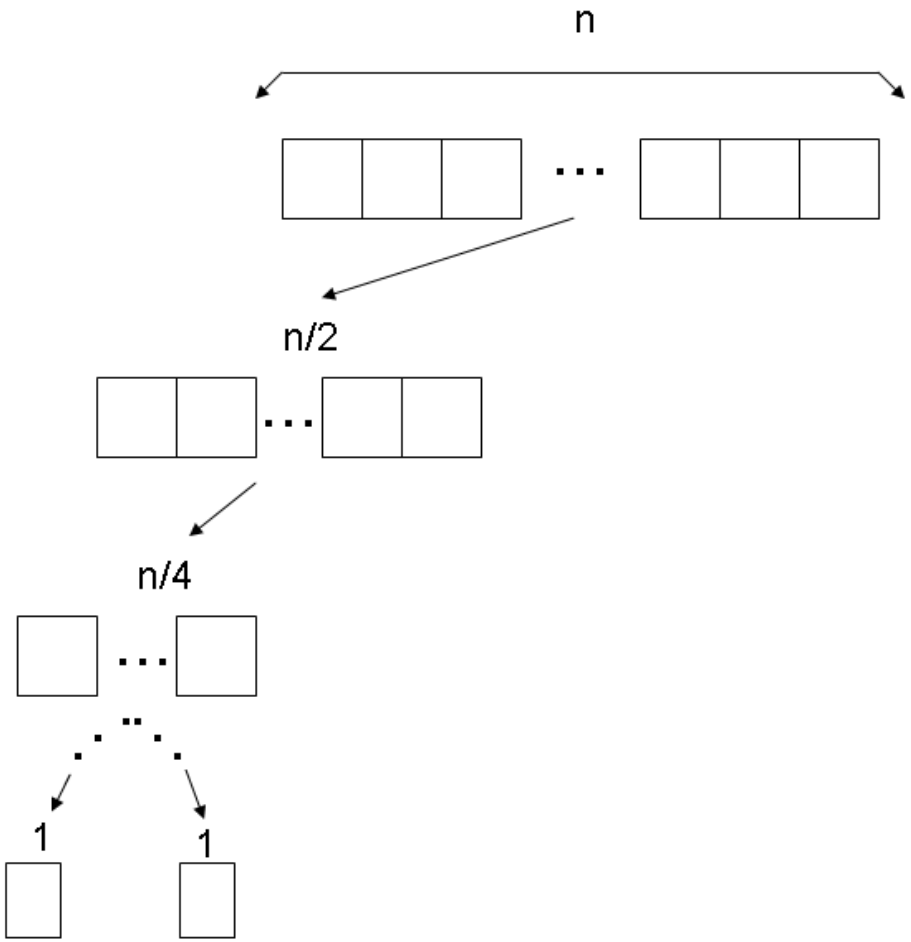
```
/* rearranja o vetor v[p..r] em ordem crescente; */
/* considera a funcao Intercala anterior */

void MergeSort(int p, int r, int v[]) {
  if (p < r) {
    int q = (p+r)/2 ;
    MergeSort(p, q, v) ;     /* ordena parte esquerda */
    MergeSort(q+1, r, v) ;     /* ordena parte direita */
    Intercala(p, q+1, r+1, v) ; /* intercala as duas partes *
  }
}
```

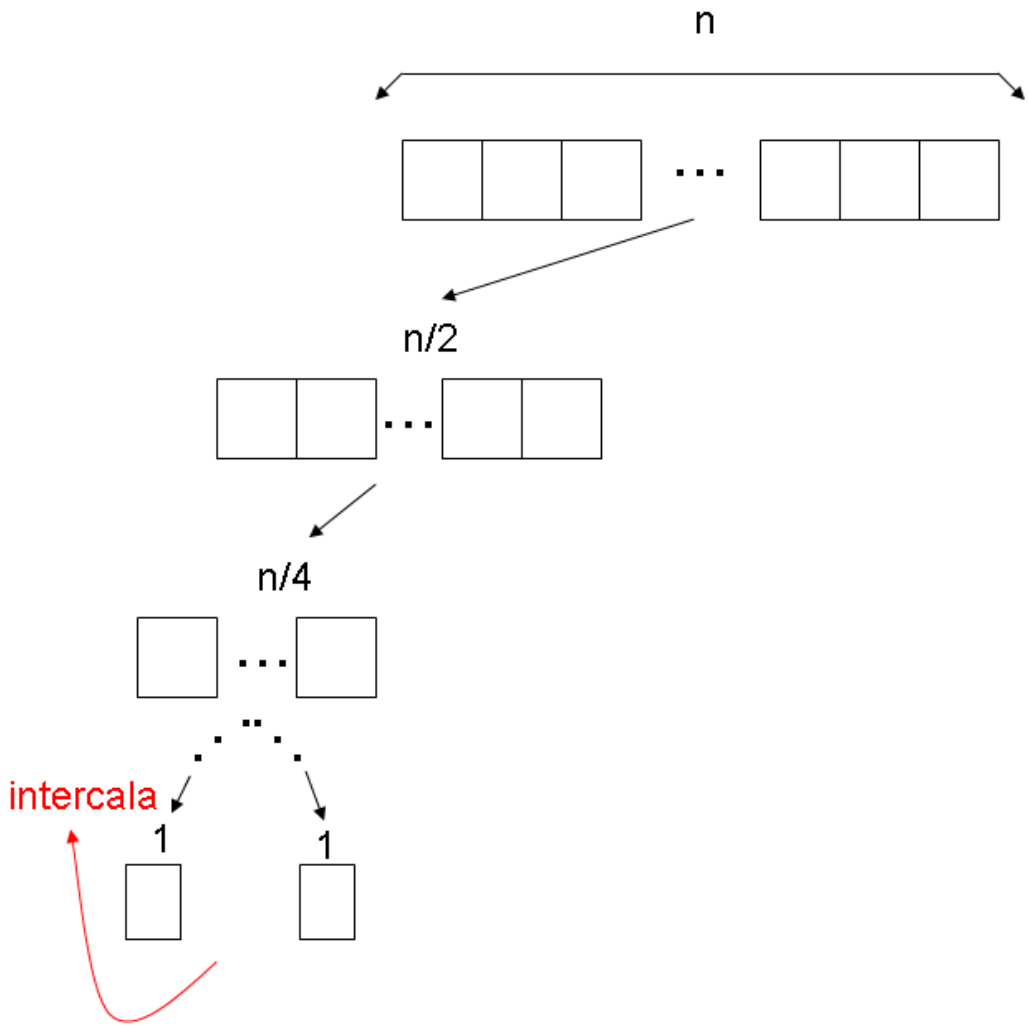

n

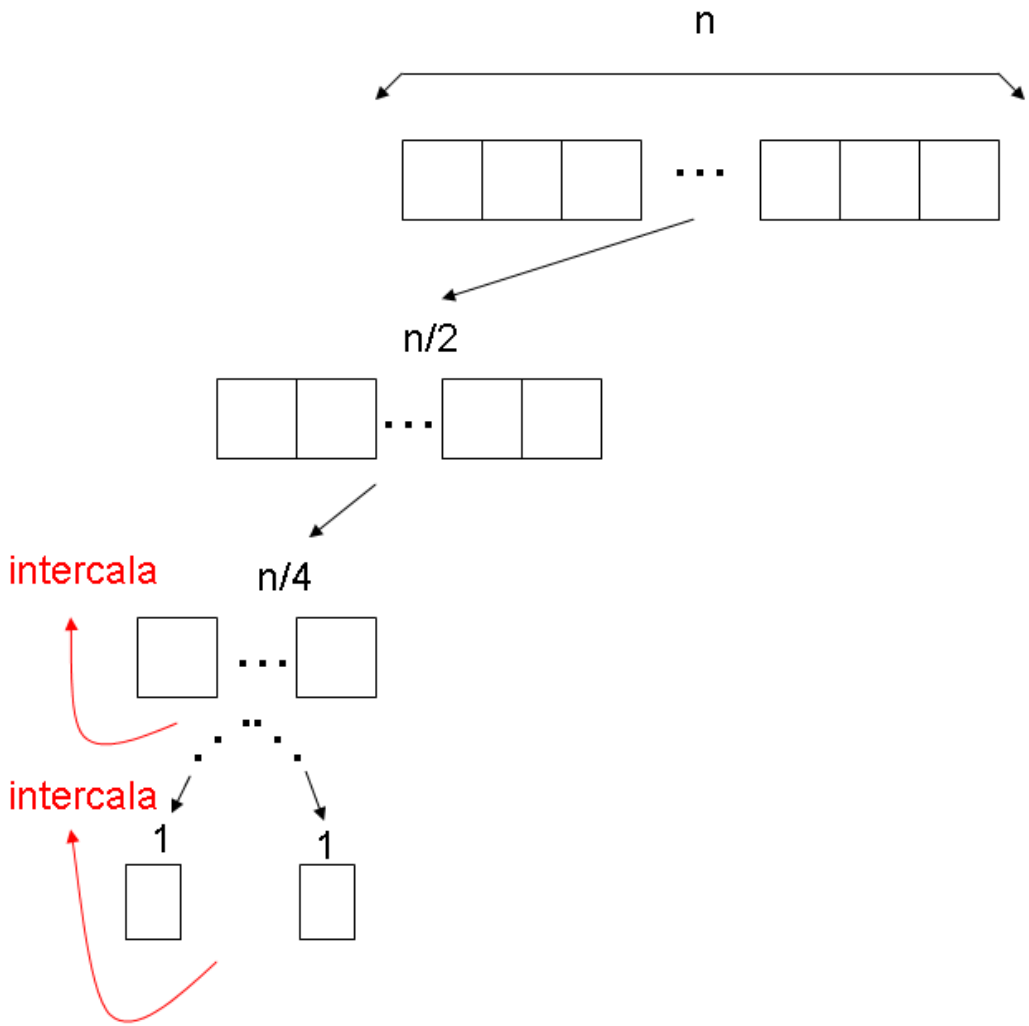

n
n/2

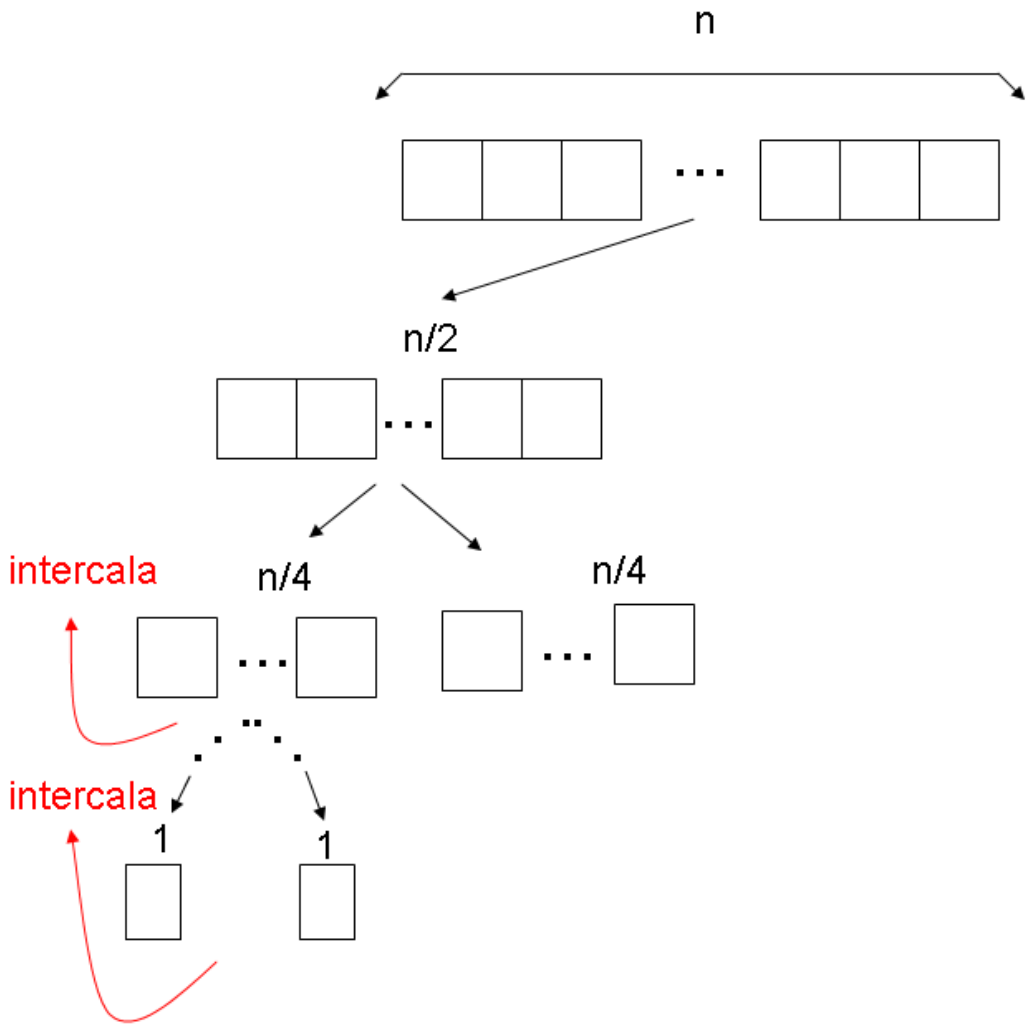

n
n/2
n/4

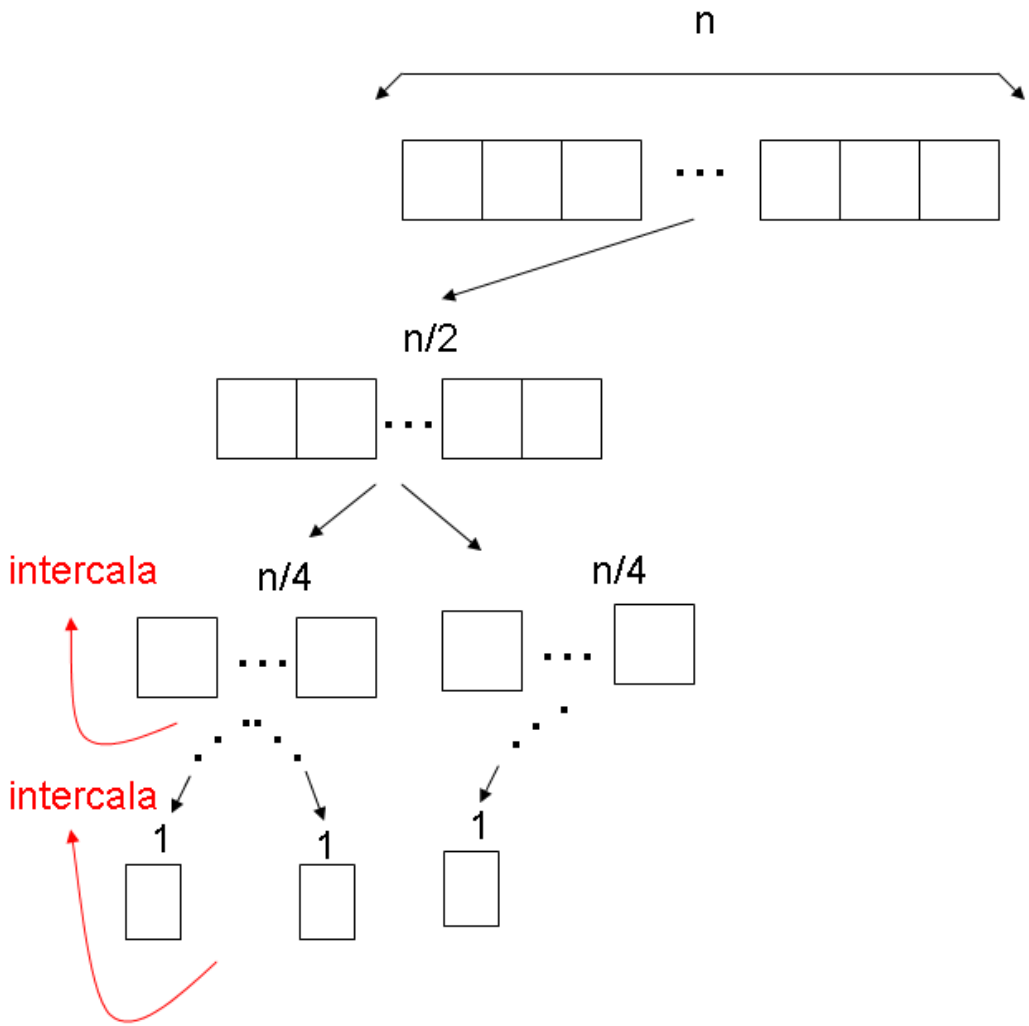

n
n/2
n/4
1

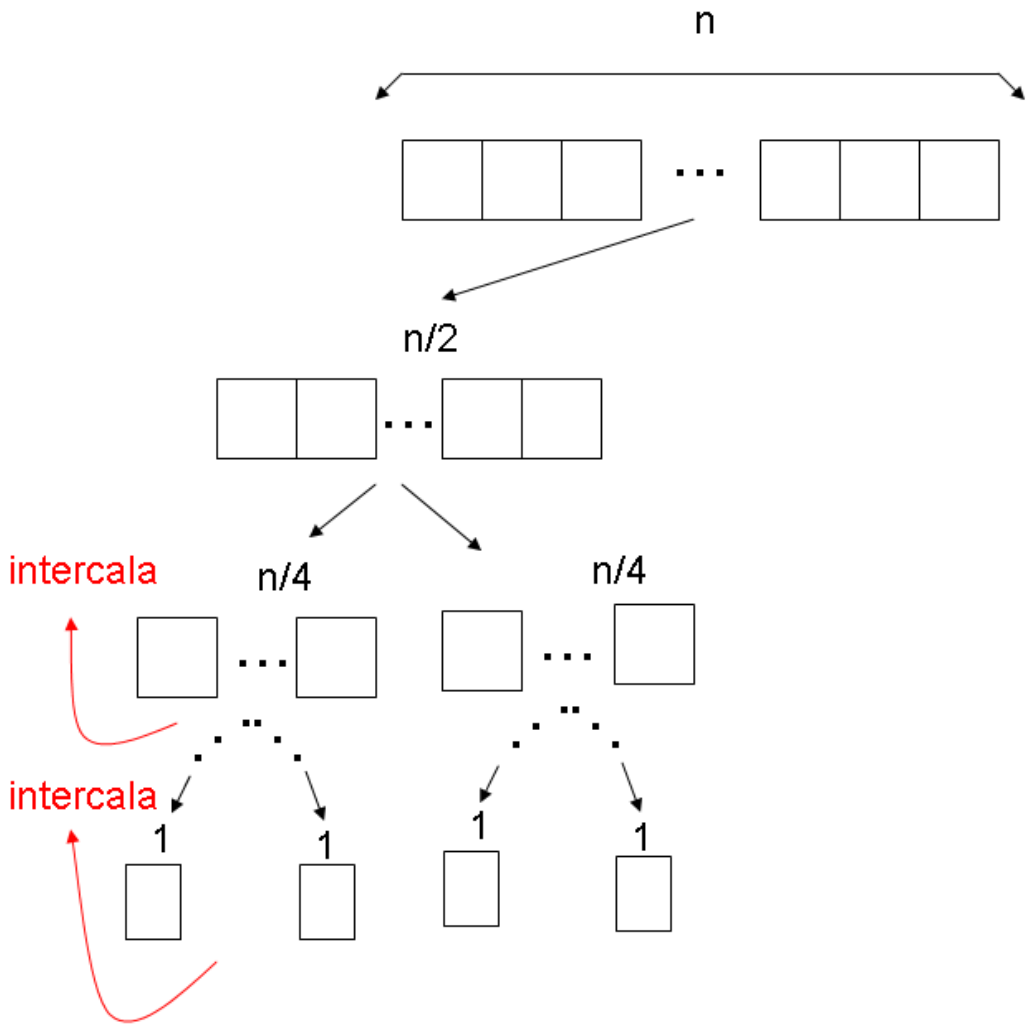

n
n/2
n/4
1
1

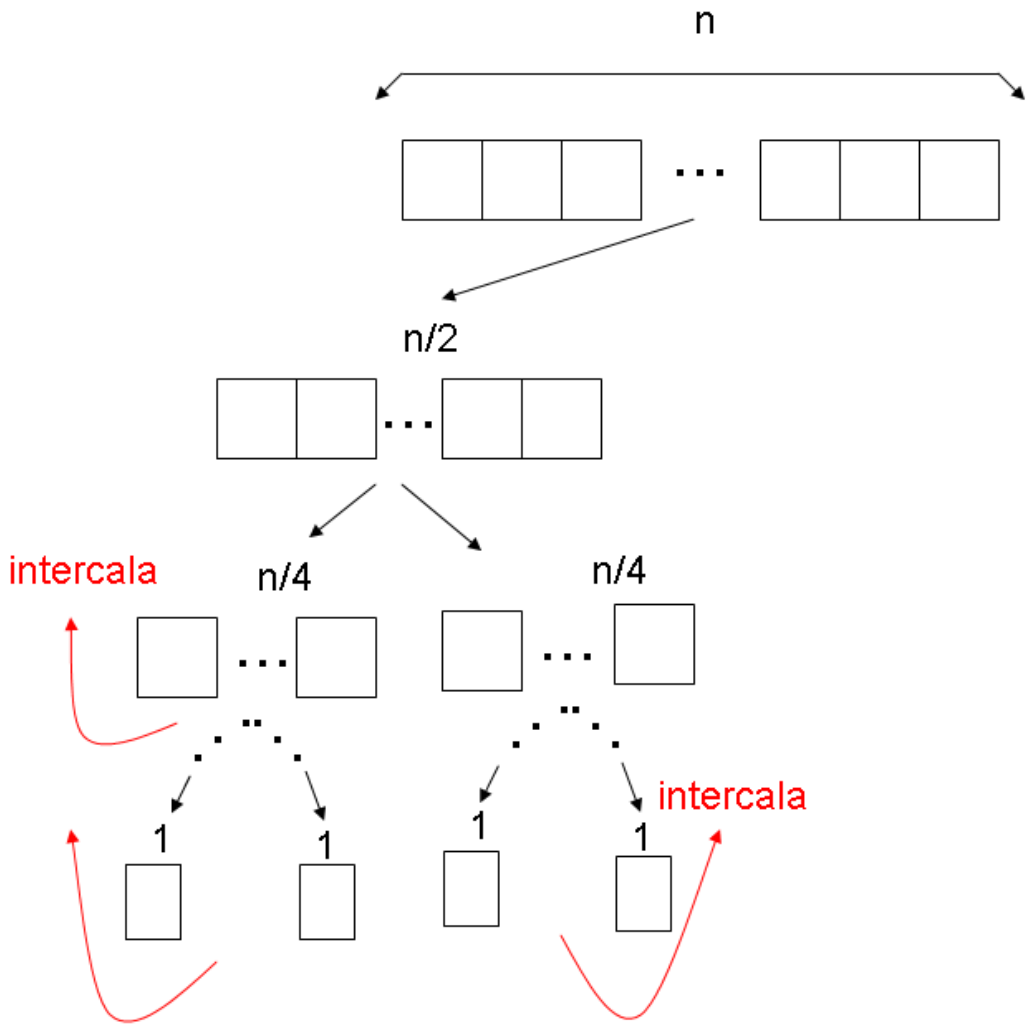

n
n/2
n/4
intercala
1
1

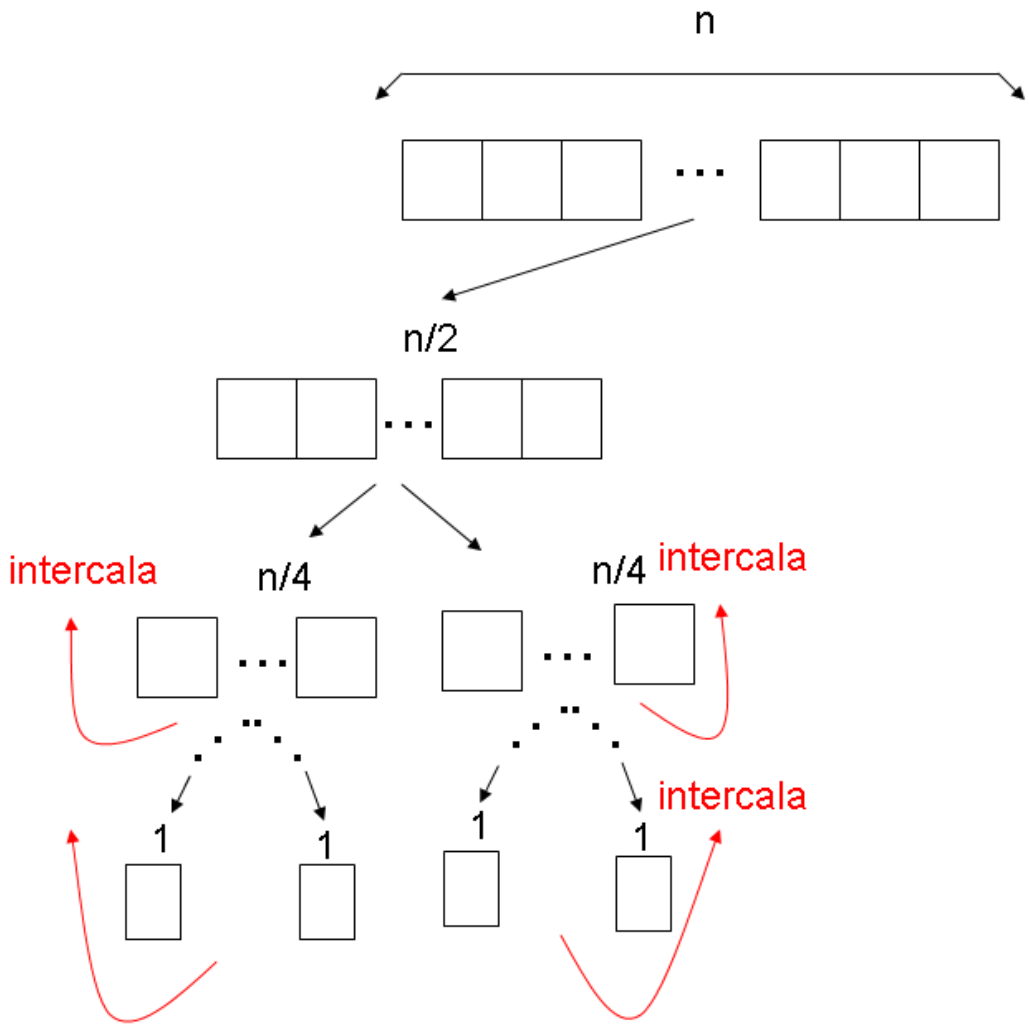

n
n/2
intercala
n/4
intercala
1
1

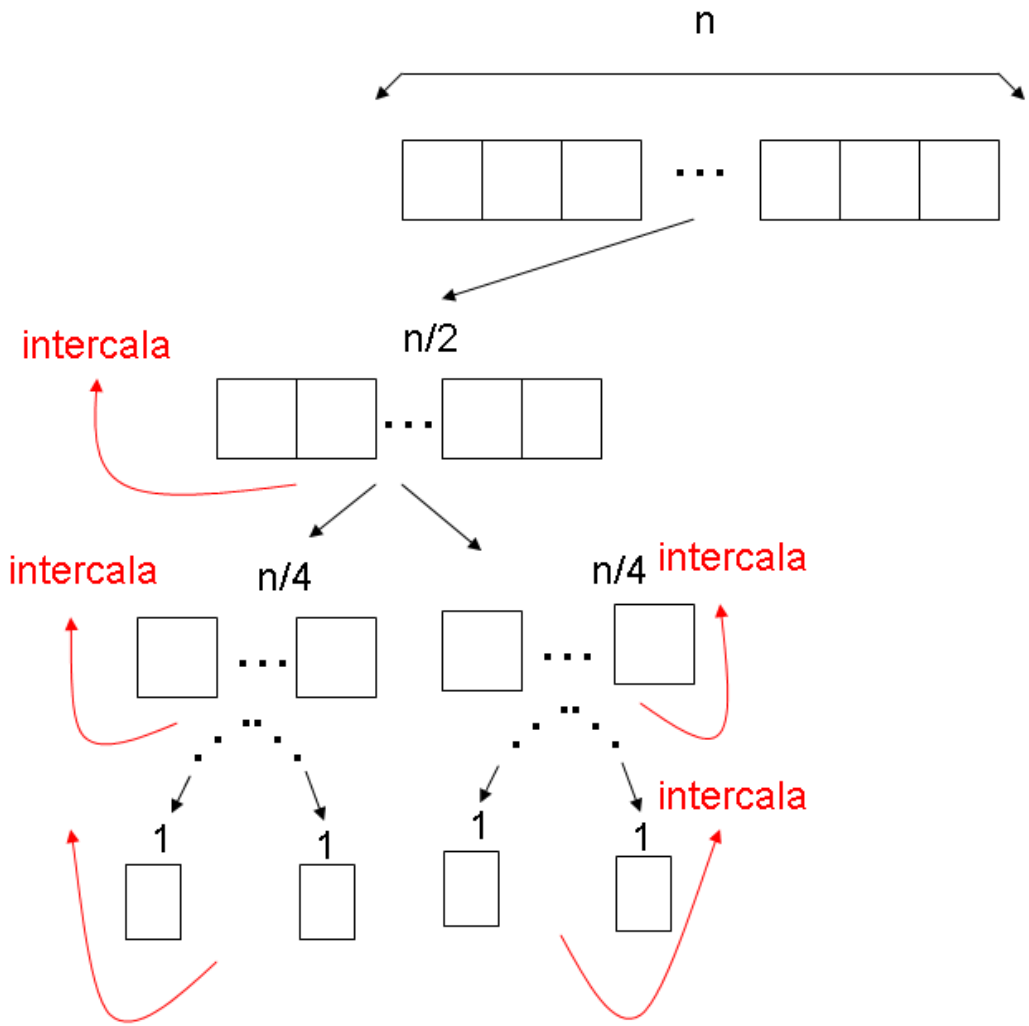

n
n/2
intercala
n/4
n/4
intercala
1
1

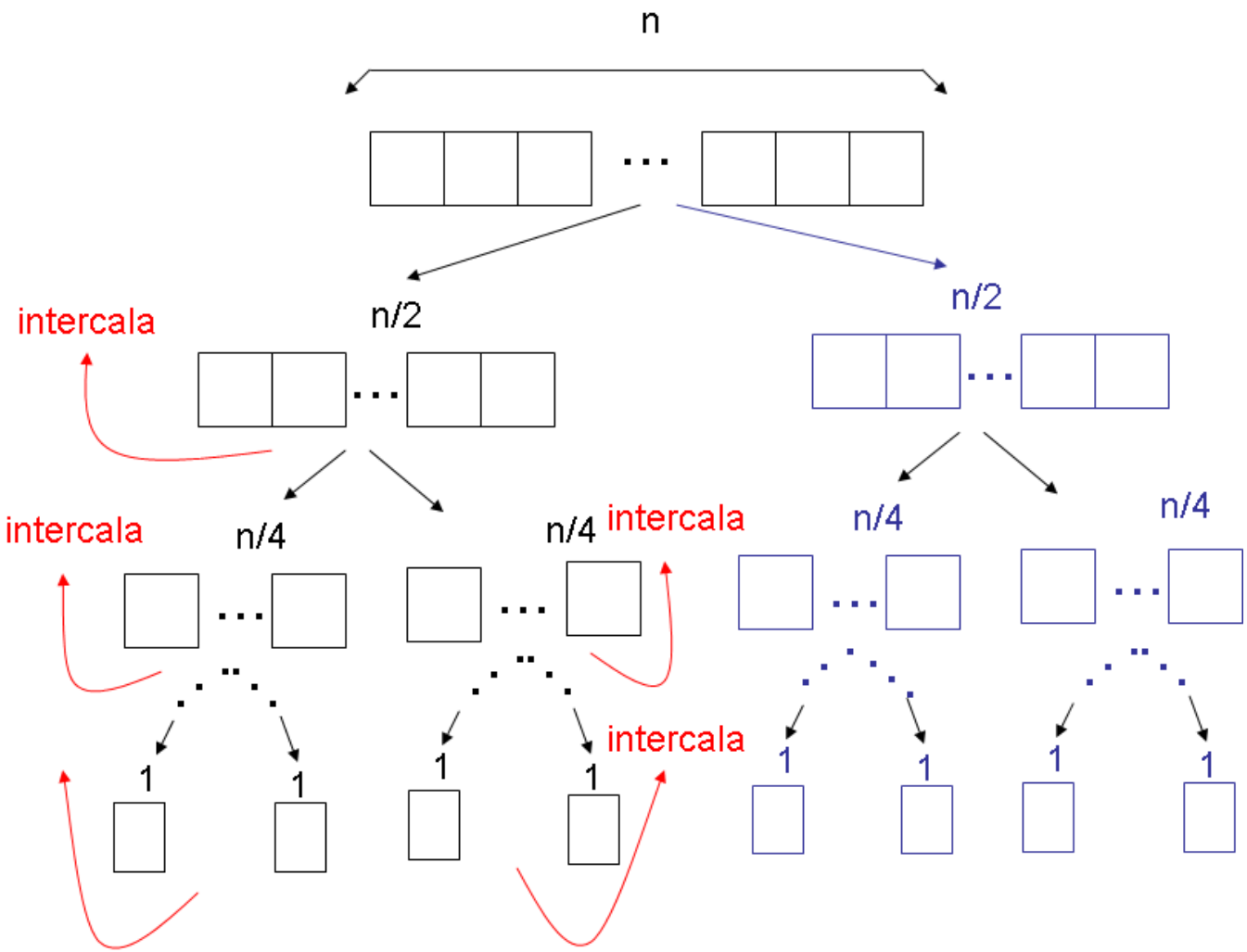

n
n/2
intercala
n/4
n/4
intercala
1
1
1


n
n/2
intercala
n/4
n/4
intercala
1
1
1
1


n
n/2
intercala
n/4
n/4
1
1
1
1
intercala


n
n/2
intercala
n/4
n/4
intercala
intercala
1
1
1
1


n
n/2
intercala
intercala
n/4
n/4
intercala
intercala
1
1
1
1



Complexidade: O(nlogn)

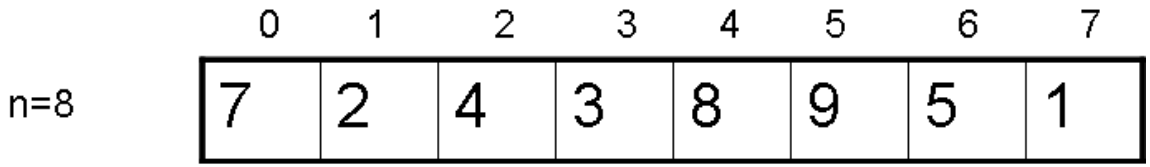
- Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

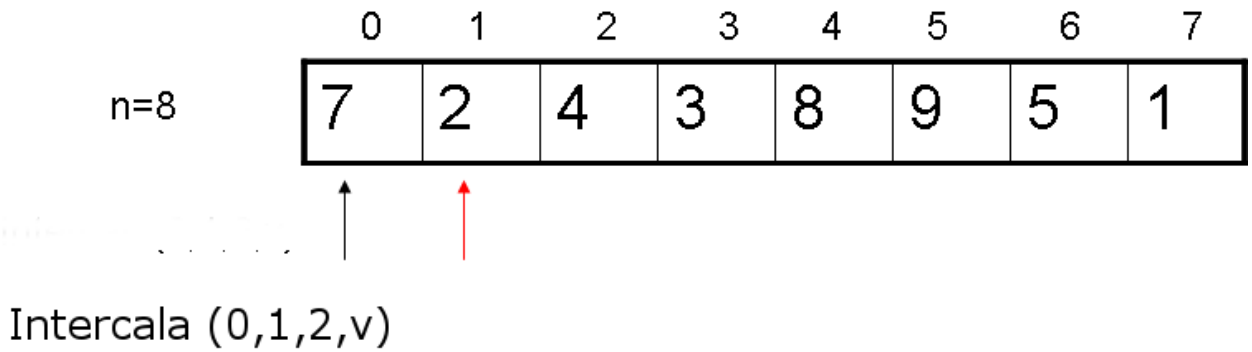
– Exemplo:



• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

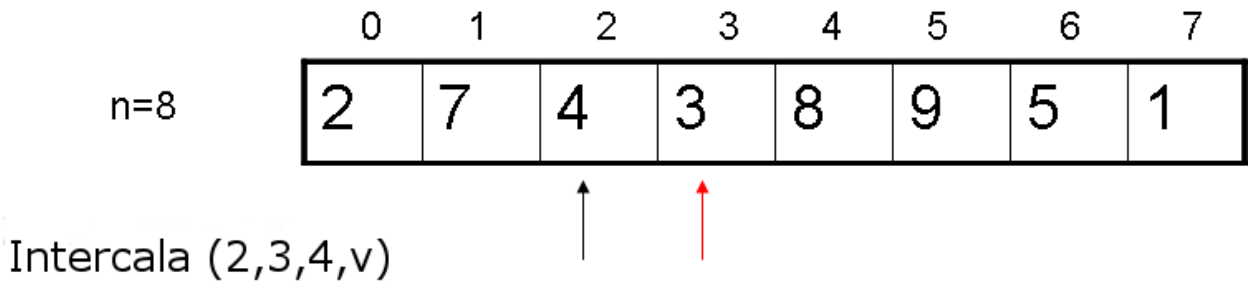
– Exemplo:



• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

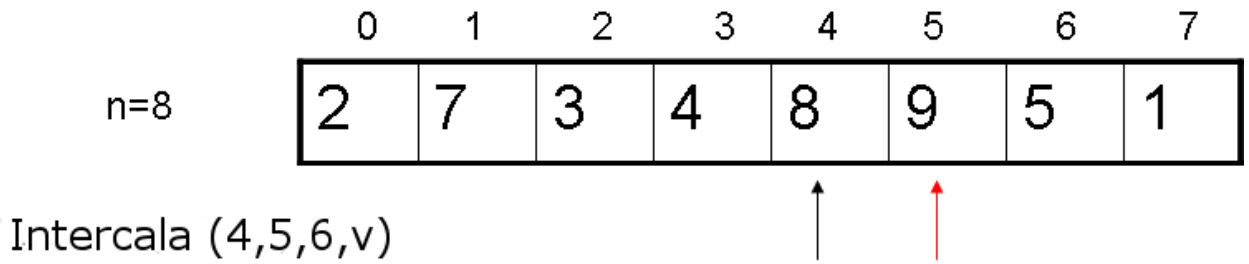
– Exemplo:



• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

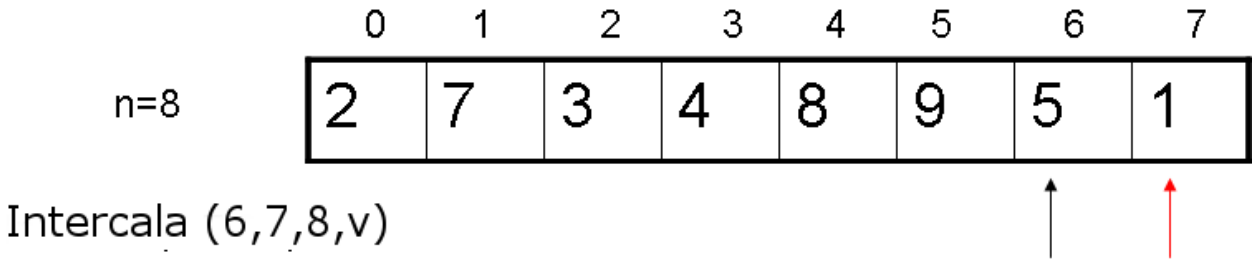
– Exemplo:



• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

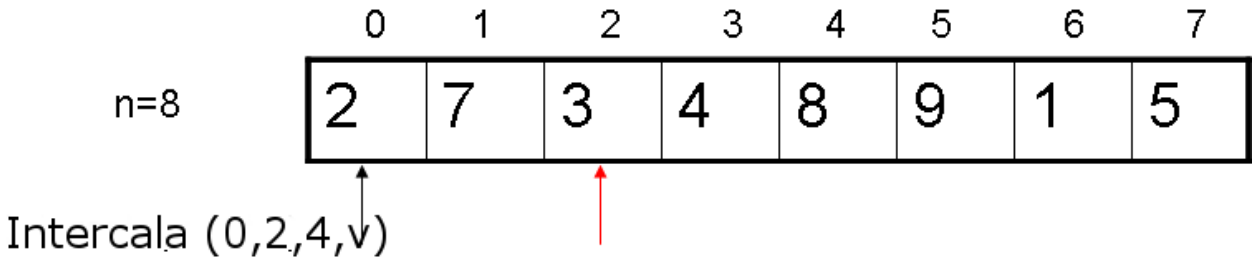
– Exemplo:



• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

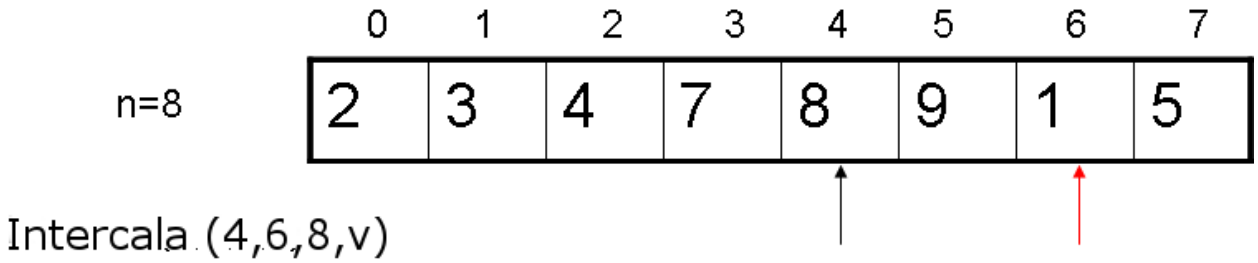
– Exemplo:



• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

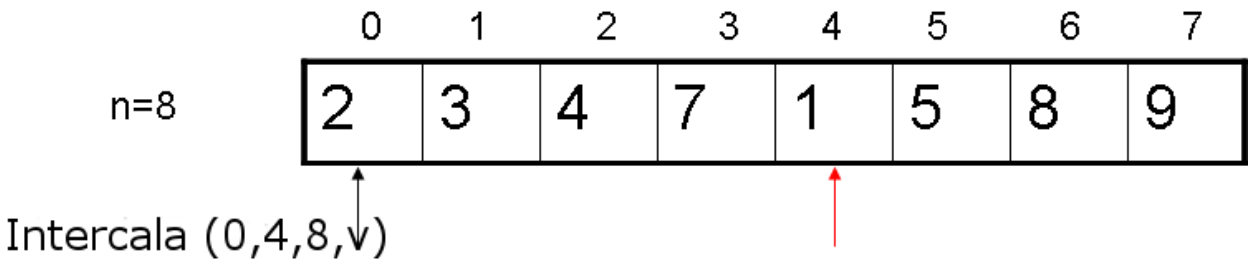
– Exemplo:



• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

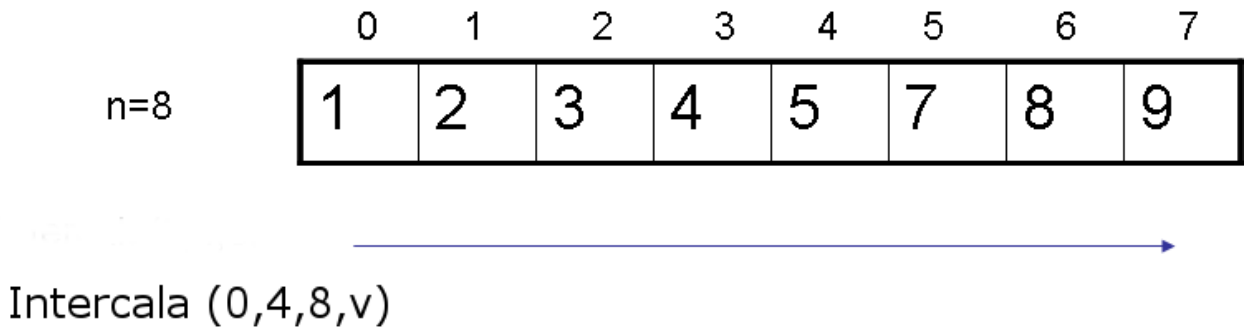
– Exemplo:



• Mergesort iterativo:

– Cada iteração intercala dois subconjuntos de b elementos: o primeiro subconjunto com o segundo, o terceiro com o quarto etc. O intervalo de intercalamento, b, assume valores 1, 2, 4, 8,...

– Exemplo:



- Programa iterativo:

```
/* Rearranja o vetor v[0..n-1] em ordem crescente */
void MergeSortI(int n, int v[]) {
  int p, r ;
  int b = 1 ;          /* intervalo do intercalamento */
  while (b < n) {
    p = 0 ;            /* posicao inicial do intercalamento */
    while (p + b < n) {
      r = p + 2*b ;
      if (r > n) r = n ;
      Intercala (p, p+b, r, v) ;
      p = p + 2*b ;
    }
    b = 2*b ;          /* dobra intervalo do intercalamento */
  }
}
```

– Exercício: Funciona para n ímpar?

- Programa iterativo:

```
/* Rearranja o vetor v[0..n-1] em ordem crescente */
void MergeSortI(int n, int v[]) {
  int p, r ;
  int b = 1 ;          /* intervalo do intercalamento */
  while (b < n) {
    p = 0 ;            /* posicao inicial do intercalamento */
    while (p + b < n) {
      r = p + 2*b ;
      if (r > n) r = n ;
      Intercala (p, p+b, r, v) ;
      p = p + 2*b ;
    }
    b = 2*b ;          /* dobra intervalo do intercalamento */
  }
}
```

– Exercício: Funciona para n ímpar?

- O algoritmo de ordenação Quicksort:

– Notação:

1. $v[h..j] \leq x \longrightarrow v[i] \leq x$ para todo i no conjunto de índices $h..j$
2. $v[h..j] \leq v[k..m] \longrightarrow v[i] \leq v[l]$ para todo i no conjunto $h..j$ e todo l no conjunto $k..m$.

– A base do Quicksort é representada pelo *algoritmo da separação* que, informalmente, consiste em rearranjar um vetor $v[p..r]$ de modo que, em relação a um certo *pivô*, os elementos menores fiquem todos do lado esquerdo e os maiores fiquem todos do lado direito.

Assim, para um pivô na posição j:
$v[p..j-1] \leq v[j] < v[j+1..r]$

- O algoritmo de ordenação Quicksort:

– Notação:

1. $v[h..j] \leq x \longrightarrow v[i] \leq x$ para todo i no conjunto de índices $h..j$
2. $v[h..j] \leq v[k..m] \longrightarrow v[i] \leq v[l]$ para todo i no conjunto $h..j$ e todo l no conjunto $k..m$.

– A base do Quicksort é representada pelo *algoritmo da separação* que, informalmente, consiste em rearranjar um vetor $v[p..r]$ de modo que, em relação a um certo *pivô*, os elementos menores fiquem todos do lado esquerdo e os maiores fiquem todos do lado direito.

Assim, para um pivô na posição j:
$v[p..j-1] \leq v[j] < v[j+1..r]$

- O algoritmo de ordenação Quicksort:

– Notação:

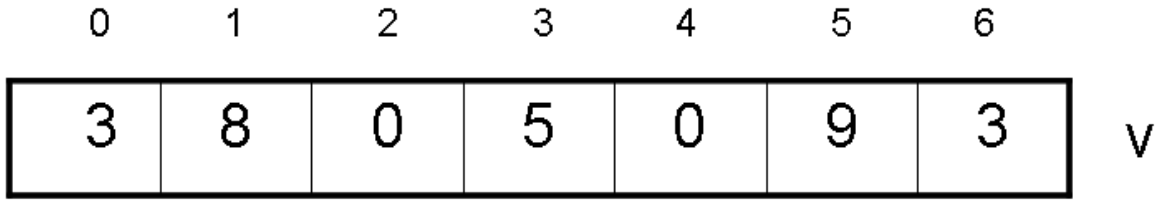
1. $v[h..j] \leq x \longrightarrow v[i] \leq x$ para todo i no conjunto de índices $h..j$
2. $v[h..j] \leq v[k..m] \longrightarrow v[i] \leq v[l]$ para todo i no conjunto $h..j$ e todo l no conjunto $k..m$.

– A base do Quicksort é representada pelo *algoritmo da separação* que, informalmente, consiste em rearranjar um vetor $v[p..r]$ de modo que, em relação a um certo *pivô*, os elementos menores fiquem todos do lado esquerdo e os maiores fiquem todos do lado direito.

Assim, para um pivô na posição j:
$v[p..j-1] \leq v[j] < v[j+1..r]$

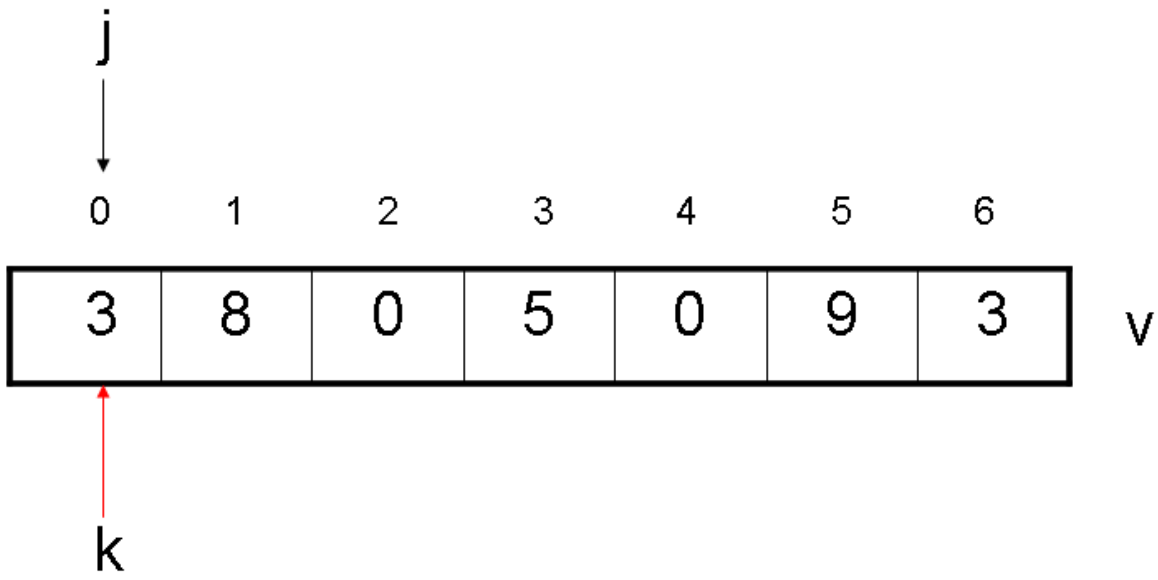
- Exemplo: algoritmo da separação

Para k $<$ 6:



- Exemplo: algoritmo da separação

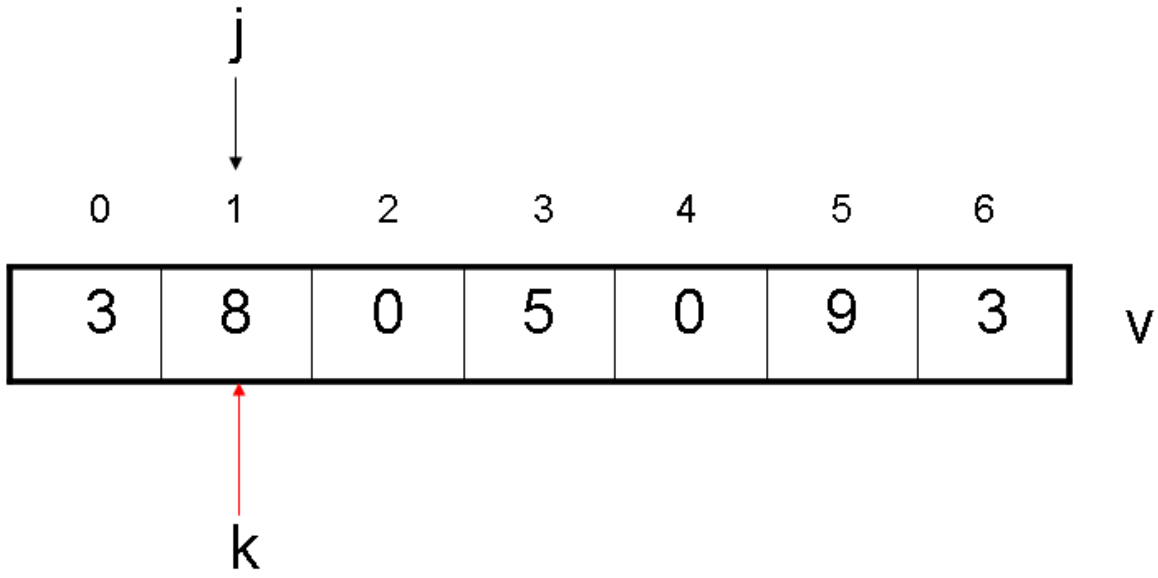
Para k $<$ 6:



-- troca v[j] com v[k] se v[k] <= v[6]
-- neste caso, avança k e j
-- senão, avança k

- Exemplo: algoritmo da separação

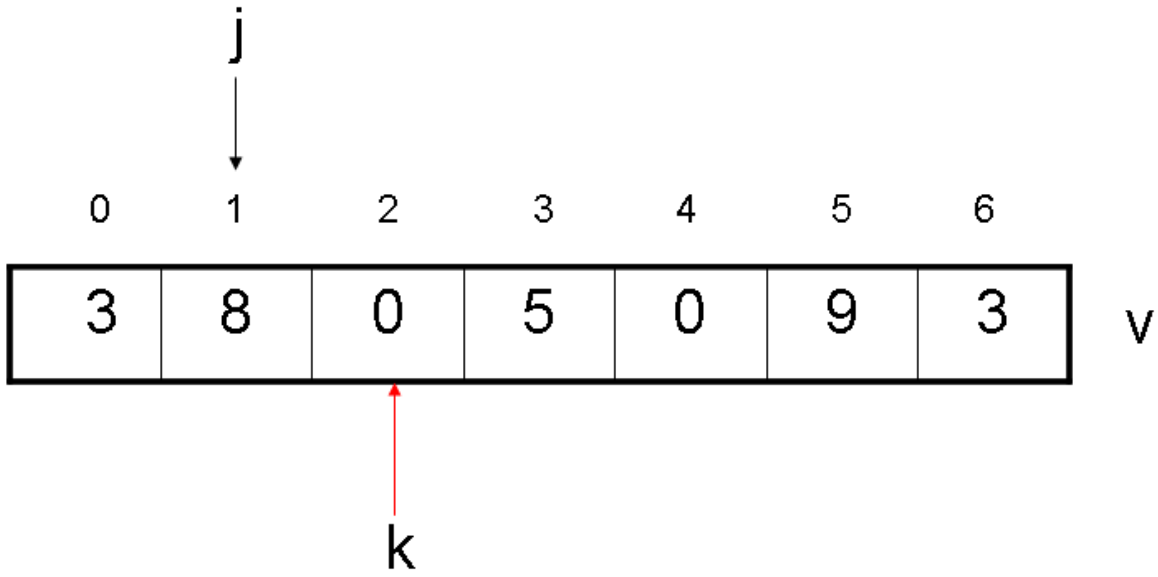
Para k $<$ 6:



-- troca v[j] com v[k] se v[k] <= v[6]
-- neste caso, avança k e j
-- senão, avança k

- Exemplo: algoritmo da separação

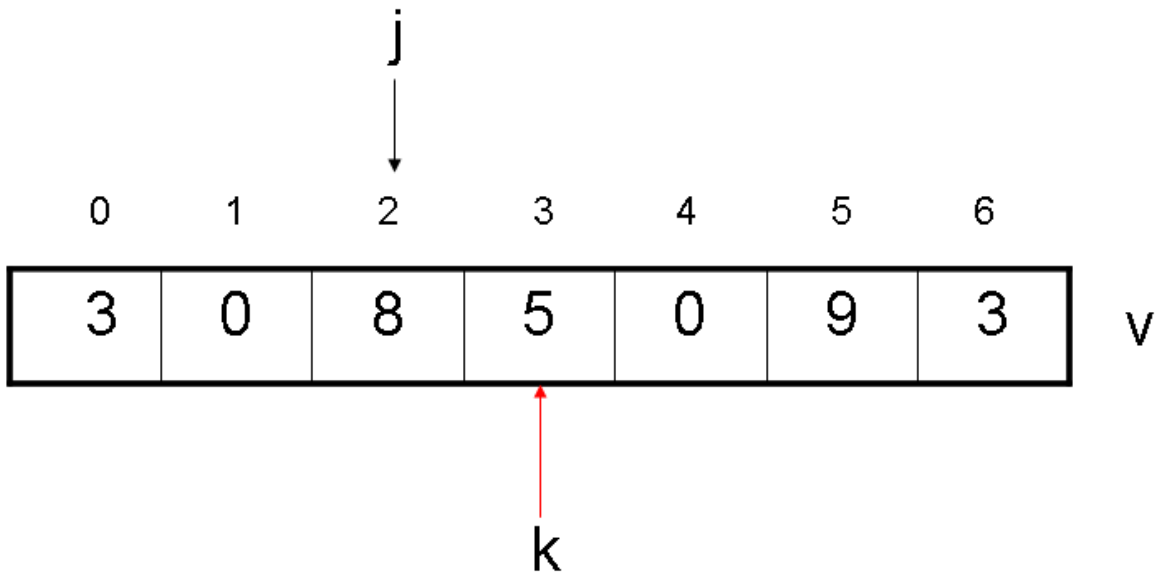
Para k $<$ 6:



-- troca v[j] com v[k] se v[k] <= v[6]
-- neste caso, avança k e j
-- senão, avança k

- Exemplo: algoritmo da separação

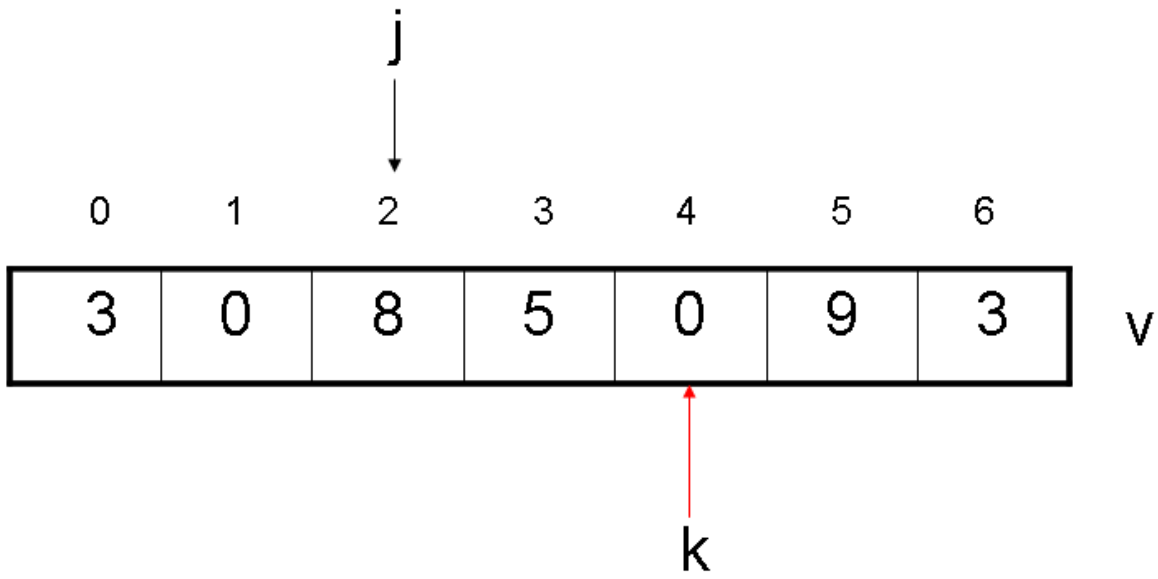
Para k $<$ 6:



-- troca v[j] com v[k] se v[k] <= v[6]
-- neste caso, avança k e j
-- senão, avança k

- Exemplo: algoritmo da separação

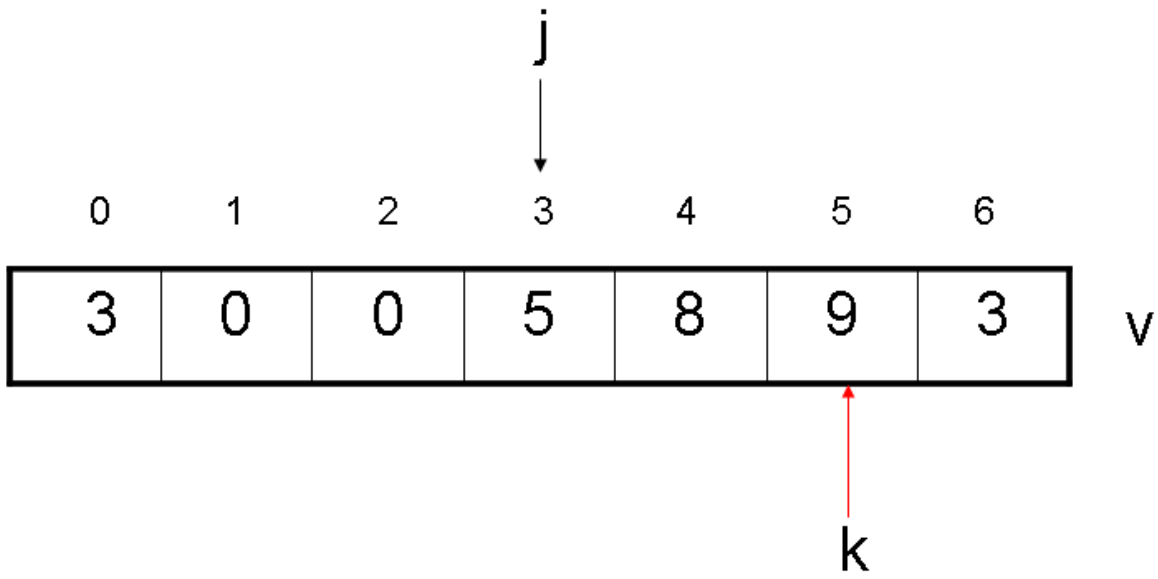
Para k $<$ 6:



-- troca v[j] com v[k] se v[k] <= v[6]
-- neste caso, avança k e j
-- senão, avança k

- Exemplo: algoritmo da separação

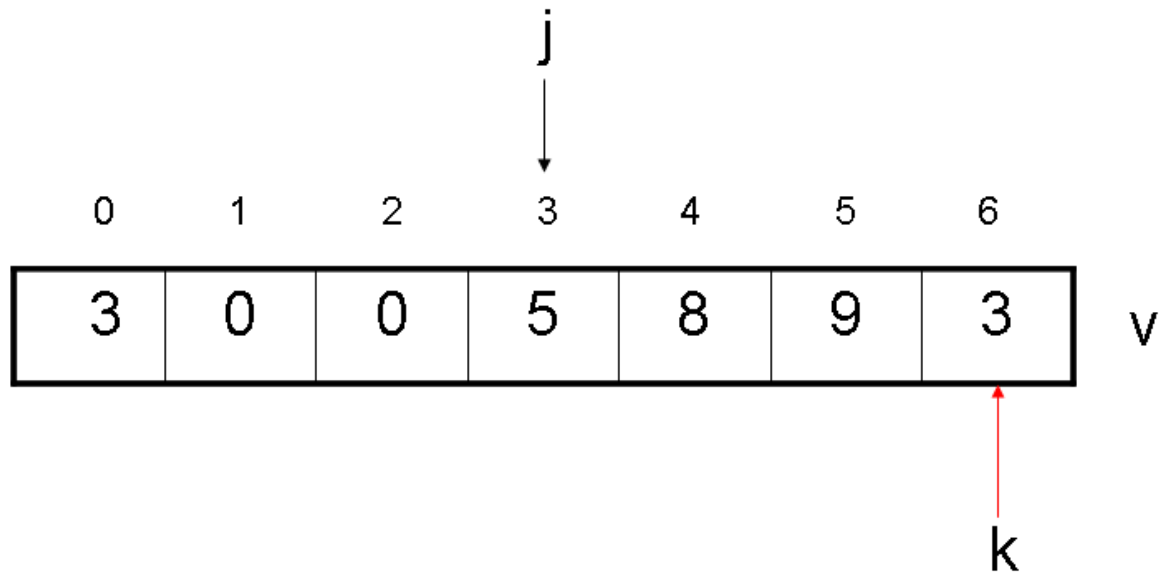
Para k $<$ 6:



-- troca v[j] com v[k] se v[k] <= v[6]
-- neste caso, avança k e j
-- senão, avança k

- Exemplo: algoritmo da separação

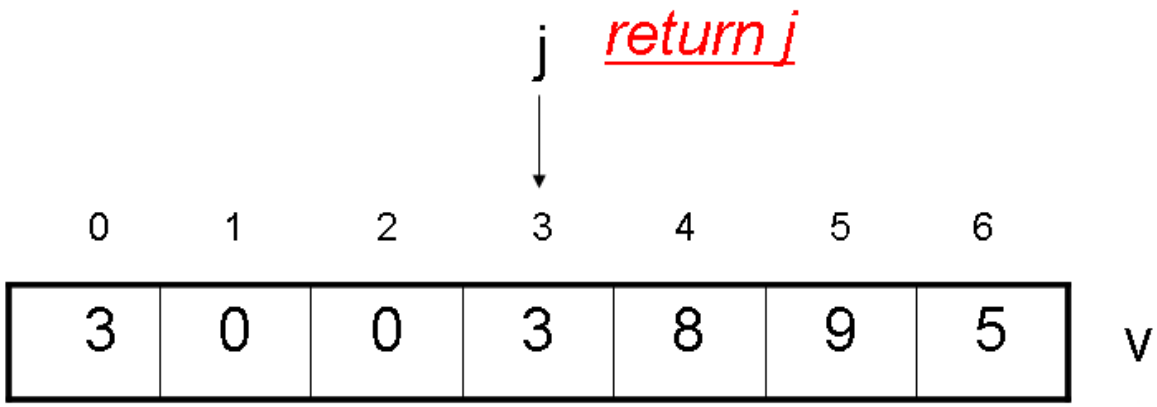
Para k == 6:



-- troca v[j] com v[k]

- Exemplo: algoritmo da separação

Para k == 6:



- Programa:

```
/* Recebe um vetor v[p..r] com p<=r. Rearranja os
elementos do vetor e devolve o pivo em algum j tal
que v[p..j-1] <= v[j] < v[j+1..r]. */

int Separa(int p, int r, int v[]) {
  int c, j, k, t ;
  c = v[r]; j = p ;
  for (k = p; k < r; k++)
    if (v[k] <= c) {
      t = v[j]; v[j] = v[k]; v[k]=t ;   /* troca */
      j++ ;
    }
  v[r] = v[j] ; v[j] = c ;
  return j ;       /* retorna posicao do pivo */
}
```

- Algoritmo do Quicksort:

```
/* rearranja o vetor v[p..r] em ordem crescente */

void Quicksort(int p, int r, int v[]) {
  int j ;
  if(p < r) {
    j = Separa (p, r, v);  /* v[j] esta na posicao correta */
    Quicksort (p, j-1, v); /* ordena lado esquerdo */
    Quicksort (j+1, r, v); /* ordena lado direito */
  }
}
```

– Qual a complexidade caso o vetor esteja ordenado?

- Algoritmo do Quicksort:

```
/* rearranja o vetor v[p..r] em ordem crescente */

void Quicksort(int p, int r, int v[]) {
  int j ;
  if(p < r) {
    j = Separa (p, r, v);  /* v[j] esta na posicao correta */
    Quicksort (p, j-1, v); /* ordena lado esquerdo */
    Quicksort (j+1, r, v); /* ordena lado direito */
  }
}
```

– Qual a complexidade caso o vetor esteja ordenado?

Referências:

– Paulo Feofiloff. Algoritmos em linguagem C, Elsevier 2009.

– Nívio Ziviani. Projeto de Algoritmos com Implementações em Pascal e C, Thomson Learning, 2004.