Universidade Federal do ABC

CMCC
Centro de Matemática, Computação e Cognição

Algoritmos e Estruturas de Dados I

Ordenação (parte I)

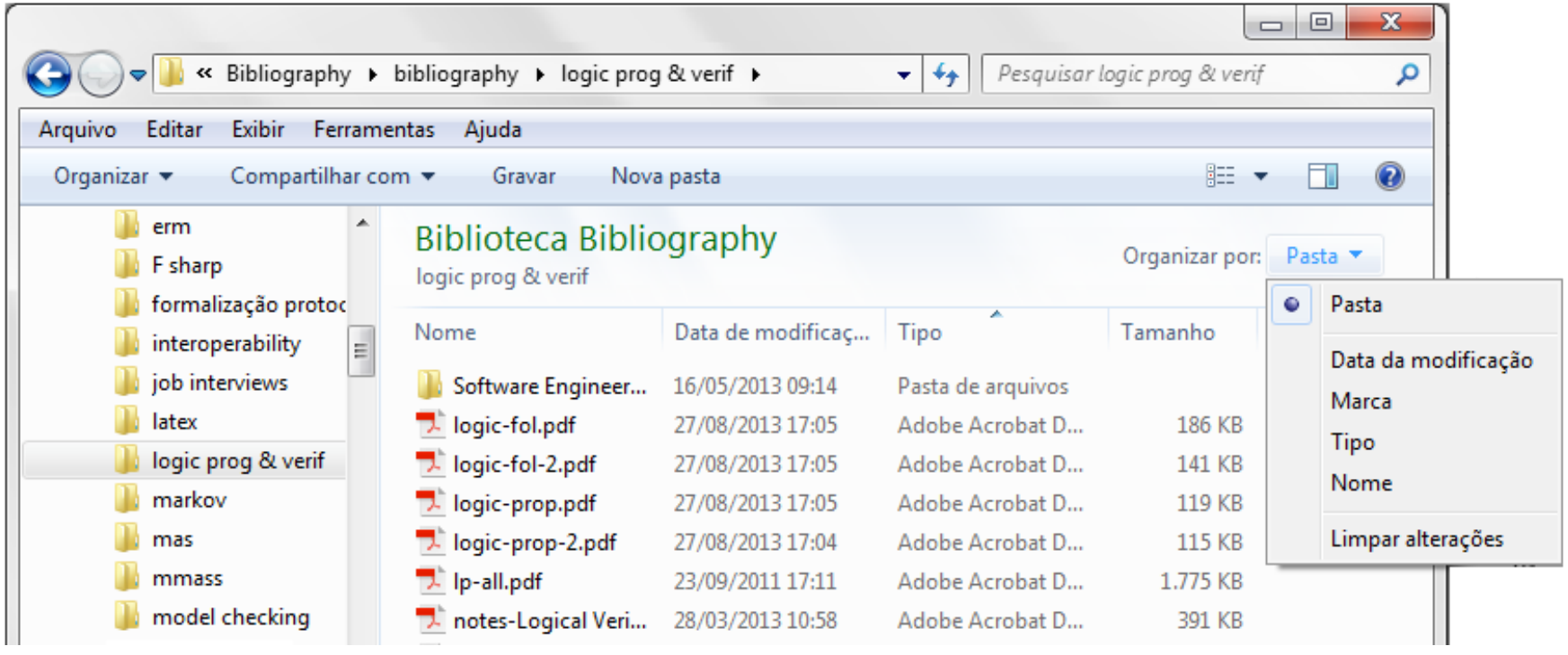
Mirtha Lina Fernández Venero

http://professor.ufabc.edu.br/~mirtha.lina/aedi.html

20 de abril de 2019

O problema de ordenar (Sorting)

dada uma sequência de N registros de dados $d_1, d_2, \ldots, d_N$ obter uma permutação $d_{p(1)} \preceq d_{p(2)} \preceq \ldots \preceq d_{p(N)}$ onde $\preceq$ é uma **ordem** total definida para a $chave$ do registro



Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

O problema de ordenar (Sorting)

dada uma sequência de N registros de dados $d_1, d_2, \ldots, d_N$ obter uma permutação $d_{p(1)} \preceq d_{p(2)} \preceq \ldots \preceq d_{p(N)}$ onde $\preceq$ é uma **ordem** total definida para a $chave$ do registro

O problema de embaralhar (Shuffling):

Operação contrária a ordenar que consiste em obter uma permutação aleatória duma sequência de dados

O problema de ordenar (Sorting)

dada uma sequência de N registros de dados $d_1, d_2, \ldots, d_N$ obter uma permutação $d_{p(1)} \preceq d_{p(2)} \preceq \ldots \preceq d_{p(N)}$ onde $\preceq$ é uma **ordem** total definida para a $chave$ do registro

O problema de embaralhar (Shuffling):

Operação contrária a ordenar que consiste em obter uma permutação aleatória duma sequência de dados

```
void shufflingArr(int *v, int n) {
    for(int i=n-1; i>0; i--){
      int r = rand() % (i+1);  // random int between 0...i
      int aux = v[i]; v[i] = v[r]; v[r] = aux; // swap
    }
}
```

Vantagens de ordenar

- Melhora a visualização dos dados
- Após ordenar a busca (elemento, mínimo, máximo) é mais eficiente
- Permite remover elementos repetidos, checar se não há repetições, calcular a distribuição de frequências com facilidade
- Passo inicial para resolver de forma mais simples/eficiente um problema e.g. calcular a mediana e a moda, obter uma árvore geradora mínima, construir uma árvore B+
- Usada em muitas áreas da Computação, e.g. banco de dados, sistemas operacionais, recuperação de informação, etc

Tipos de Ordenação

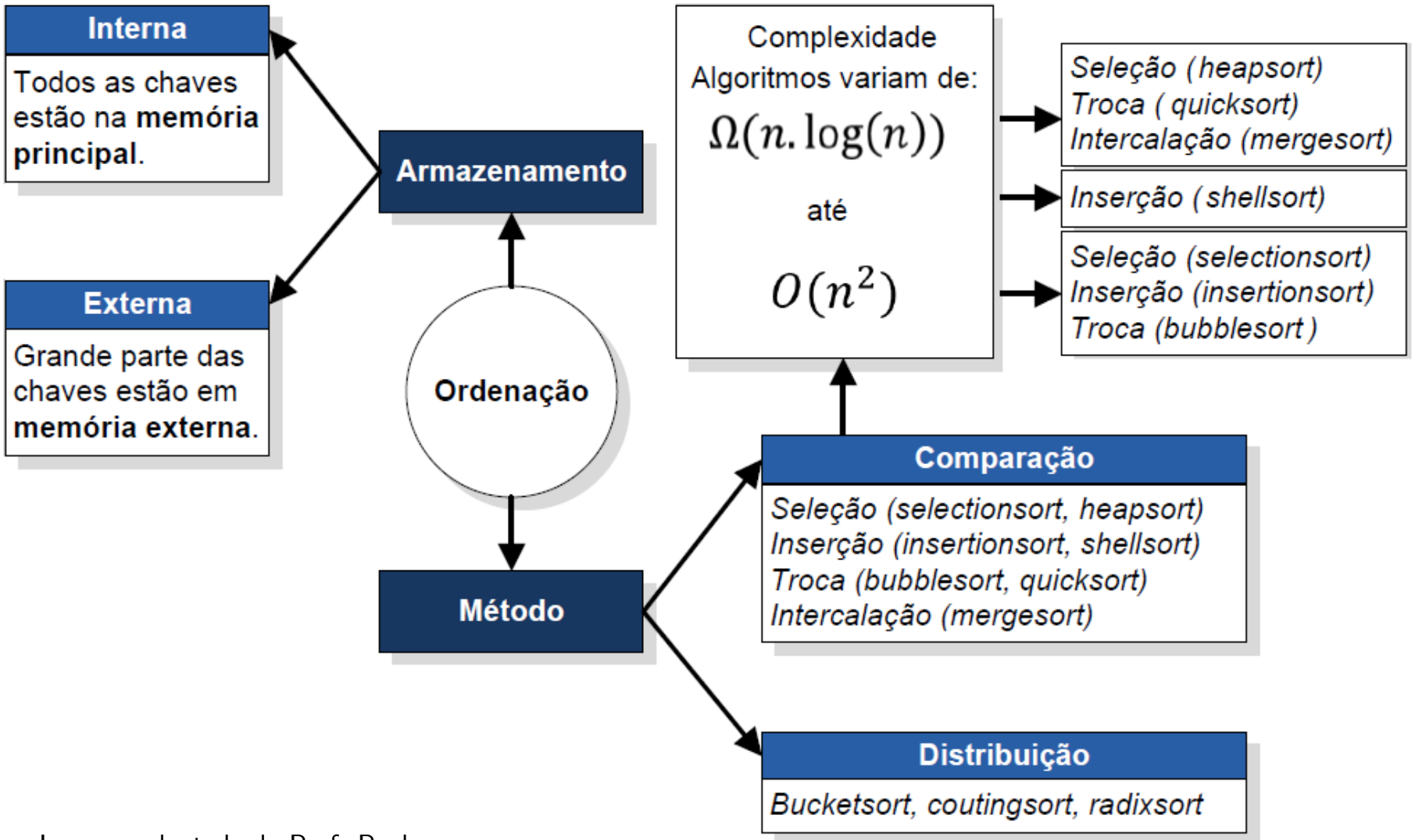


Imagem adaptada do Prof. Paulo.

Características dos Algoritmos de Ordenação

- **In-place**: se a quantidade de memória auxiliar utilizada pelo algoritmo é $O(log(n))$
  Exemplo: Mergesort **não** é um algoritmo *in-place*
- **Estável**: se a ordem relativa dos itens com chaves iguais mantém-se inalterada

Chaves		InsertionSort		SelectionSort
3		1		1
5		1		1
2		2		2
1		3		3
5		5		5
1		5		5

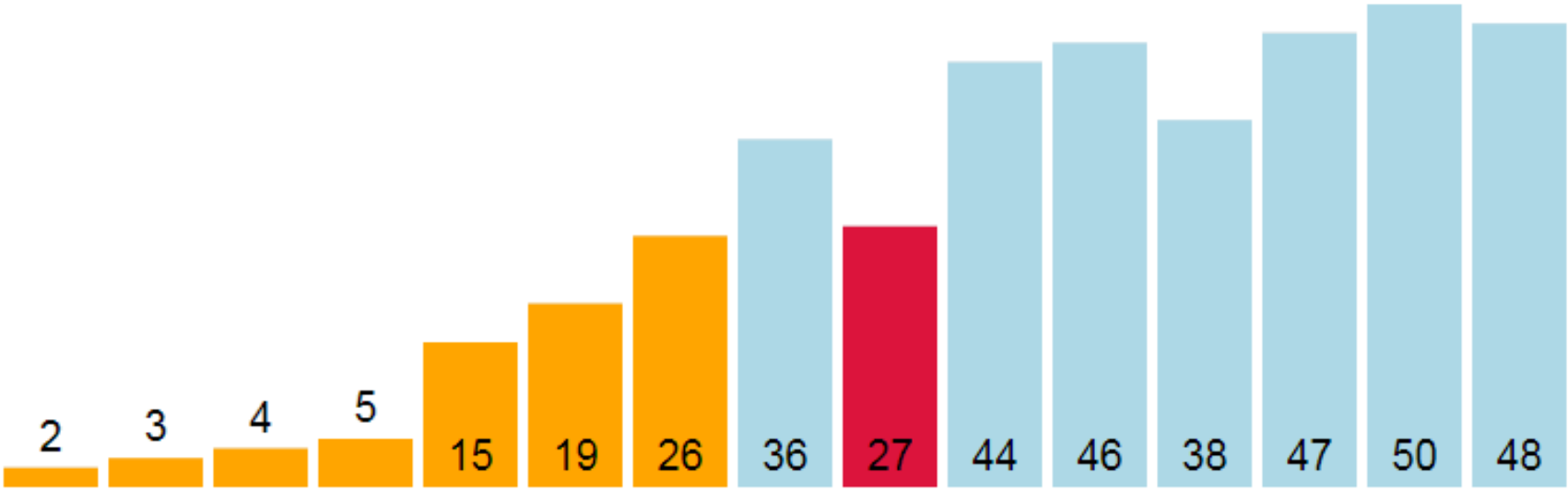
- **Adaptativo**: se o fato da sequência já estar (quase) ordenada melhora a complexidade do algoritmo
  Exemplo: Insertionsort é adaptativo

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Agenda

Ordenação por seleção - Selection Sort



Funcionamento:

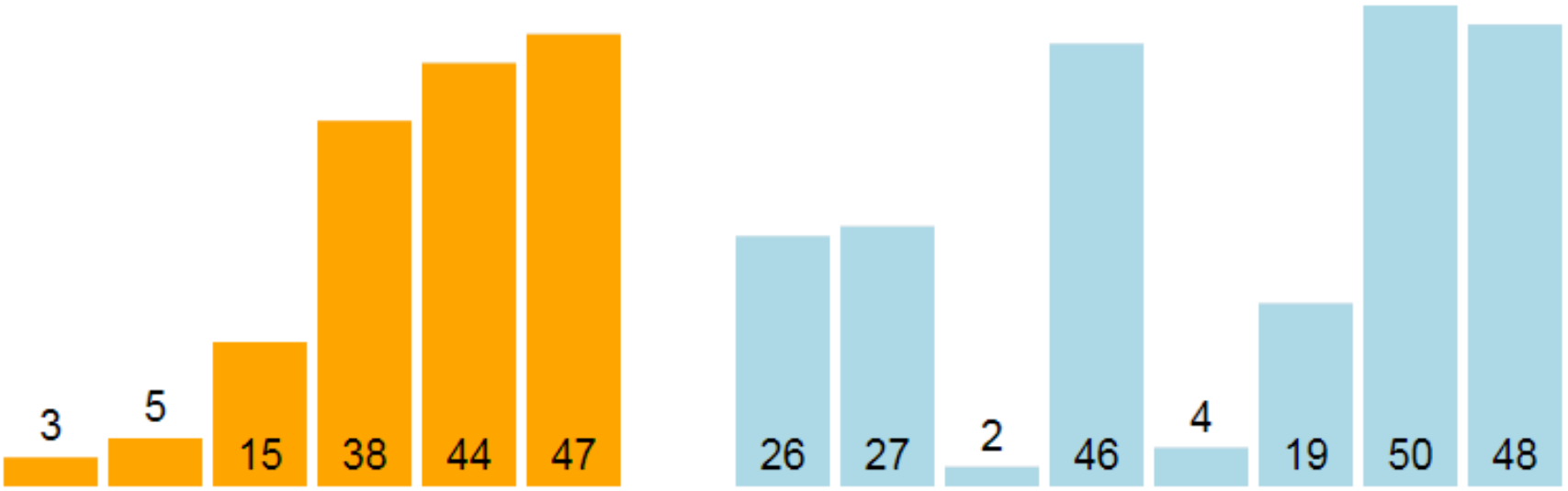
$\forall i = 1 \ldots n-1$

- selecione o mínimo da parte não ordenada
- coloque o mínimo na posição i

```
Set 27 as the new minimum.

repeat (numOfElements - 1) times
  set the first unsorted element as the minimum
  for each of the unsorted elements
    if element < currentMinimum
      set element as new minimum
  swap minimum with first unsorted position
```

https://visualgo.net/en/sorting

Ordenação por inserção - Insertion Sort



Funcionamento:

$\forall i = 2 \ldots n$

- insira o elemento i na posição correta da parte ordenada


```
Extract the first unsorted element (36).
mark first element as sorted
for each unsorted element X
  'extract' the element X
  for j = lastSortedIndex down to 0
    if current element j > X
      move sorted element to the right by 1
    break loop and insert X here
```

https://visualgo.net/en/sorting

Exemplo de Implementação em C

```c
int selectionSort(int *v, int n) {
    int i, j, min, aux;
    long int comp=0;

    for (i=0; i<n-1; i++) {
        min = i;
        for (j=i+1; j<n; j++) {
            if (v[min] > v[j])
                min = j;
            comp++;
        }

        if (min!=i) {
            aux = v[min];
            v[min] = v[i];
            v[i] = aux;
        }
    }

    return comp;
}
```

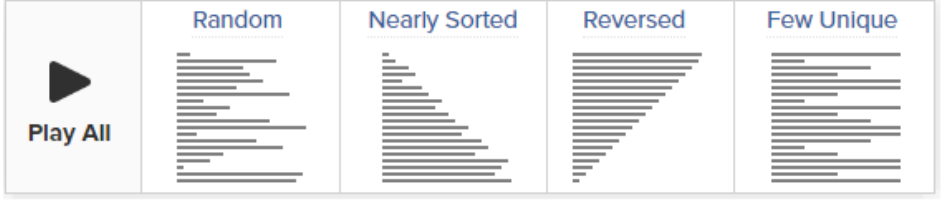
```c
int insertionSort(int v[], int n) {
    int i, j, key;
    long int comp=0;

    for (i=1; i<n; i++) {
        key = v[i];
        j = i-1;
        while(j>=0  && ++comp &&
                        v[j]>key) {
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = key;
    }

    return comp;
}
```

Ver código completo aqui.

Análise Selection Sort

	Comparações	Movimentações
Caso Melhor (sequência em ordem)	n (n - 1) / 2	0
Caso Pior	n (n - 1) / 2	3 (n - 1)

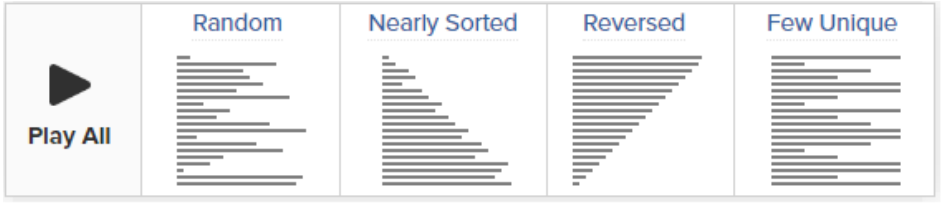
- Apropriado para sequências pequenas com registros grandes
- *In-place*, não estável, não adaptativo **Caso pior?**



Análise Insertion Sort

	Comparações	Movimentações
Caso Melhor (sequência em ordem)	$n - 1$	$2 (n - 1)$
Caso Pior (sequência em ordem reversa)	$n (n - 1) / 2$	$n^2/2 + 3n/2 - 2$

- Apropriado para sequências pequenas com registros pequenos ou para inserir poucos registros em uma sequência já ordenada
- *In-place*, estável, adaptativo



Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Agenda

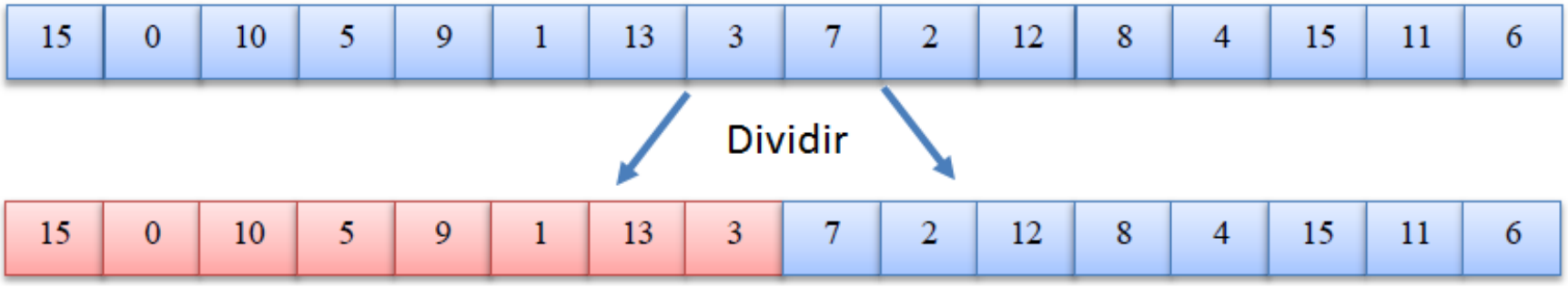
Mergesort - Ordenação por Intercalação

- inventado por John von Neumann em 1945
- algoritmo simples, de propósito geral, ótimo, estável
- exemplo clássico de algoritmo recursivo de divisão e conquista

15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6

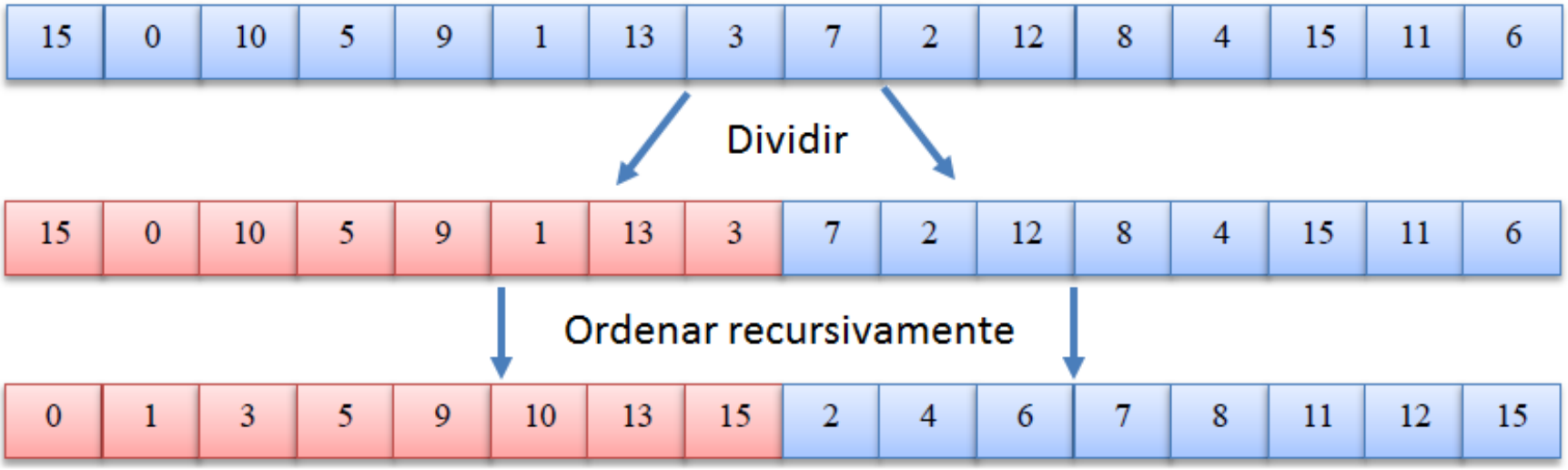
Mergesort - Ordenação por Intercalação

- inventado por John von Neumann em 1945
- algoritmo simples, de propósito geral, ótimo, estável
- exemplo clássico de algoritmo recursivo de divisão e conquista



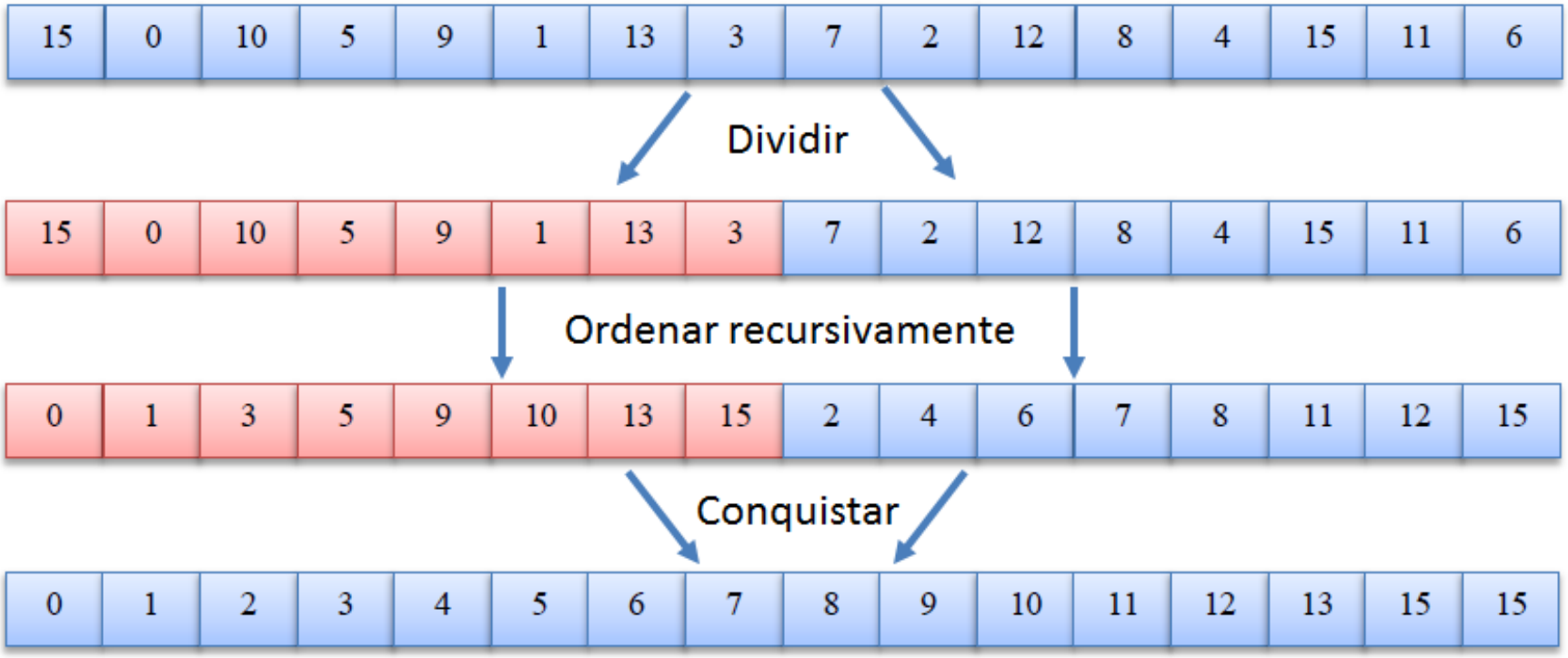
Mergesort - Ordenação por Intercalação

- inventado por John von Neumann em 1945
- algoritmo simples, de propósito geral, ótimo, estável
- exemplo clássico de algoritmo recursivo de divisão e conquista



Mergesort - Ordenação por Intercalação

- inventado por John von Neumann em 1945
- algoritmo simples, de propósito geral, ótimo, estável
- exemplo clássico de algoritmo recursivo de divisão e conquista

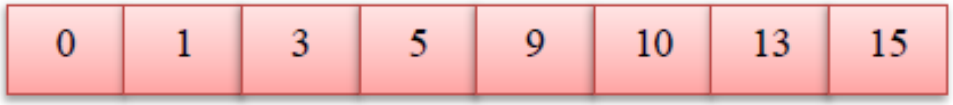


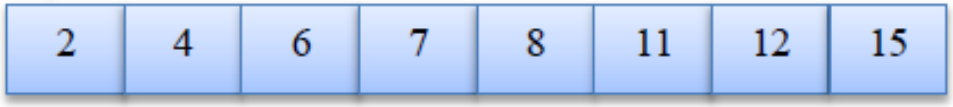
Mergesort - Ordenação por Intercalação

- inventado por John von Neumann em 1945
- algoritmo simples, de propósito geral, ótimo, estável
- exemplo clássico de algoritmo recursivo de divisão e conquista
- A conquista é feita usando intercalação

Intercalação

Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado

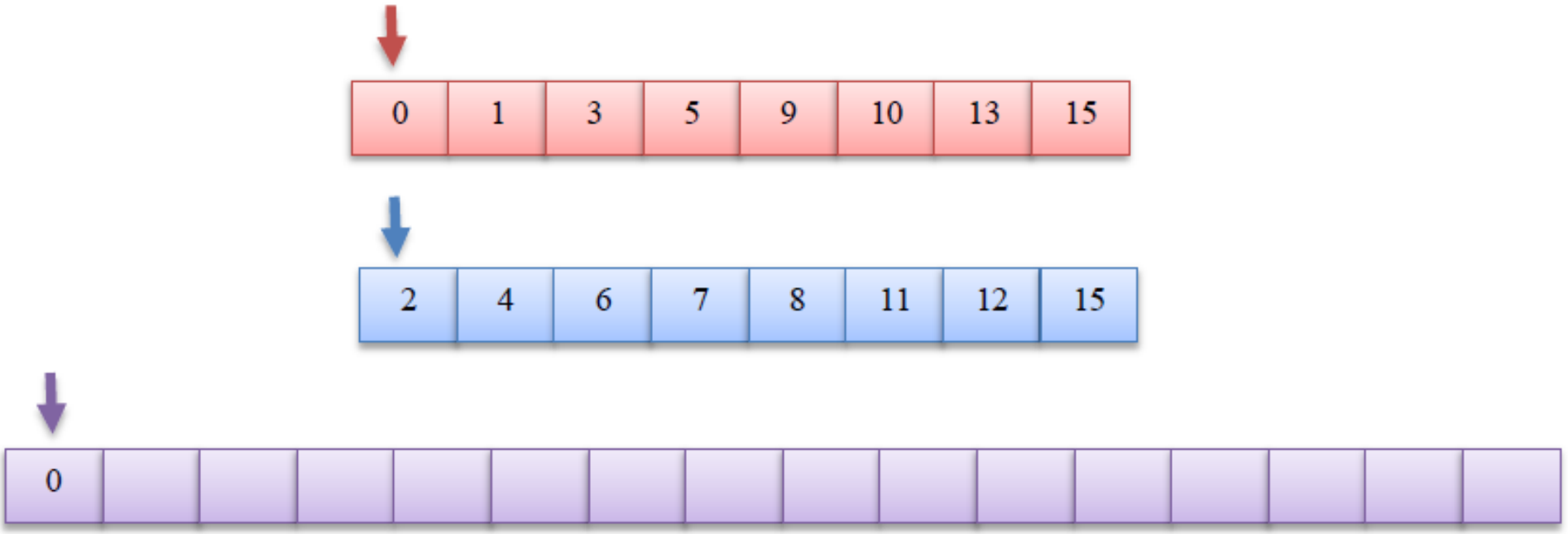




- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

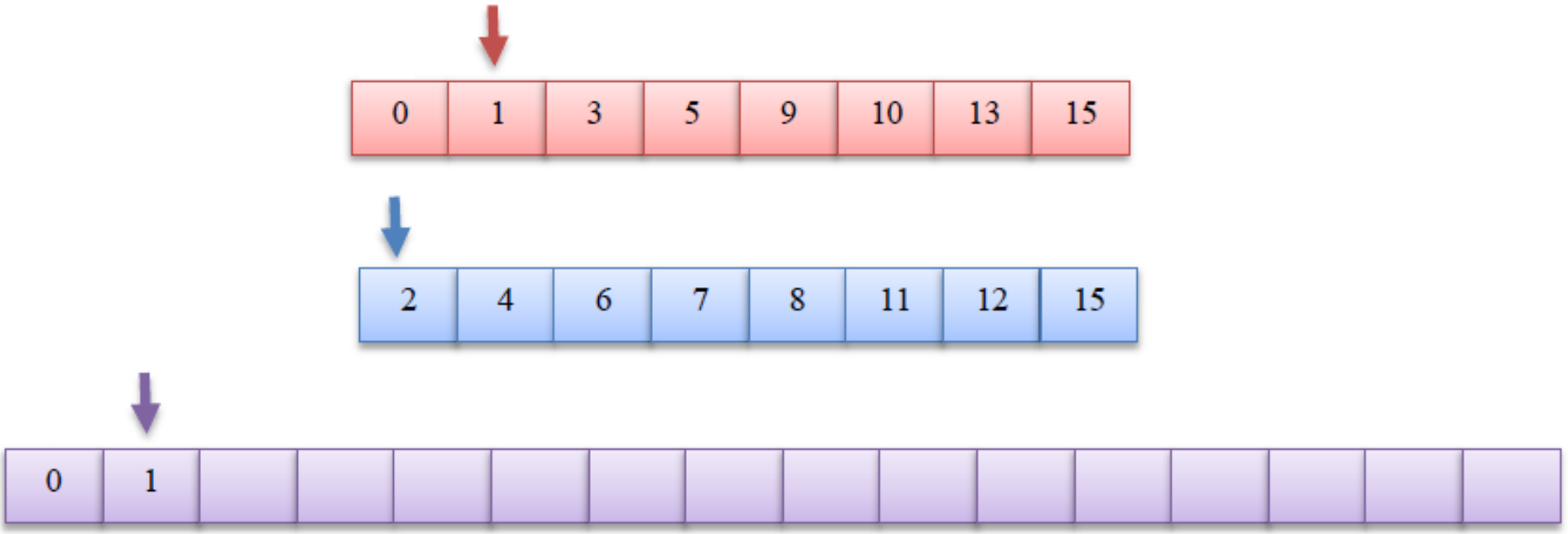
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

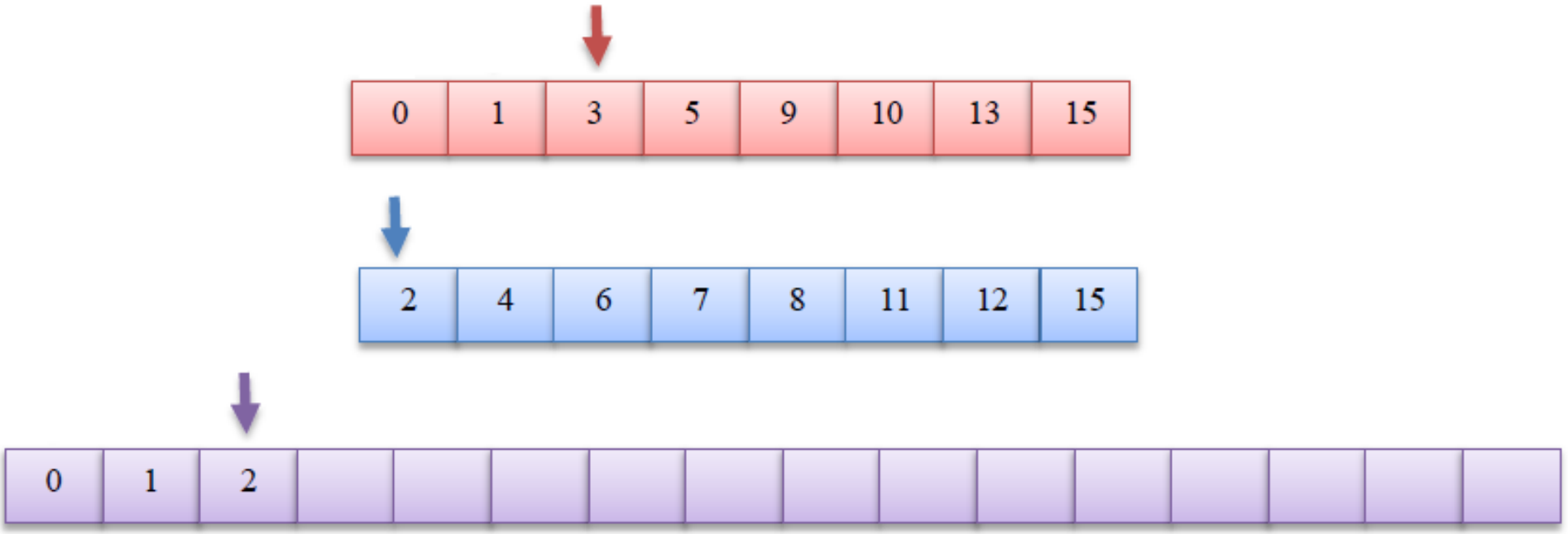
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

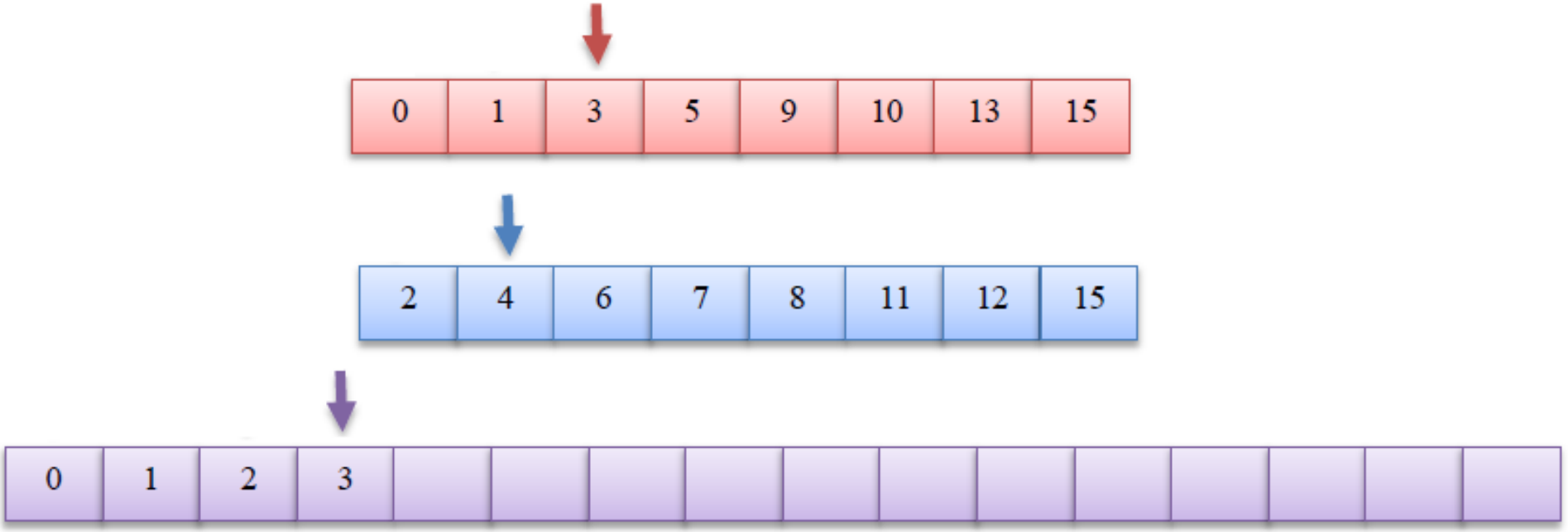
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

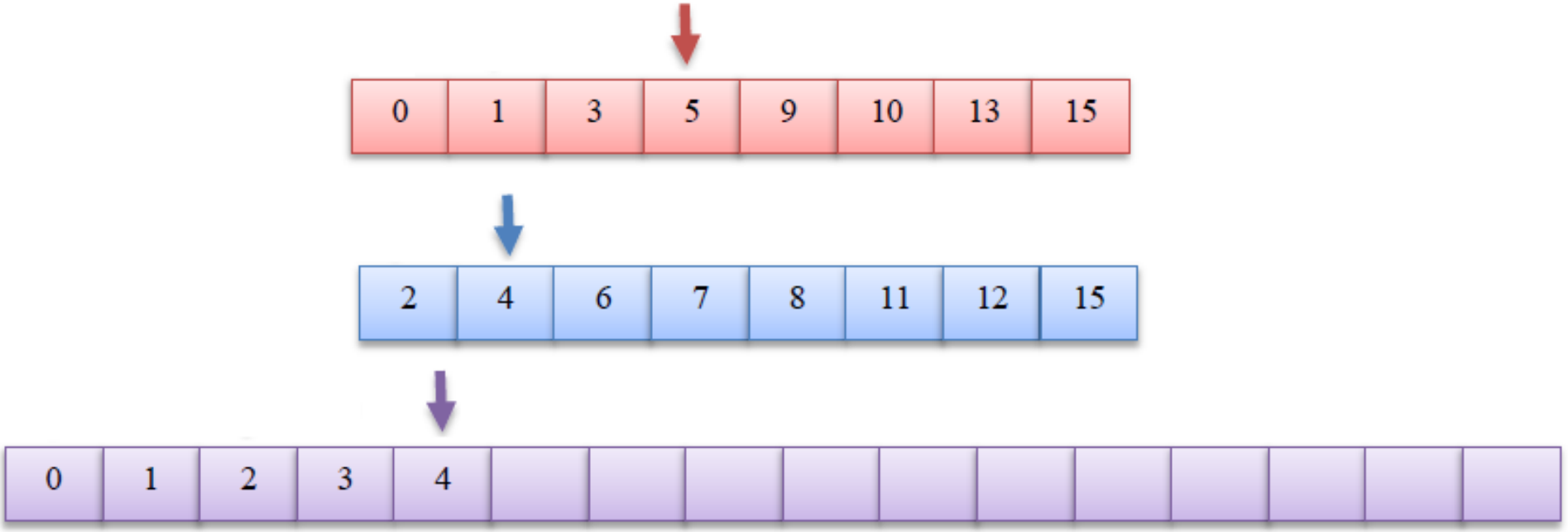
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

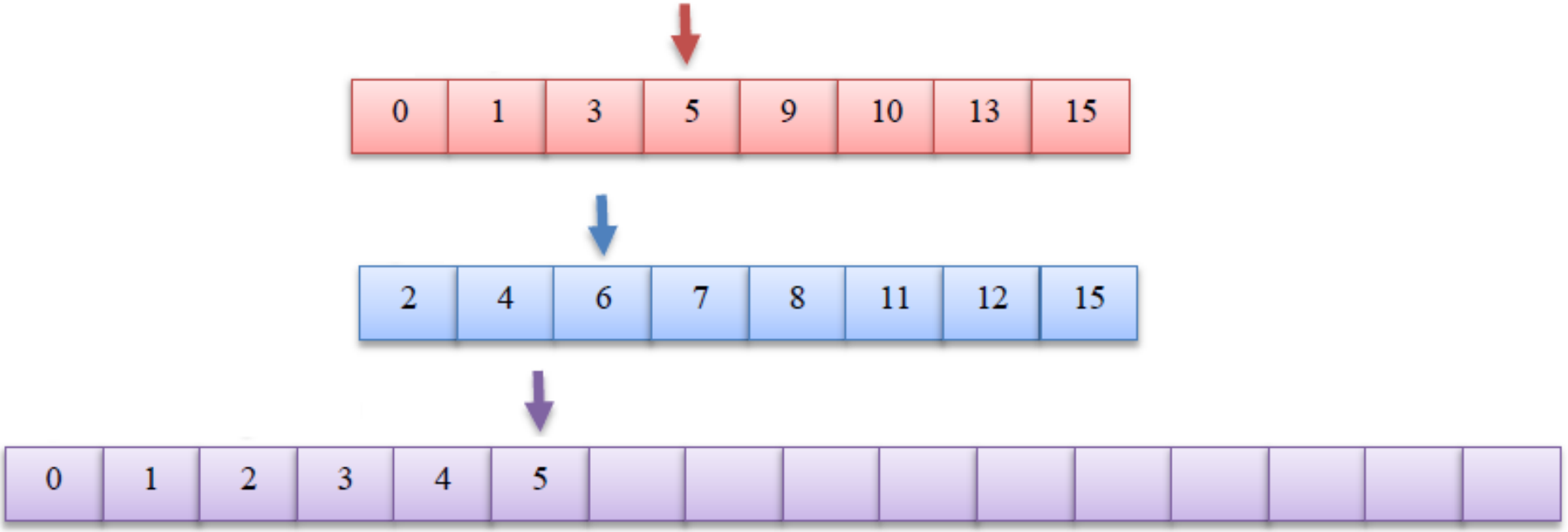
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

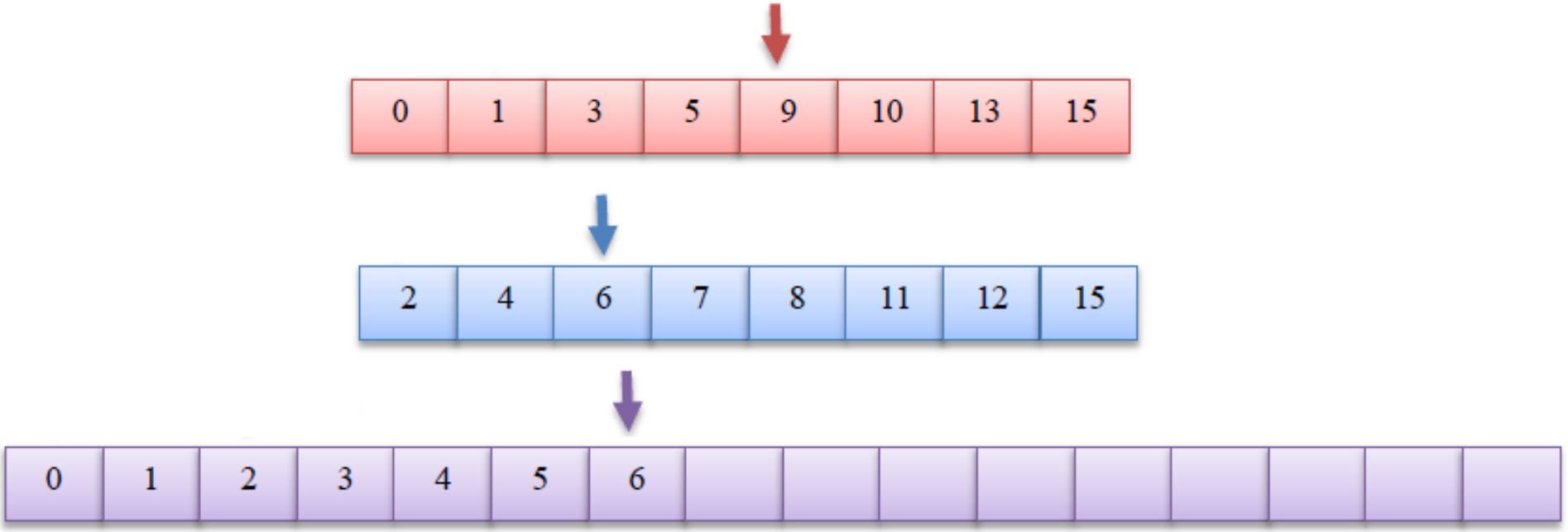
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

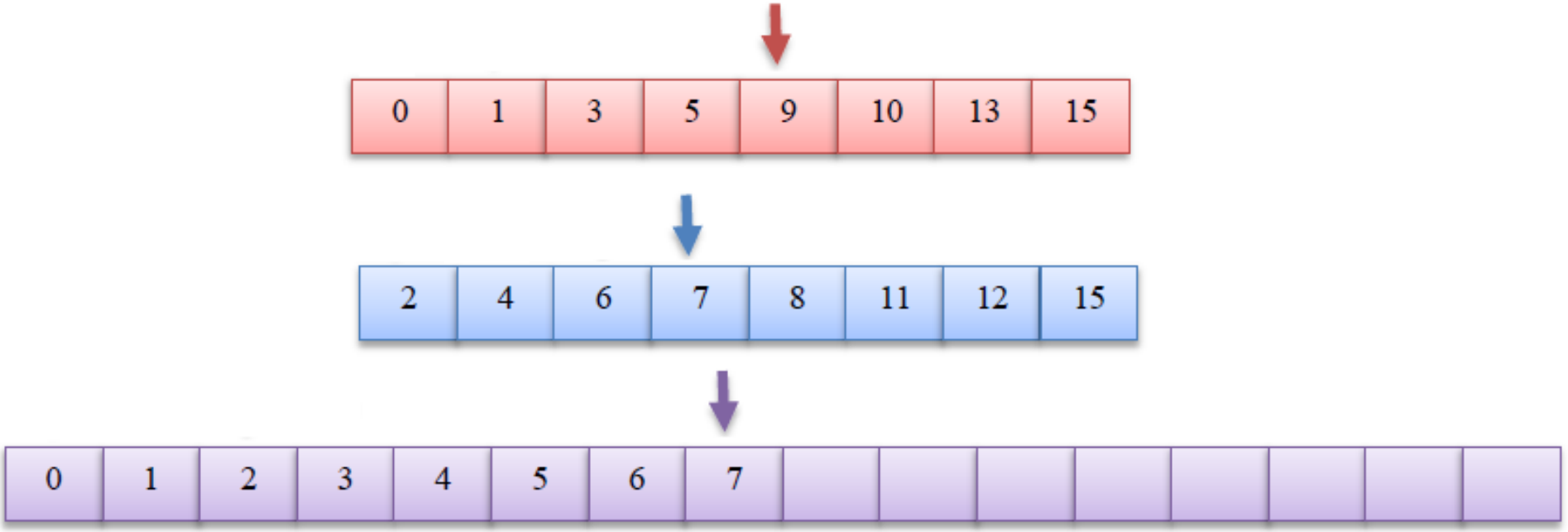
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

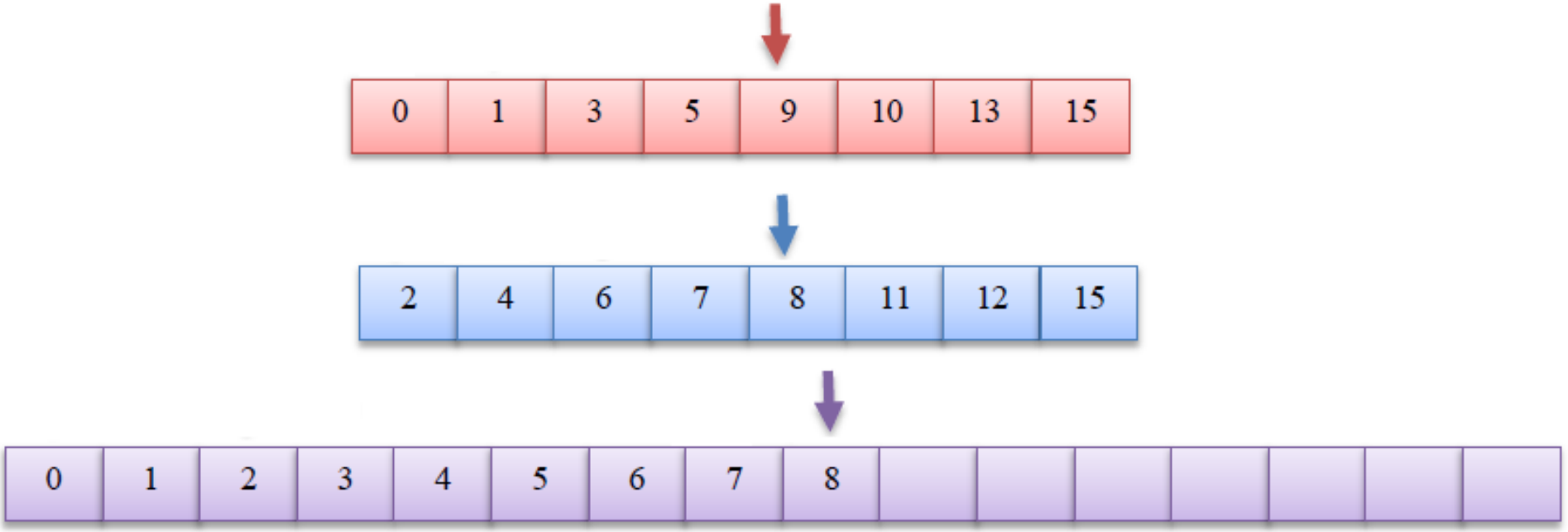
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

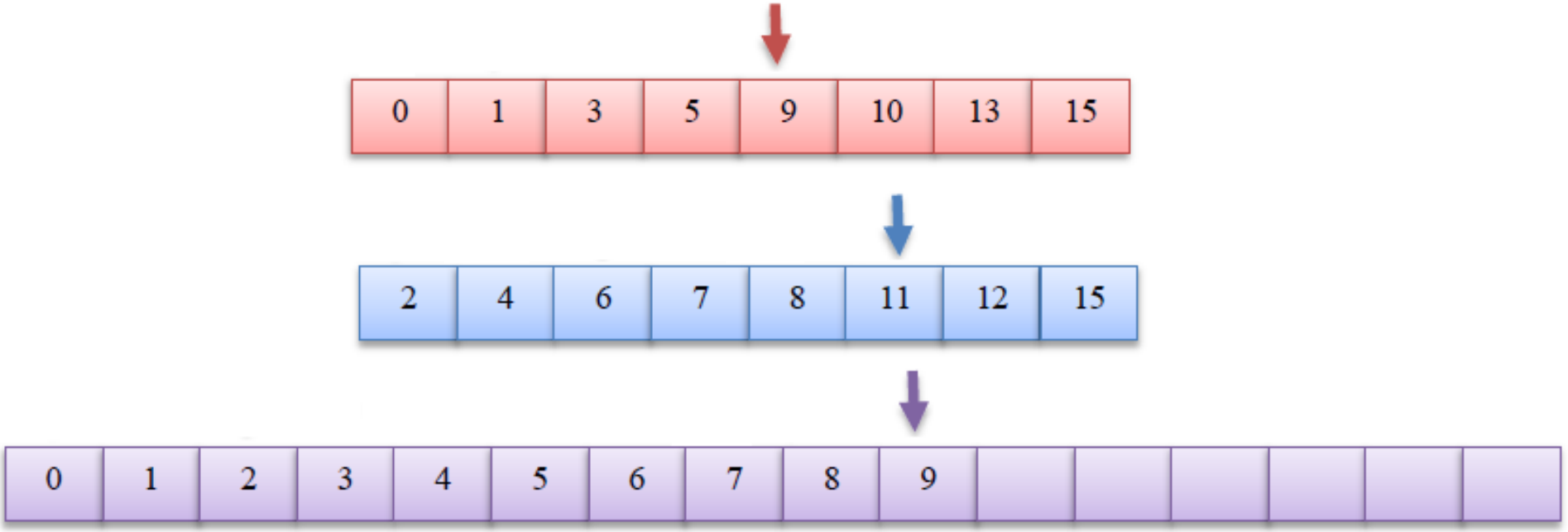
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

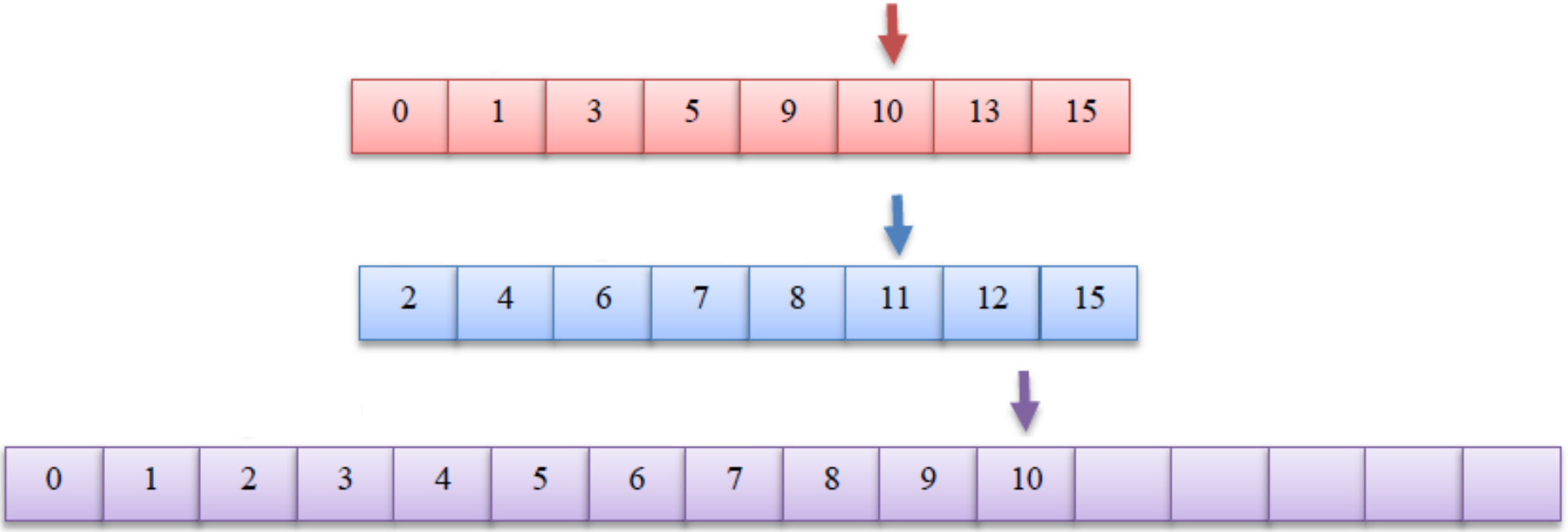
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

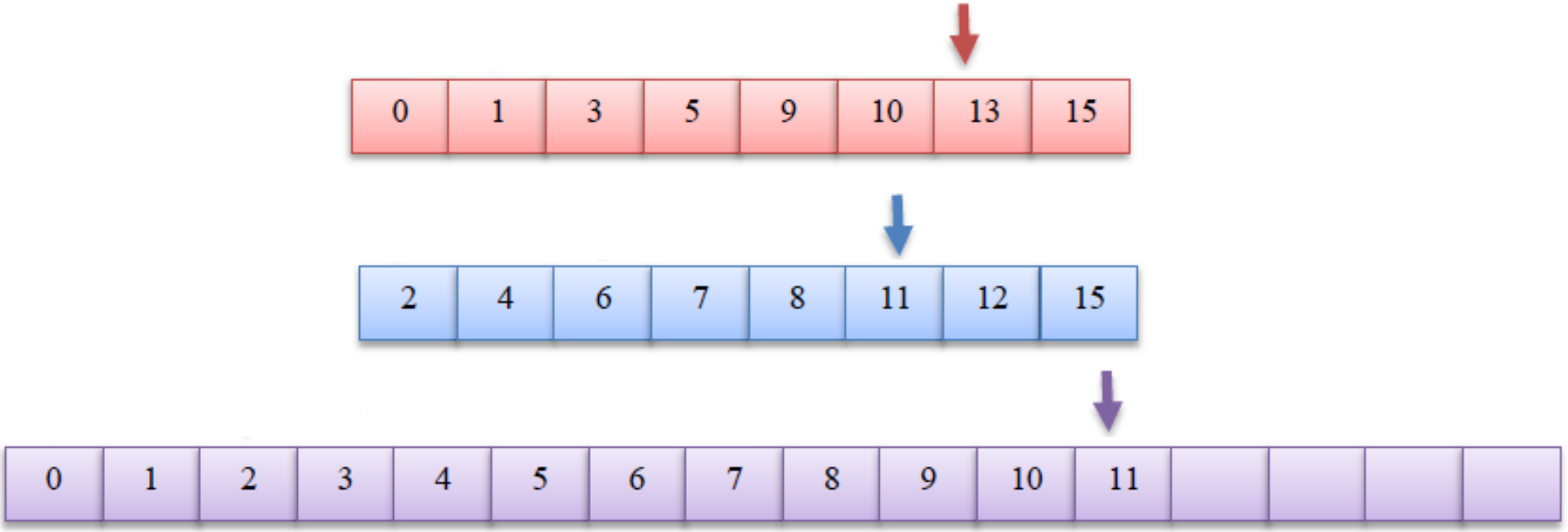
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

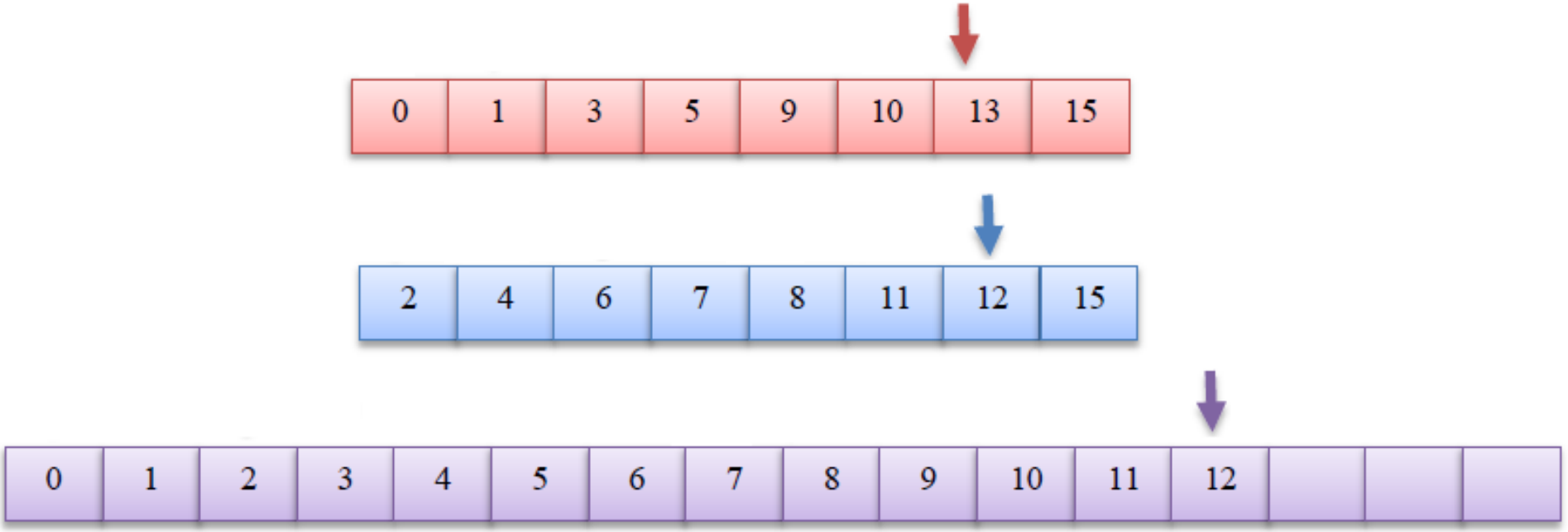
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

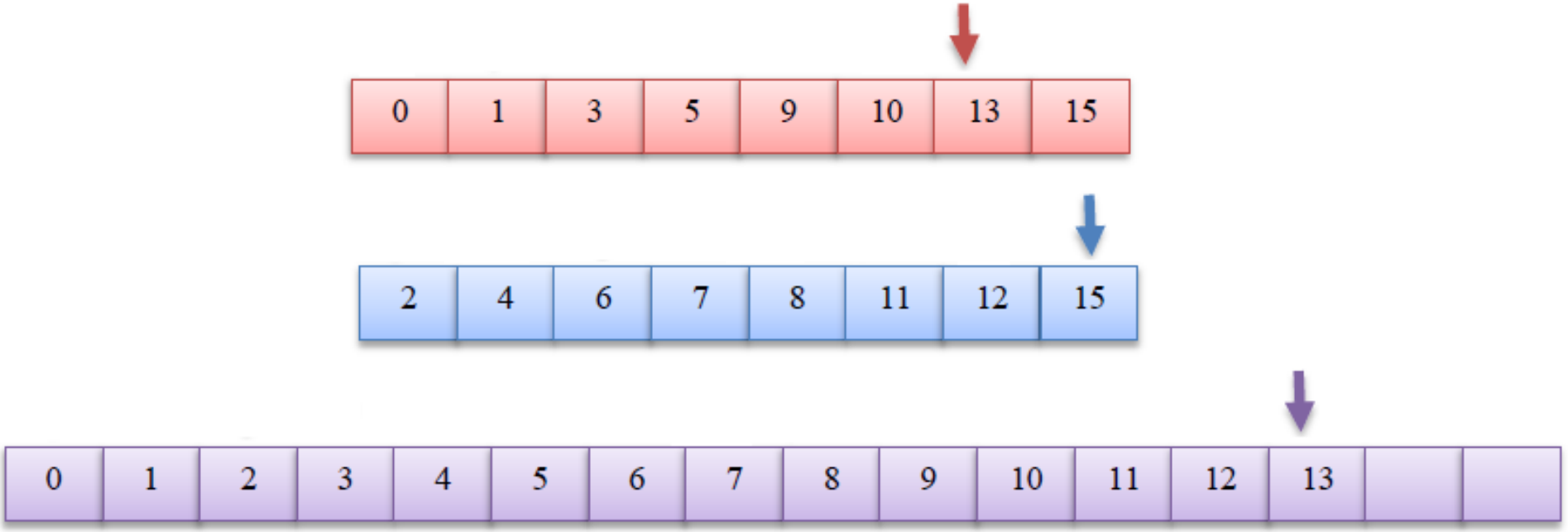
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

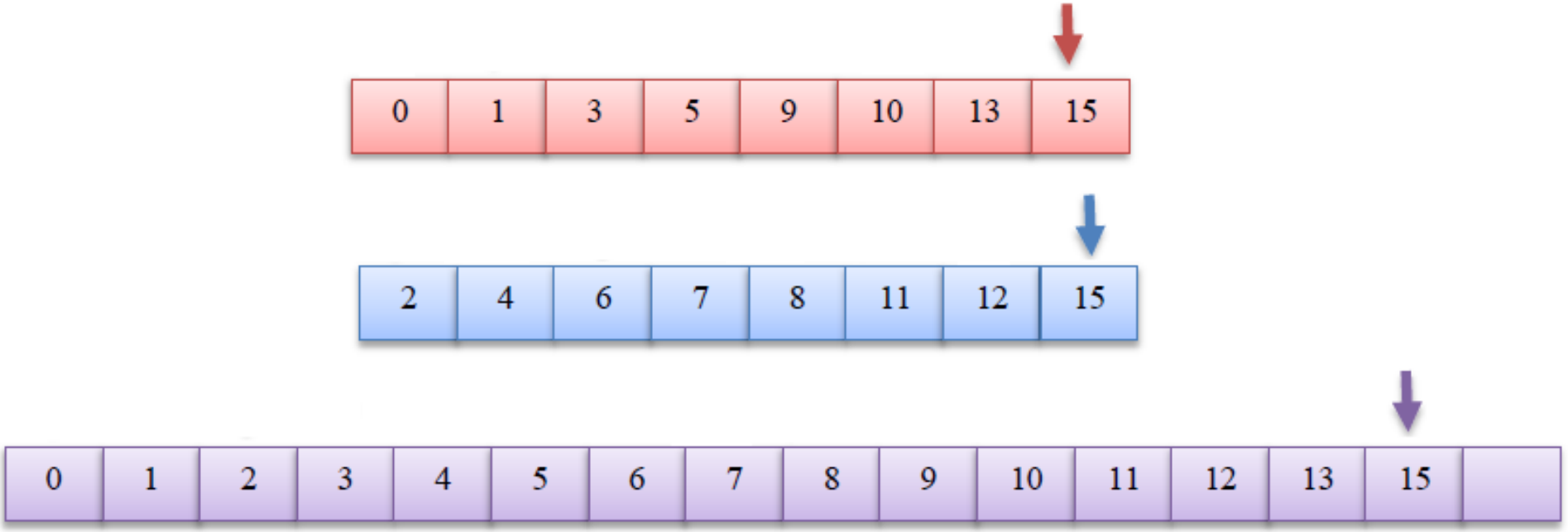
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

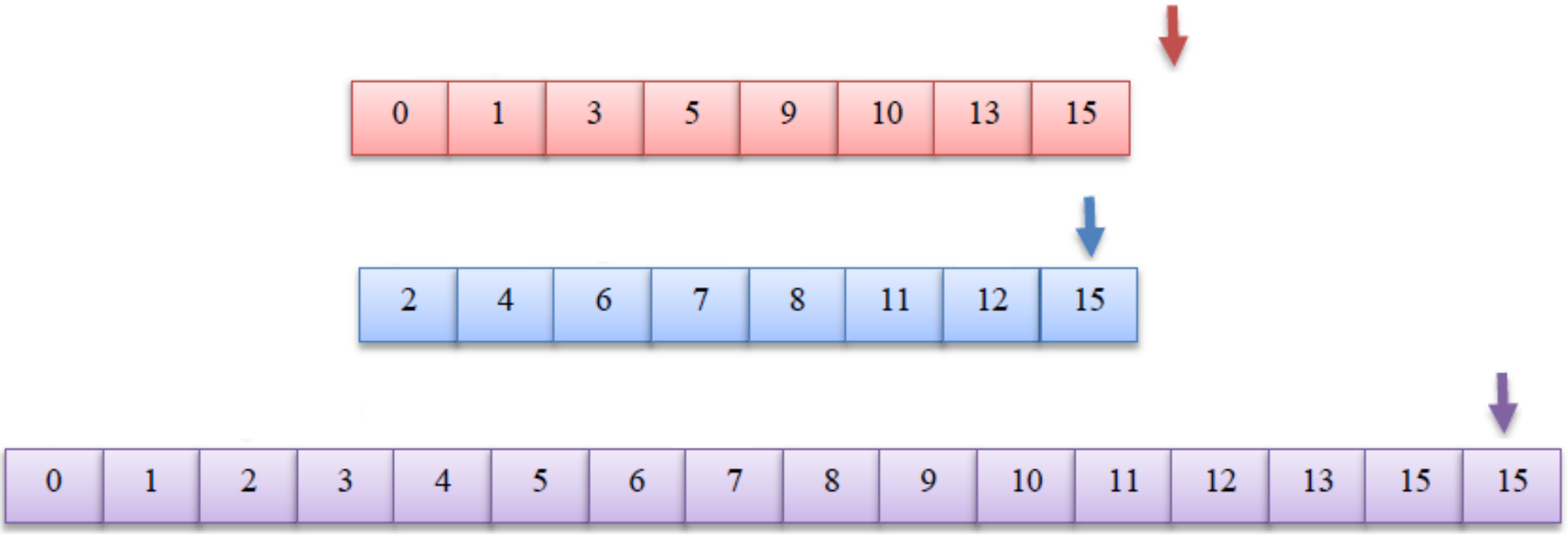
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

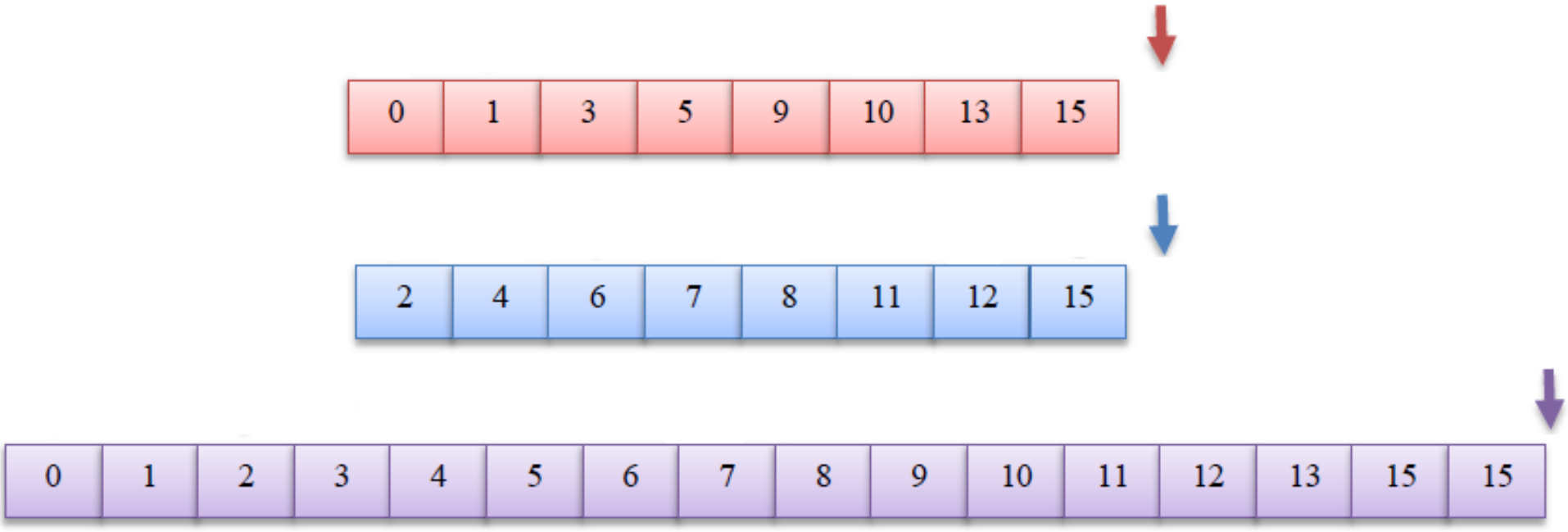
Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Intercalação

Combinação de dois ou mais arranjos (arquivos, listas) de entrada ordenados num único vetor ordenado



- Comparar os primeros de cada vetor de entrada e copiar o menor para o vetor de saída

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Algoritmo de Intercalação

- Implementação simples usando memória auxiliar
- Uma das duas listas vai acabar primeiro; passar diretamente os restantes.
- Ao intercalar elementos iguais dar prioridade ao primeiro vetor; isto garante a estabilidade da ordenação
- O teste de final de vetor pode ser eliminado colocando inicialmente o segundo subarranjo invertido (**Exercício independente**)

- A memória auxiliar pode ser de tamanho $n/2$ (**Exercício independente**)
- existem variantes que usam memória auxiliar $O(1)$ (*in-place*); porém ou são mais complexas e/ou menos ineficientes

Algoritmo de Ordenação por Intercalação (Mergesort)

- Dividir o vetor em duas partes de tamanhos similares e ordenar recursivamente
- intercalar os sub-arrajos ordenados

```
long int MergeSort(int v[], int lo, int hi) {
  long int comp = 0;

  if (lo < hi) {
    int mid = lo + (hi - lo) / 2;
    comp = MergeSort(v, lo, mid);
    comp += MergeSort(v, mid+1, hi);
    comp += merge(v, lo, mid, hi);
  }

  return comp;
}
```

https://repl.it/@mirthalina/optimalSortingAlgorithms

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6

sort(0,15)

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6

15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6

sort(0,15) → sort(0,7)

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6

sort(0,15) → sort(0,7) → sort(0,3)

Universidade Federal do ABC

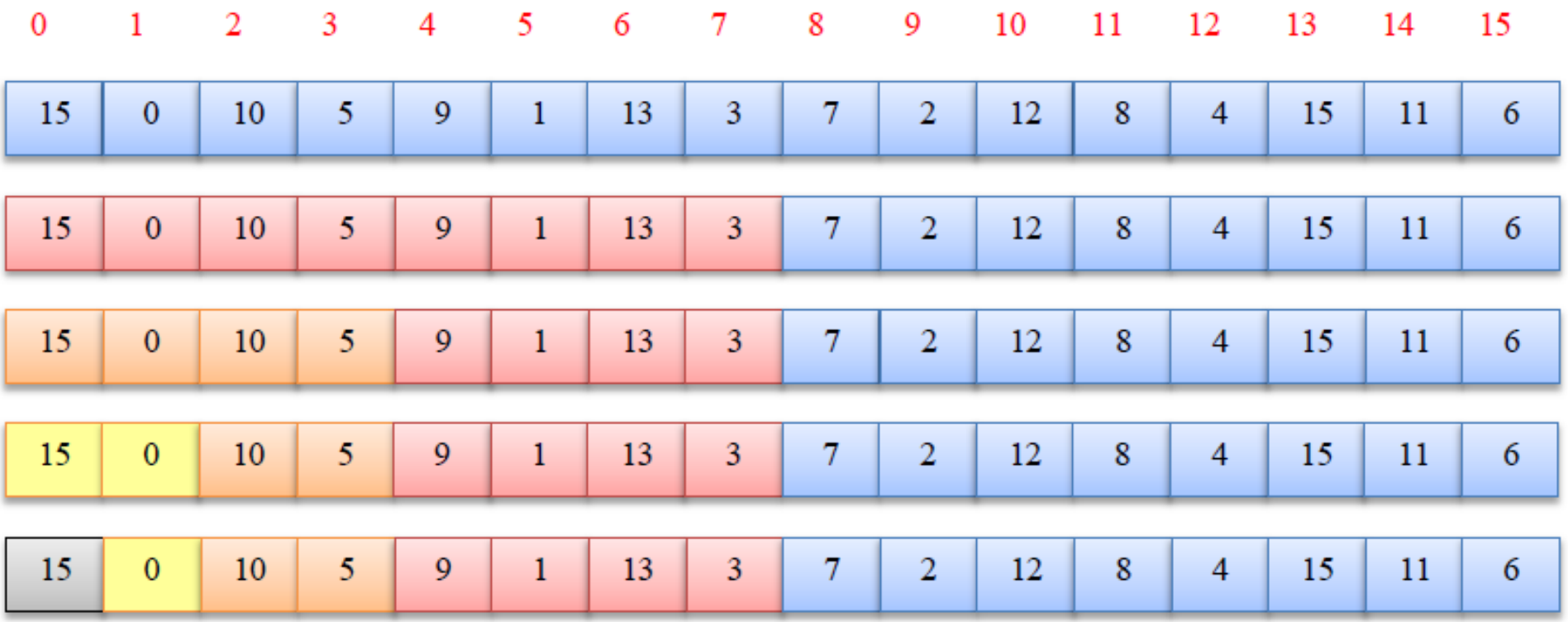
CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6

sort(0,15) → sort(0,7) → sort(0,3) → sort(0,1)

Exemplo Mergesort



sort(0,15) → sort(0,7) → sort(0,3) → sort(0,1) → sort(0,0); sort(1,1); merge(0,1)

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
0	15	5	10	9	1	13	3	7	2	12	8	4	15	11	6

sort(0,15) → sort(0,7) → sort(0,3) → sort(2,3) → sort(2,2); sort(2,3); merge(2,3)

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
15	0	10	5	9	1	13	3	7	2	12	8	4	15	11	6
0	15	5	10	9	1	13	3	7	2	12	8	4	15	11	6
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6

sort(0,15) → sort(0,7) → sort(0,3) → merge(0,3)

Exemplo Mergesort

sort(0,15) → sort(0,7) → sort(4,7)

Universidade Federal do ABC

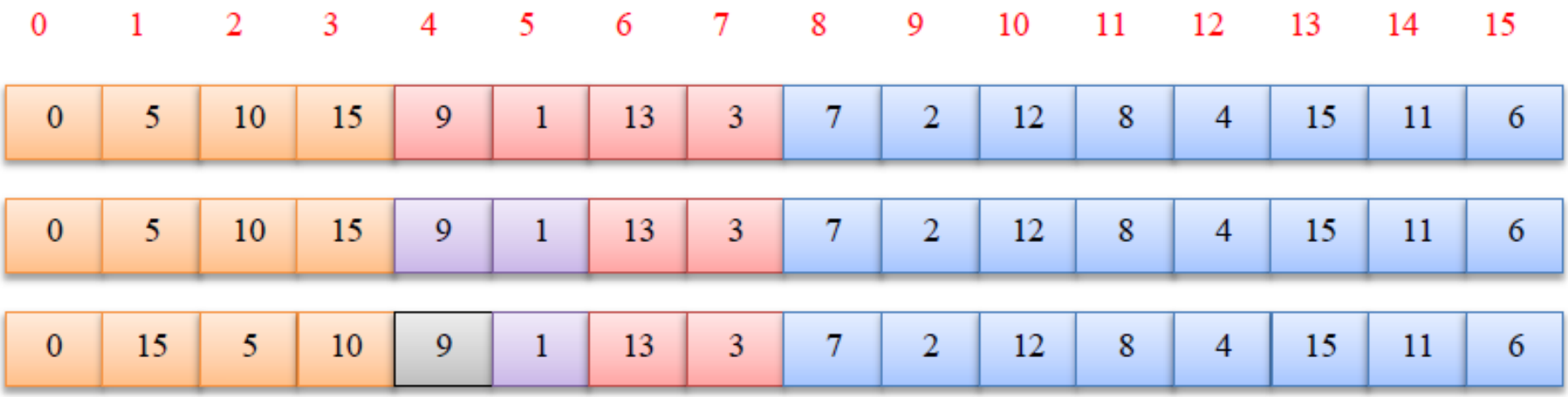
CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6

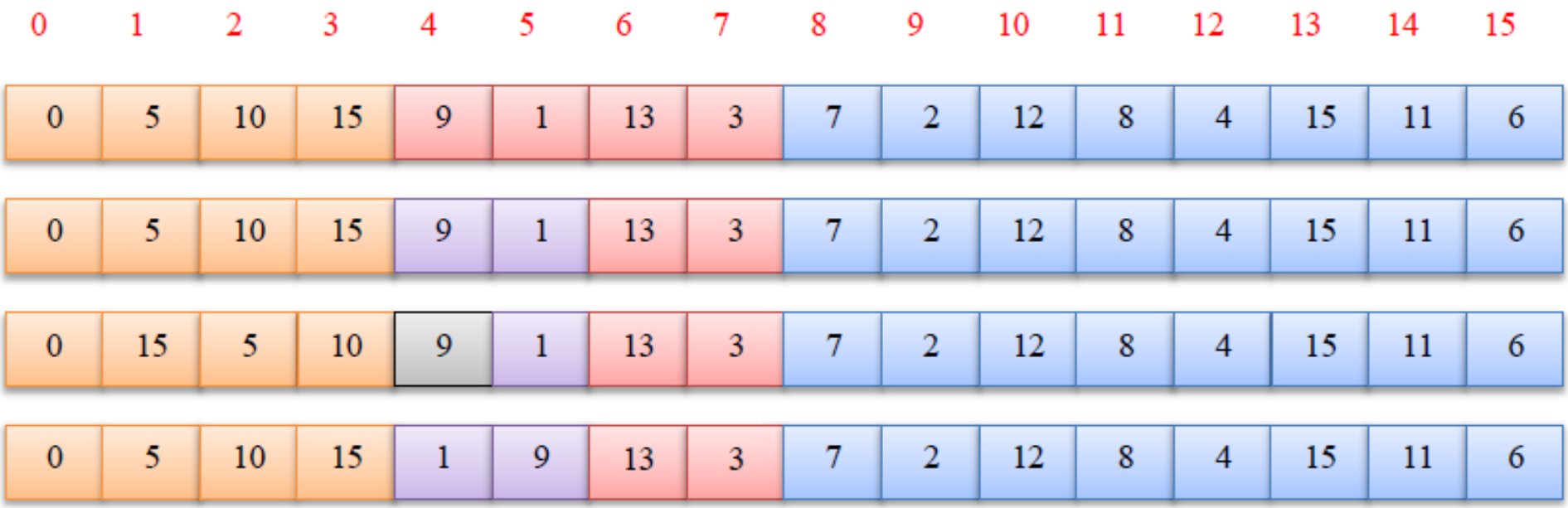
sort(0,15) → sort(0,7) → sort(4,7) → sort(4,5);

Exemplo Mergesort



sort(0,15) → sort(0,7) → sort(4,7) → sort(4,5); → sort(4,4); sort(5,5); merge(4,5)

Exemplo Mergesort



sort(0,15) → sort(0,7) → sort(4,7) → sort(6,7)→ sort(6,6); sort(7,7); merge(6,7)

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6
0	15	5	10	9	1	13	3	7	2	12	8	4	15	11	6
0	5	10	15	1	9	13	3	7	2	12	8	4	15	11	6
0	5	10	15	1	9	3	13	7	2	12	8	4	15	11	6

sort(0,15) → sort(0,7) → sort(4,7) → merge(4,7)

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6
0	15	5	10	9	1	13	3	7	2	12	8	4	15	11	6
0	5	10	15	1	9	13	3	7	2	12	8	4	15	11	6
0	5	10	15	1	9	3	13	7	2	12	8	4	15	11	6
0	5	10	15	1	3	9	13	7	2	12	8	4	15	11	6

sort(0,15) → sort(0,7) → merge(0,7)

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6
0	5	10	15	9	1	13	3	7	2	12	8	4	15	11	6
0	15	5	10	9	1	13	3	7	2	12	8	4	15	11	6
0	5	10	15	1	9	13	3	7	2	12	8	4	15	11	6
0	5	10	15	1	9	3	13	7	2	12	8	4	15	11	6
0	5	10	15	1	3	9	13	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6

sort(0,15) → sort(0,7)

Exemplo Mergesort

sort(0,15) $\rightarrow$ sort(0,7)

Exemplo Mergesort

sort(0,15) $\rightarrow$ sort(0,7)

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6

sort(0,15) $\rightarrow$ sort(0,7)

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	2	7	8	12	4	15	11	6

sort(0,15) $\rightarrow$ sort(0,7)

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	2	7	8	12	4	15	11	6
0	1	3	5	9	10	13	15	2	7	8	12	4	15	6	11

sort(0,15) $\rightarrow$ sort(0,7)

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	2	7	8	12	4	15	11	6
0	1	3	5	9	10	13	15	2	7	8	12	4	15	6	11
0	1	3	5	9	10	13	15	2	4	6	7	8	11	12	15

sort(0,15) $\rightarrow$ sort(0,7)

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Exemplo Mergesort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	7	2	12	8	4	15	11	6
0	1	3	5	9	10	13	15	2	7	8	12	4	15	11	6
0	1	3	5	9	10	13	15	2	7	8	12	4	15	6	11
0	1	3	5	9	10	13	15	2	4	6	7	8	11	12	15
0	1	2	3	4	5	6	7	8	9	10	11	12	13	15	15

sort(0,15) $\rightarrow$ sort(0,7)

Análise do Mergesort

O vetor é sempre dividido na metade. No caso pior, o tamanho do vetor inicial é potência de 2. Para a intercalação temos,

- **Caso pior**: Todos os elementos são comparados no mínimo uma vez (os dois maiores não estão no mesmo sub-vetor)
- **Caso melhor**: Quando metade dos elementos são comparados (o maior de um vetor é menor que o primeiro do outro)

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

A recorrência geral tem a forma $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(N)$. A mesma solução é obtida usando indução.

Análise do algoritmo - Caso pior

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

Análise do algoritmo - Caso pior

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

Relembrando a variante do Teorema Mestre:

$$T(n) = aT(\frac{n}{b}) + n^c$$

Análise do algoritmo - Caso pior

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

Relembrando a variante do Teorema Mestre:

$$T(n) = aT(\frac{n}{b}) + n^c$$

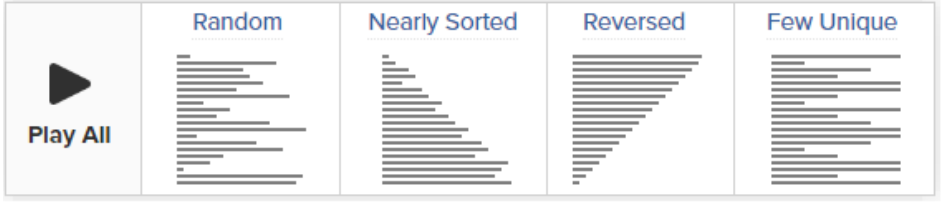
Neste caso $a = 1, b = 2, \log_b^a = 1 = c$. Logo, aplica-se o caso:

2. $\log_b^a = c \Rightarrow T(n) = \Theta(n^c * \log_b^n)$

$$\Rightarrow T(n) = \Theta(n * \log n)$$

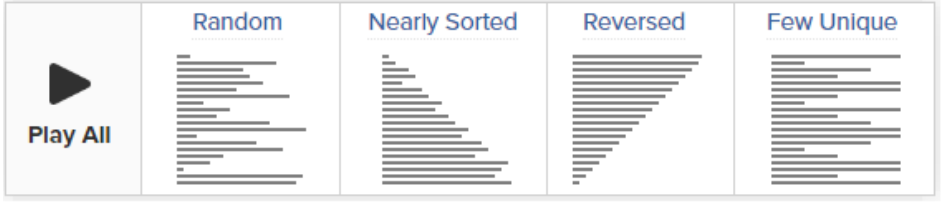
Aprimorando o algoritmo

O mergesort não é *in-place* nem adaptativo



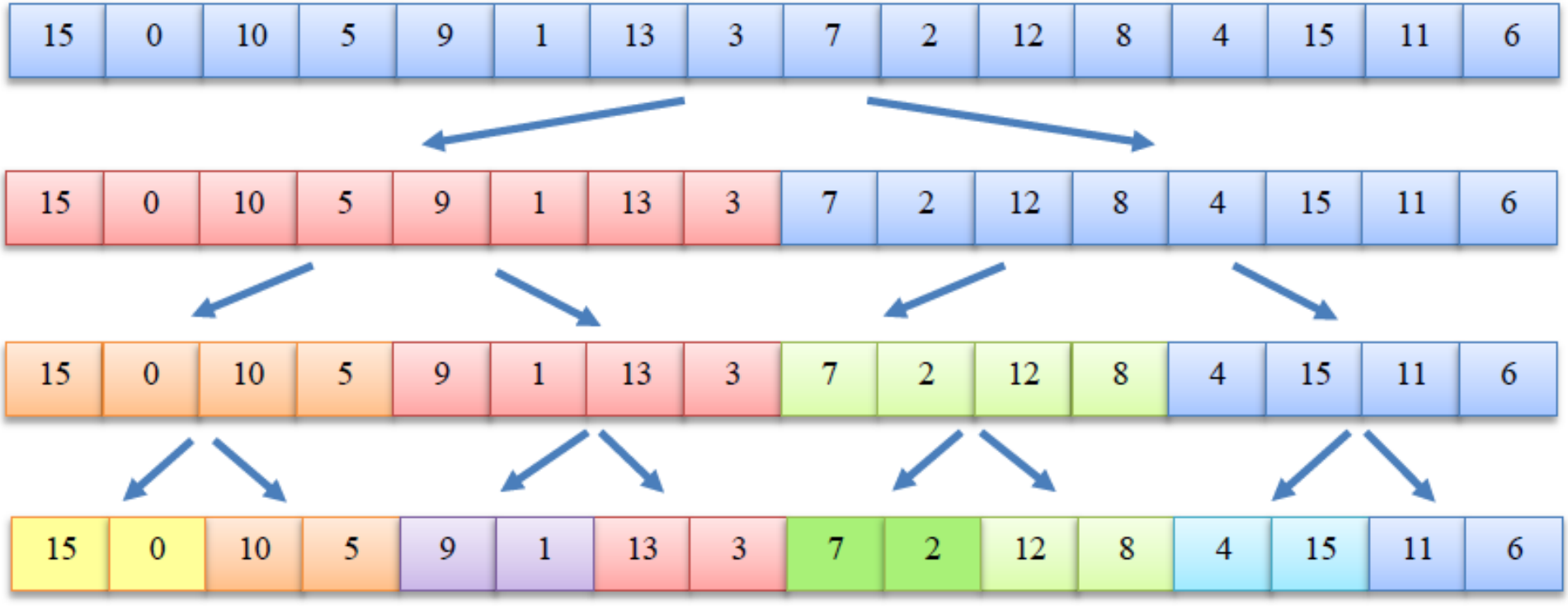
Aprimorando o algoritmo

O mergesort não é *in-place* nem adaptativo



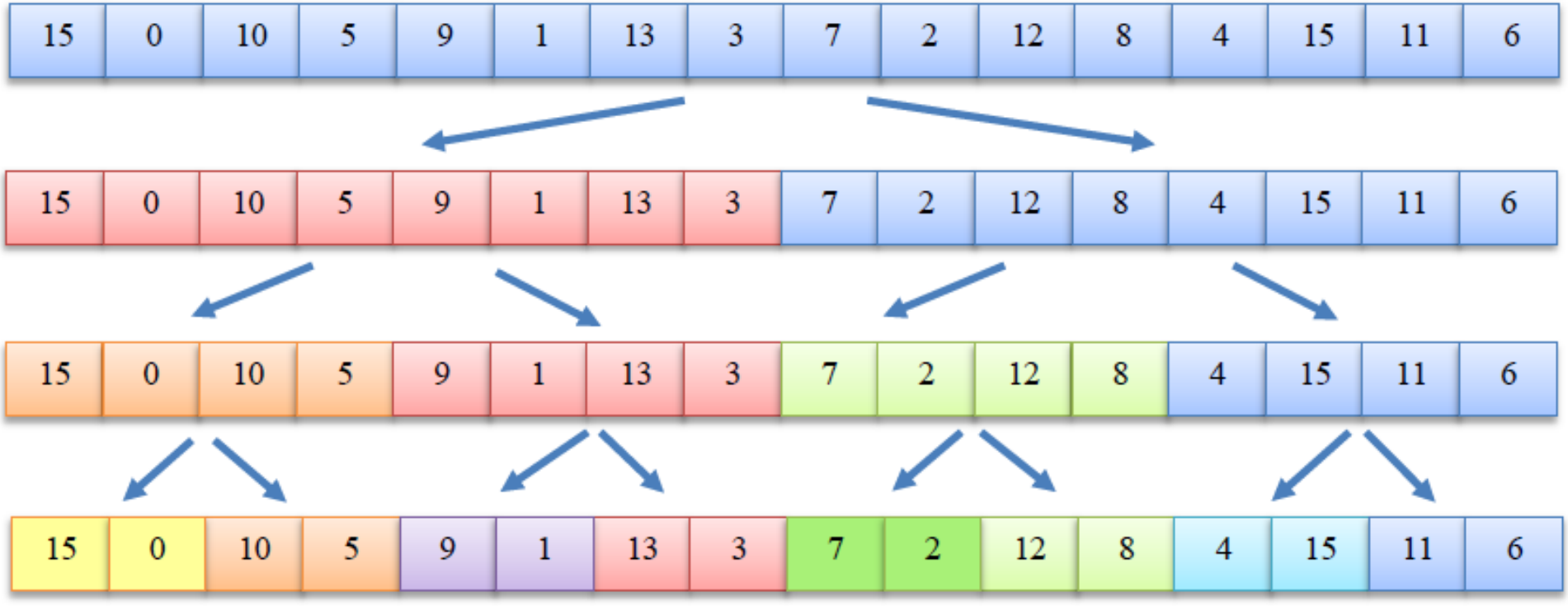
- Antes de intercalar testar se o vetor já está ordenado. Isso evita a intercalação mas não as chamadas recursivas (**como?**)
- Usar inserção para arranjos pequenos (e.g. tamanho < 15)
- É possível eliminar a copia ao vetor auxiliar durante a intercalação (reduz o tempo mas não o espaço)

Árvore de Partições



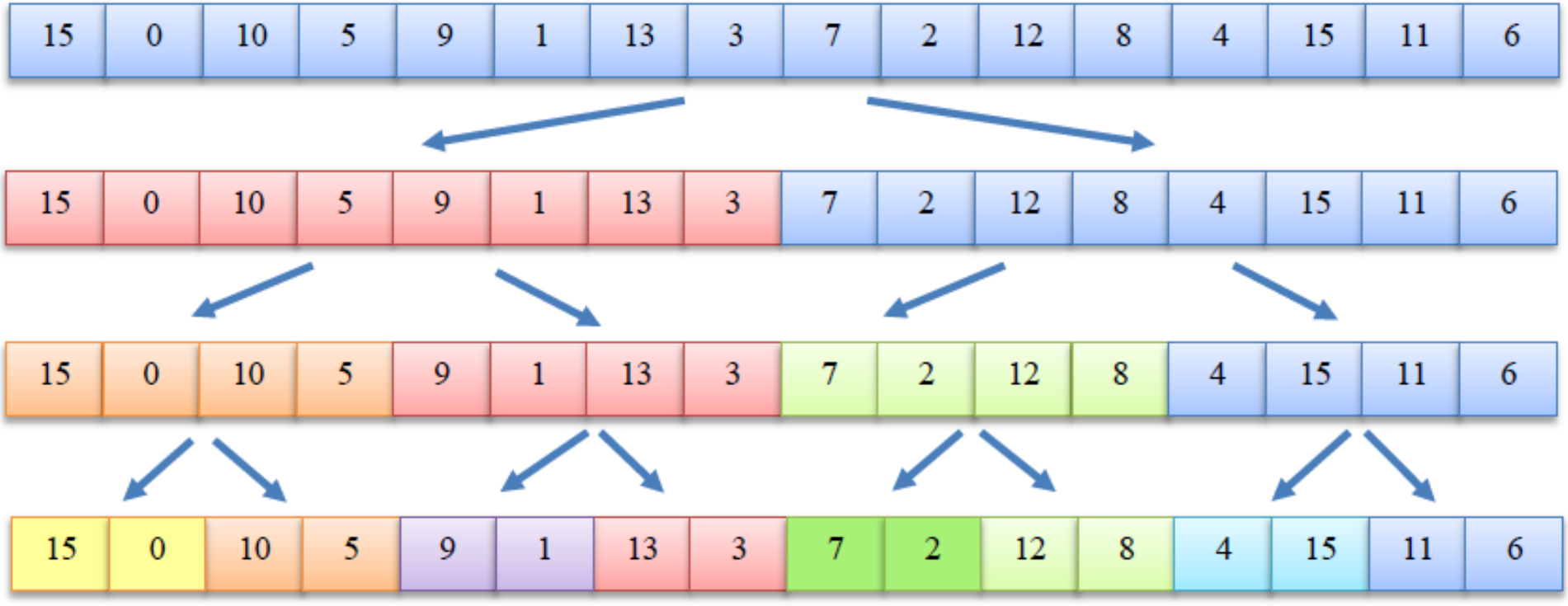
A recursividade é usada para definir as partições do vetor que precisam ser intercaladas e a ordem das intercalações

Árvore de Partições



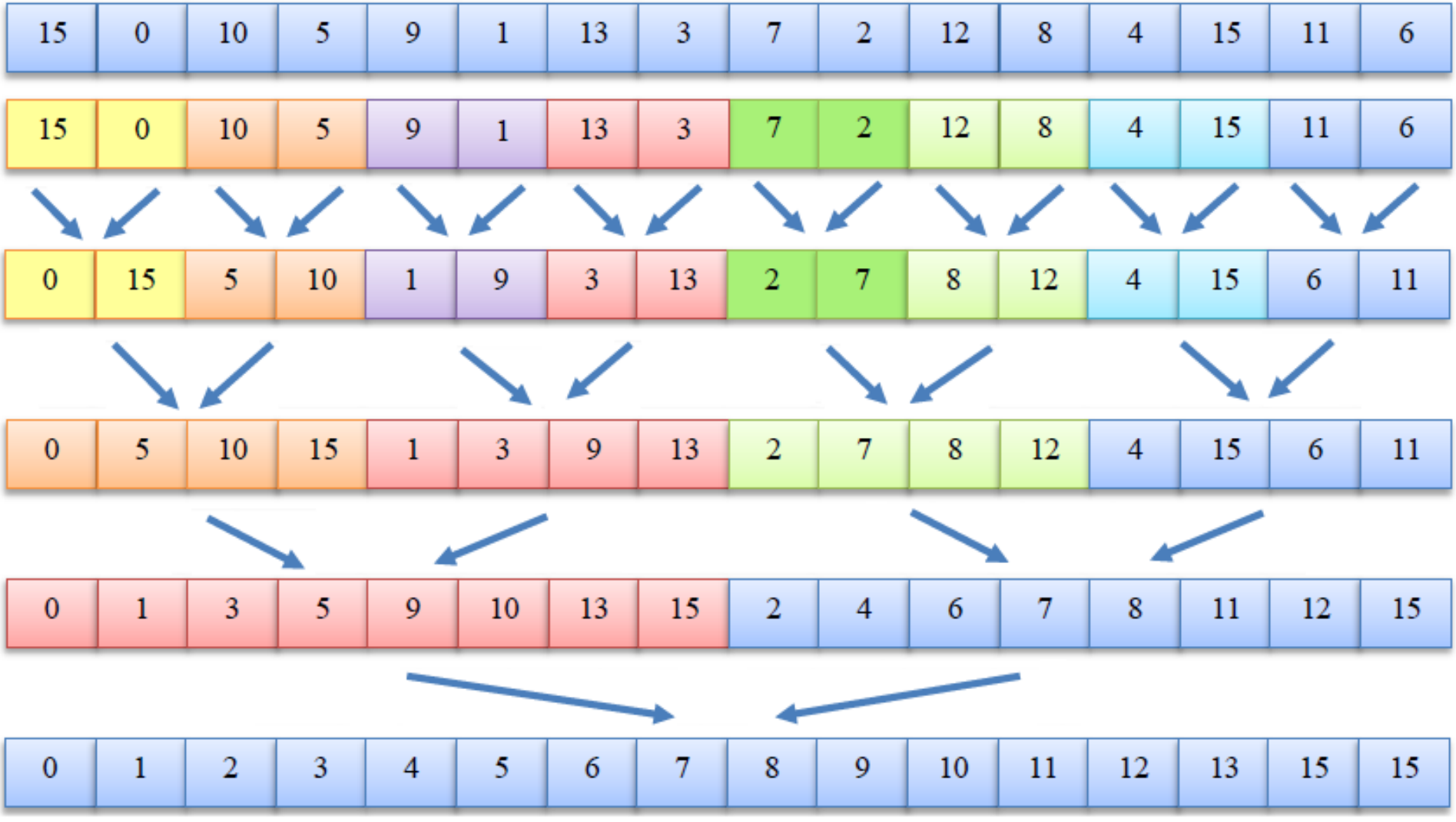
O mergesort recursivo faz um percurso em pós-ordem da árvore. No entanto, qualquer outro percurso que visite primeiro os filhos e depois o nó pode ser usado

Árvore de Intercalões



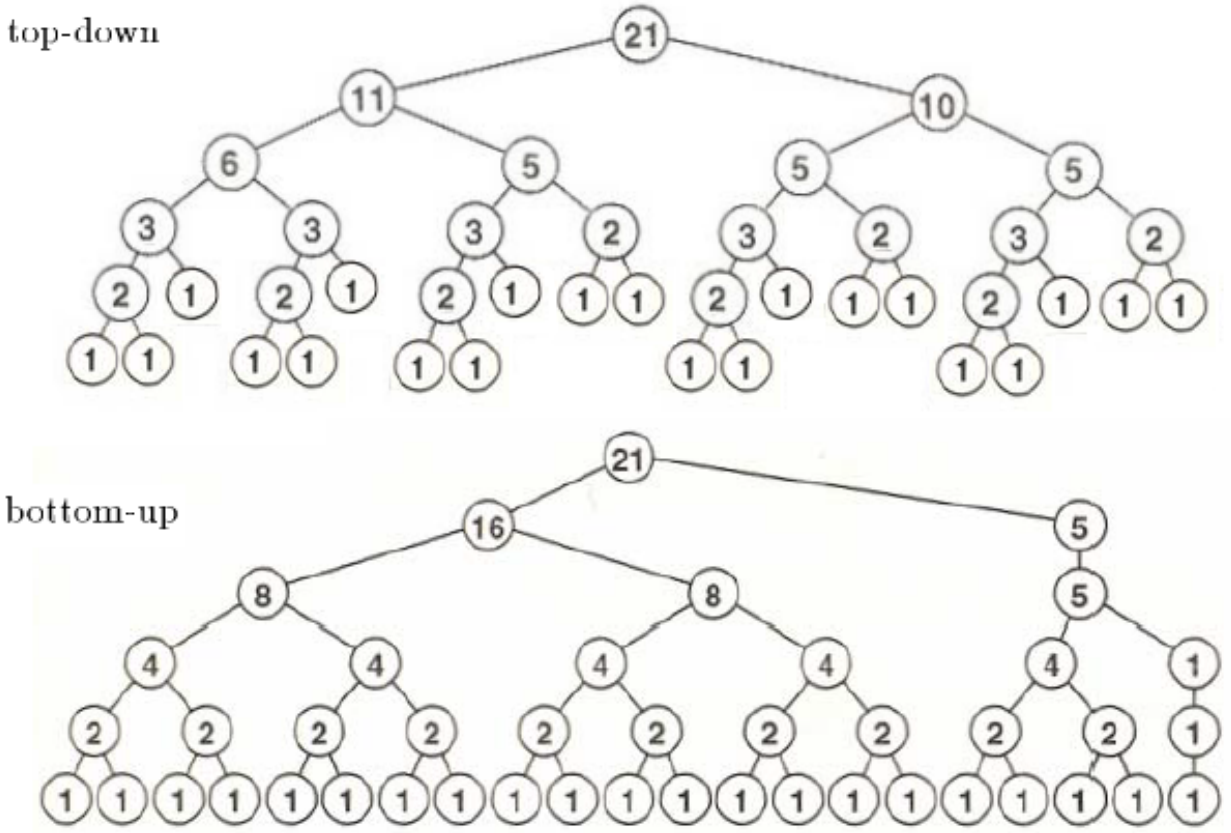
É possível fazer um percurso por níveis, de baixo para cima!

Bottom-Up Mergesort



Top-Down vs Bottom-Up Mergesort

- Se n não é uma potência de 2, então o conjunto de intercalações das duas variantes é diferente



Robert Sedgewick, Algorithms, 4th Edition, Addison-Wesley, 2011

Top-Down vs Bottom-Up Mergesort

- Se n não é uma potência de 2, então o conjunto de intercalações das duas variantes é diferente
- A versão recursiva realiza menos comparações (em todos os casos) é mais rápida (Sedgewick)

N	MergeSort	MergeSort+InsertionSort	Bottom-Up MergeSort
100000	53	43	59
200000	111	92	127
400000	237	198	267
800000	524	426	568

Robert Sedgewick, Algorithms, 4th Edition, Addison-Wesley, 2011

Top-Down vs Bottom-Up Mergesort

- Se n não é uma potência de 2, então o conjunto de intercalações das duas variantes é diferente
- A versão recursiva realiza menos comparações (em todos os casos) é mais rápida (Sedgewick)

N	MergeSort	MergeSort+InsertionSort	Bottom-Up MergeSort
100000	53	43	59
200000	111	92	127
400000	237	198	267
800000	524	426	568

- existe uma variante não recursiva que aproveita as sequencias ordenadas do vetor de entrada (*natural mergesort*)

Robert Sedgewick, Algorithms, 4th Edition, Addison-Wesley, 2011

Universidade Federal do ABC

CMCC Centro de Matemática, Computação e Cognição

Agenda

Referências Bibliográficas

- Algorithms, Robert Sedgewick and Kevin Wayne, 4th Edition, 2011, Slides http://algs4.cs.princeton.edu/lectures/
- Introduction to Algorithms, 3rd Edition. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, 2009
- The Art of Computer Programming 3rd Edition, Donald Knuth, 1997
- Projeto de Algoritmos, 2da Edição, Nivio Ziviani, 2007
- Estruturas de Dados e seus Algoritmos, 3ra edição, Jayme L. Szwarcfiter and Lilian Markezon, 2010
- Wikipedia: https://en.wikipedia.org/wiki/Sorting_algorithm

Exemplo de Intercalação *in-place*

```
1   left = first;  right = mid+1;
2
3   // One extra check:  can we SKIP the merge?
4   if ( x[mid].compareTo(x[right]) <= 0 )
5      return;
6
7   while (left <= mid && right <= last)
8   {  // Select from left:  no change, just advance left
9      if ( x[left].compareTo(x[right]) <= 0 )
10        left++;
11     // Select from right:  rotate [left..right] and correct
12     else
13     {  tmp = x[right];      // Will move to [left]
14        System.arraycopy(x, left, x, left+1, right-left);
15        x[left] = tmp;
16        // EVERYTHING has moved up by one
17        left++;  mid++;  right++;
18     }
19  }
20  // Whatever remains in [right..last] is in place
```

Tomado de http://penguin.ewu.edu/cscd300/Topic/AdvSorting/MergeSorts/InPlace.html. No caso pior, esta variante tem complexidade $O(n^2)$. Para variantes mais eficientes (porém mais complexas) ver J. Katajainen, T. Pasanen, and J. Teuhola, Practical in-place mergesort, Nordic Journal of Computing 3, 27-40, 1996