

Algoritmos de Ordenação

Merge Sort
("intercala")

(Fonte: Material adaptado dos Slides do prof. Monael.)

Alguns Algoritmos de Ordenação

- Exemplos de algoritmos $O(n \log n)$:

- — Merge Sort
 - "Intercalação"
- Quick Sort
- Heap Sort

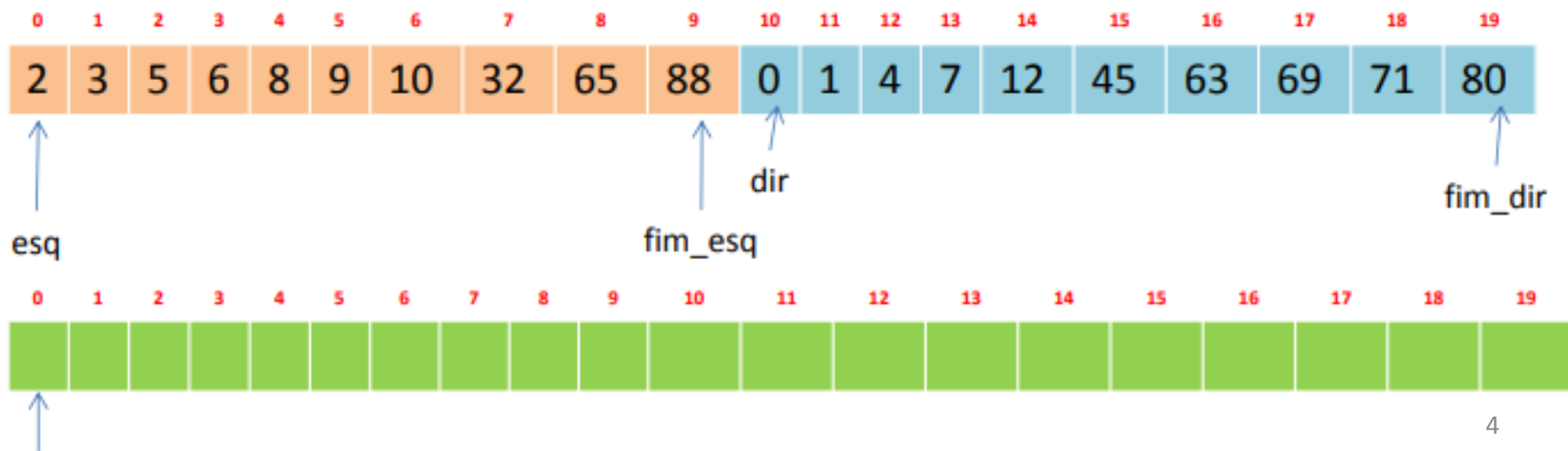
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	3	5	6	8	9	10	32	65	88	0	1	4	7	12	45	63	69	71	80

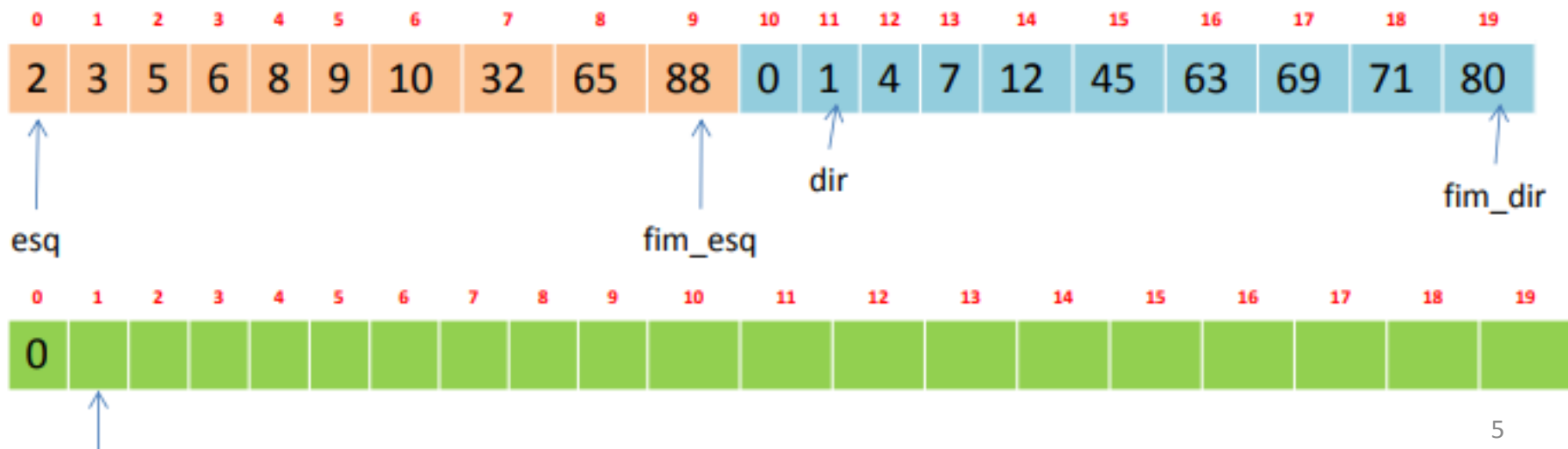
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



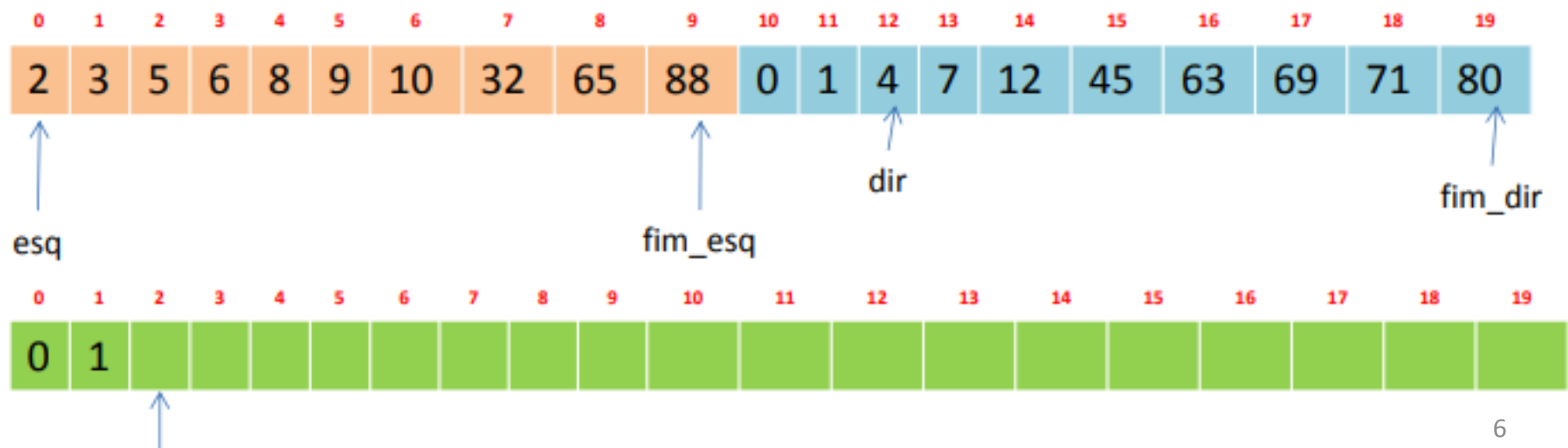
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



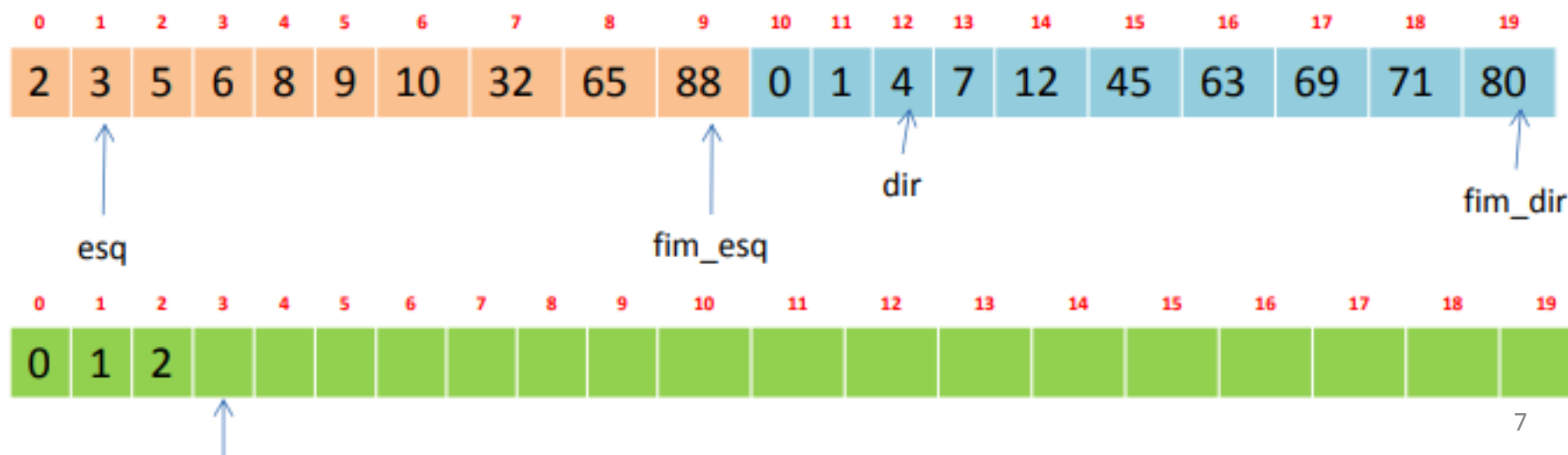
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



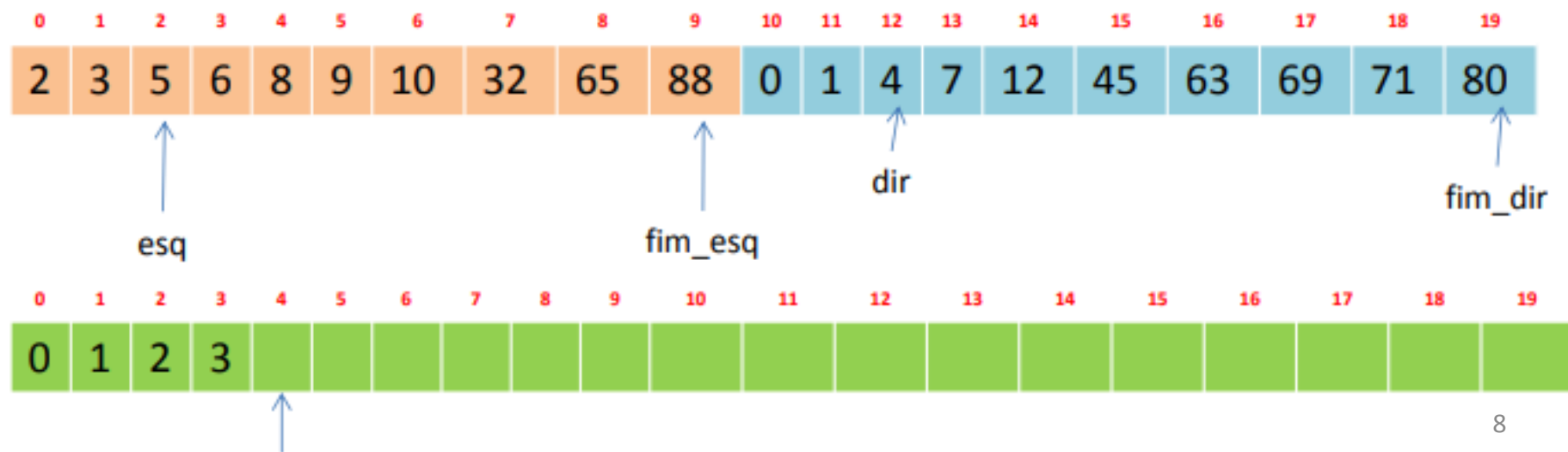
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



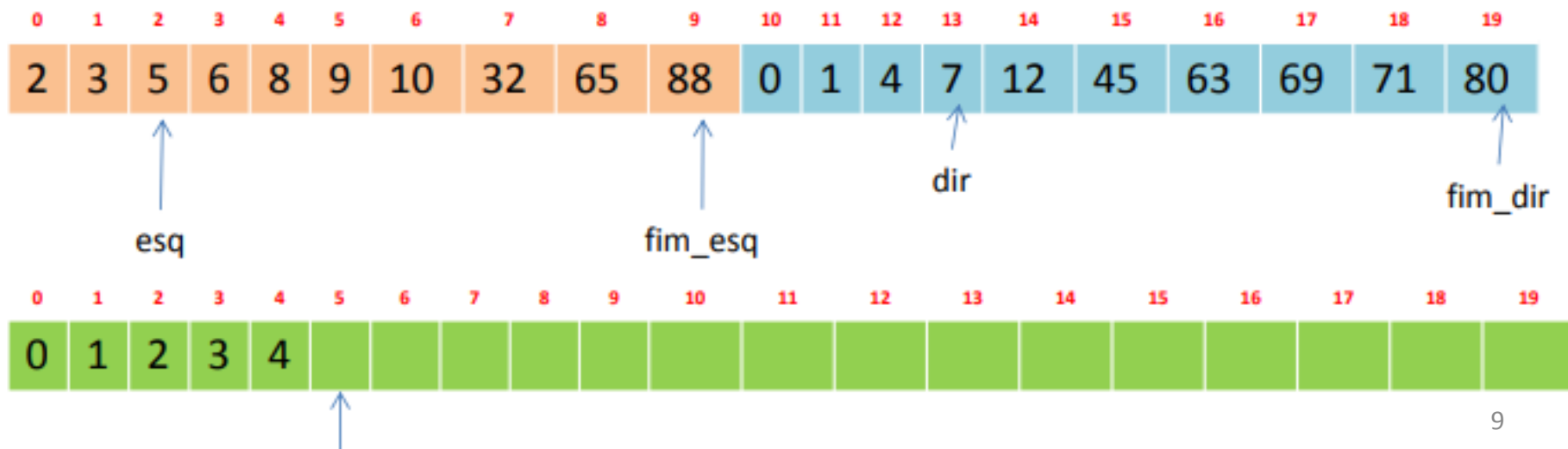
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



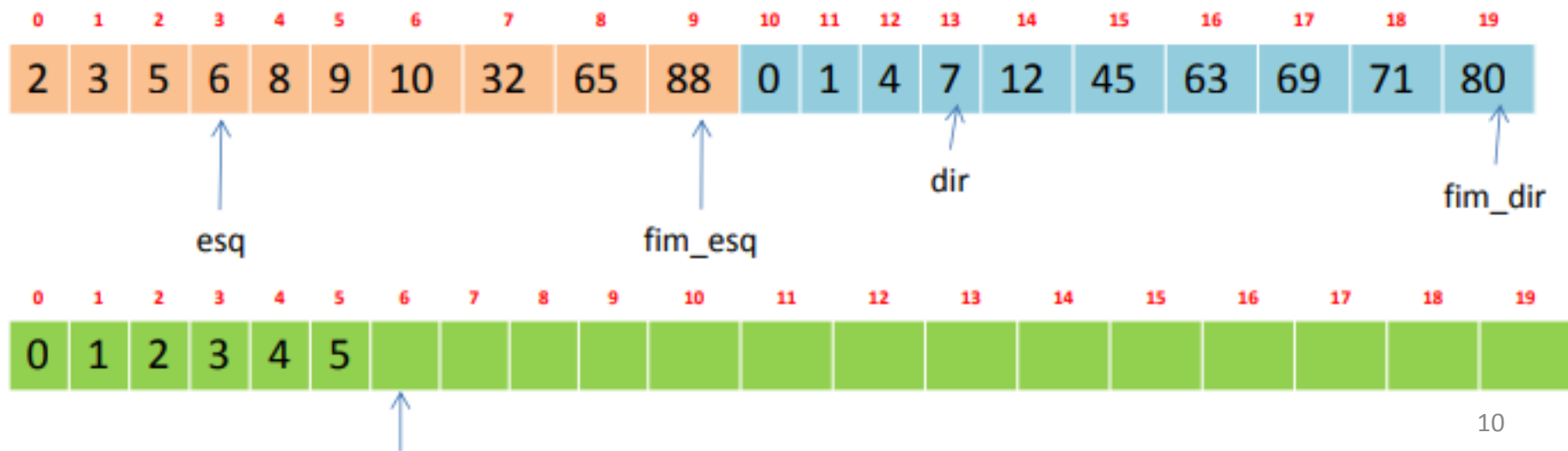
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



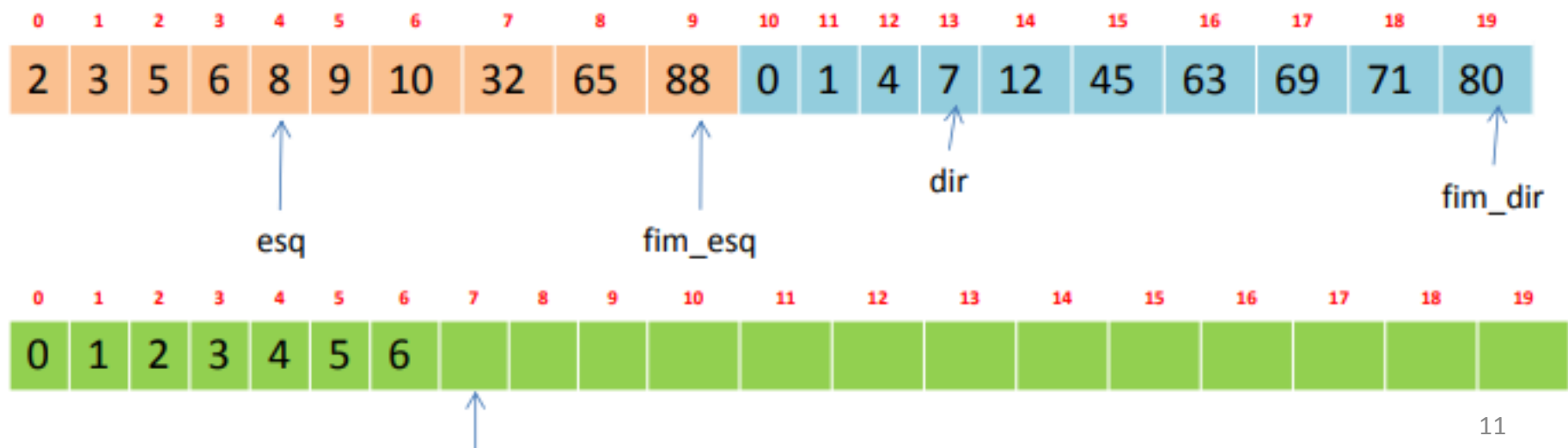
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



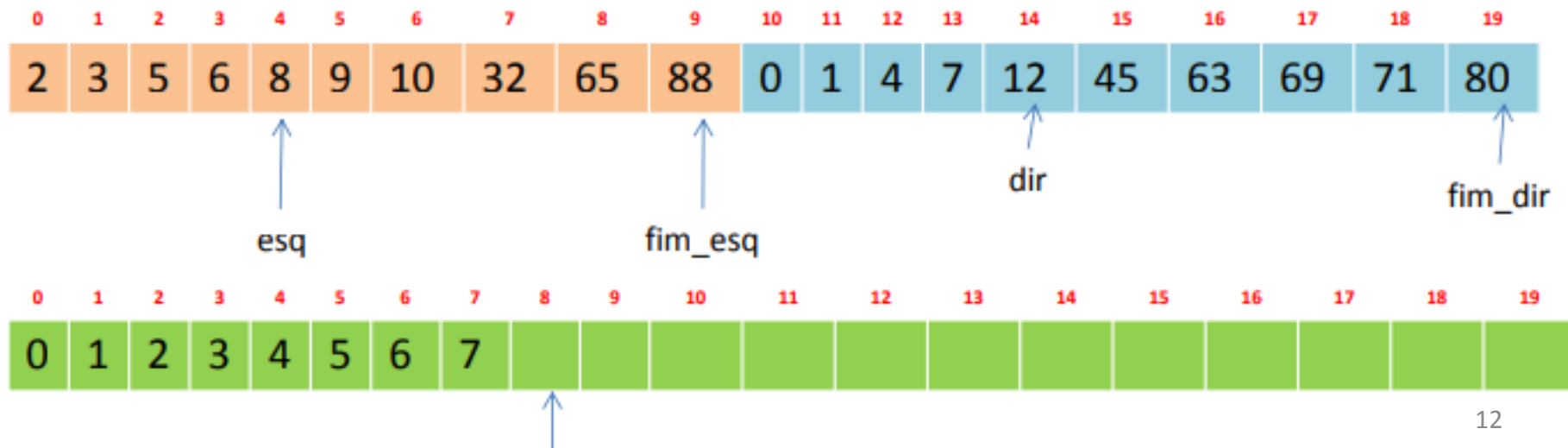
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



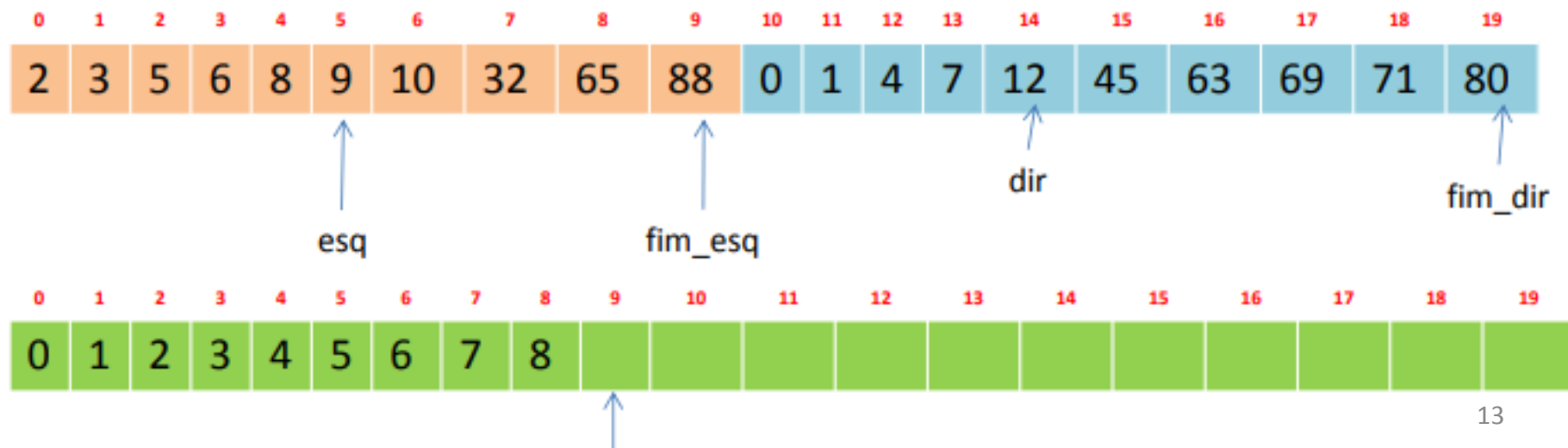
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



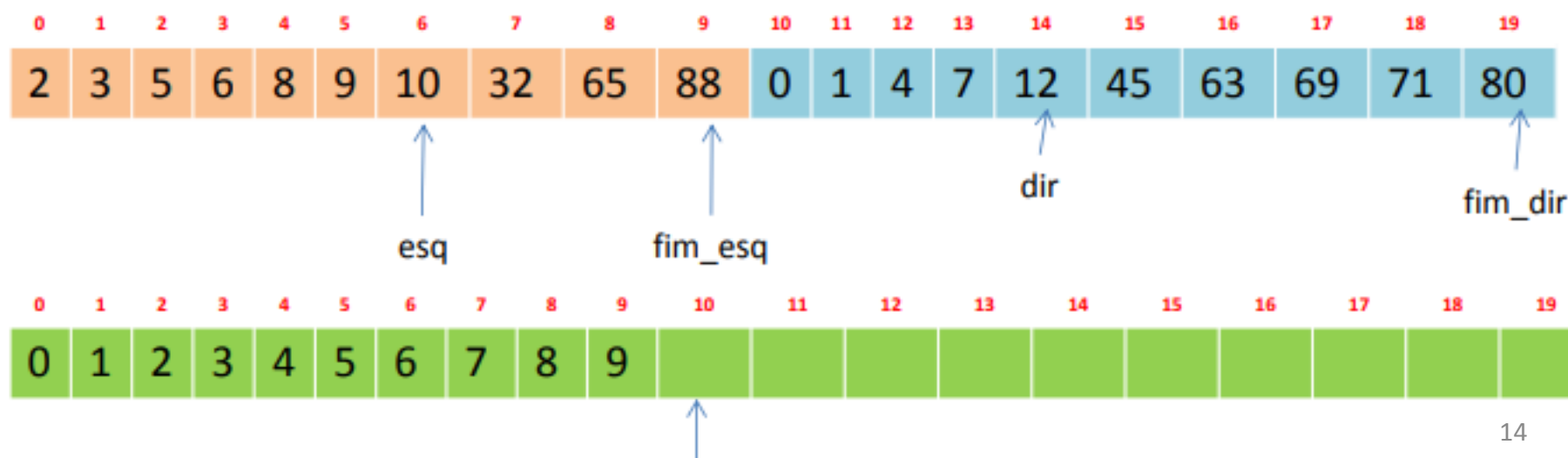
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



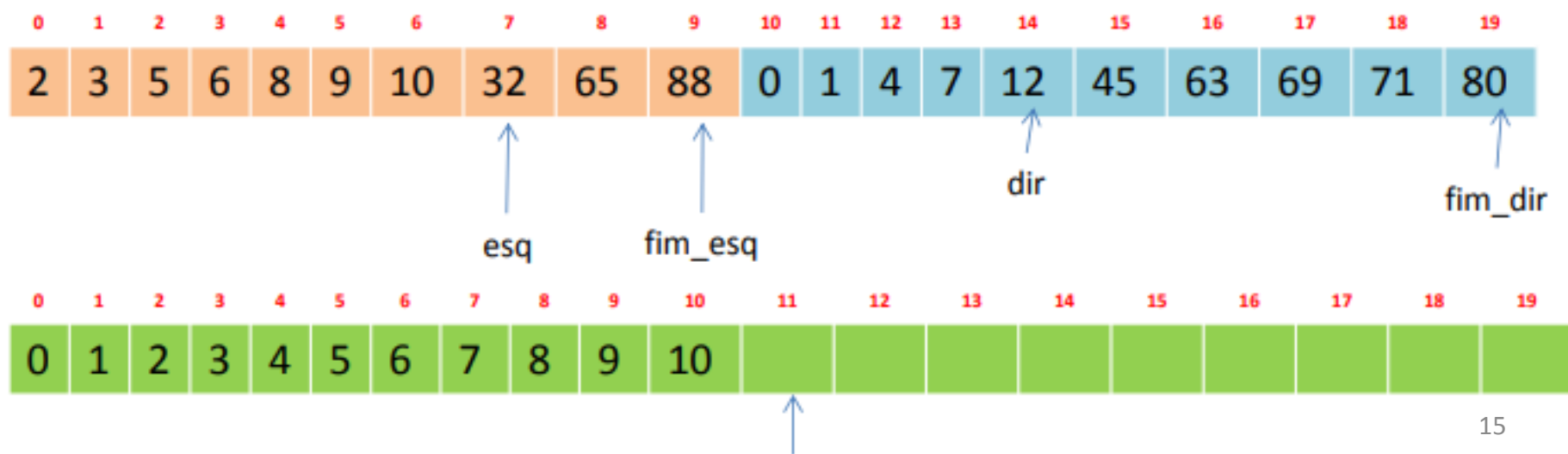
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



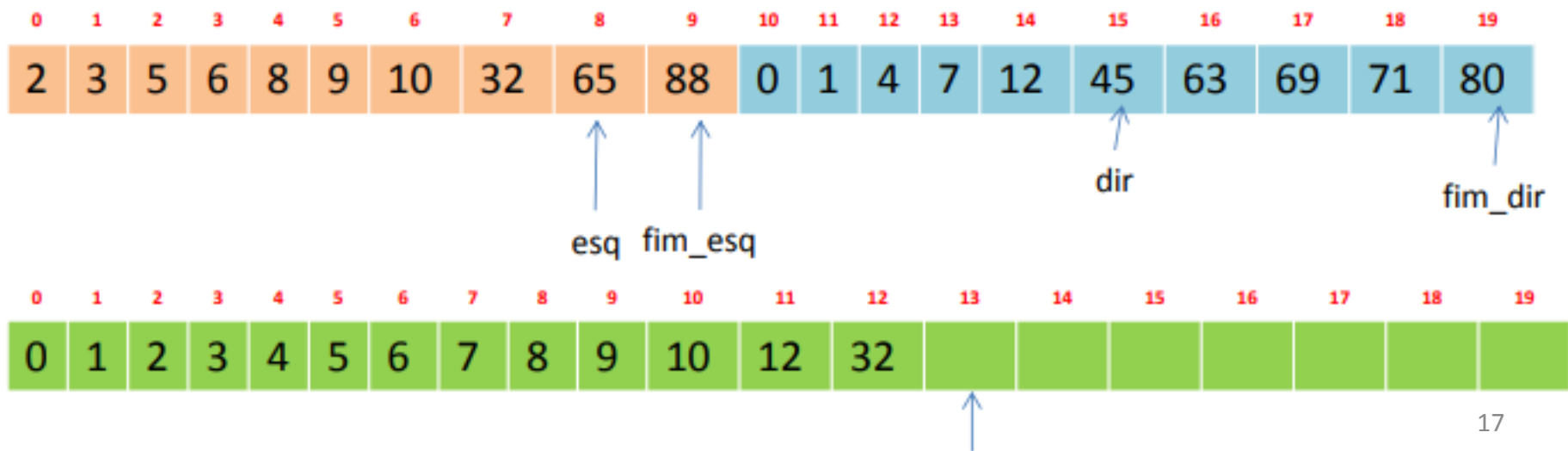
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



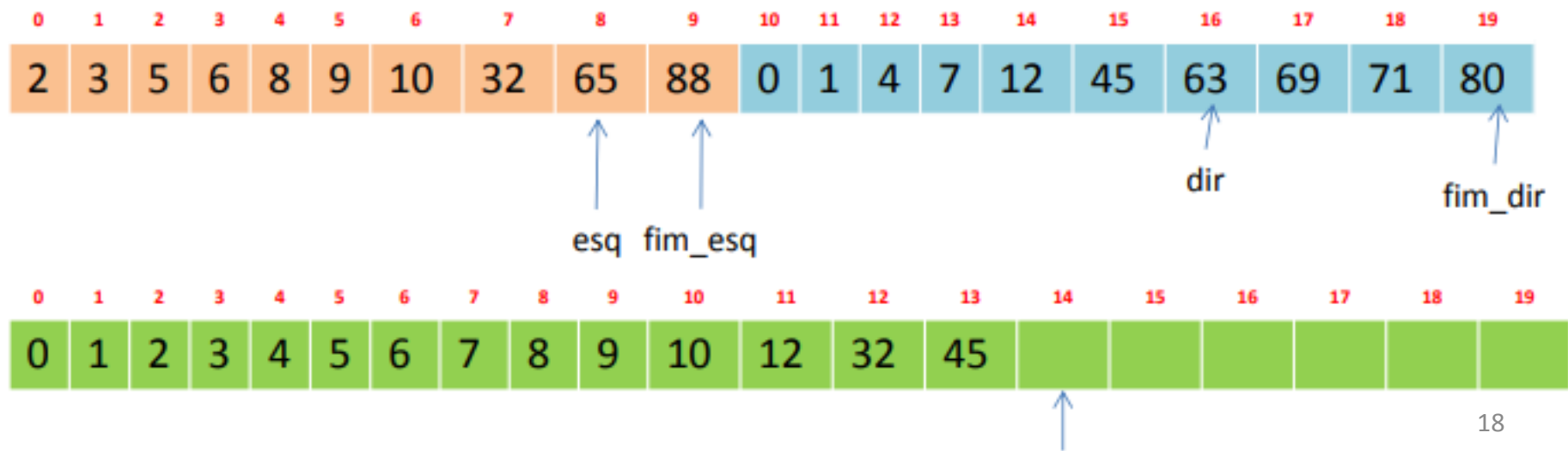
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



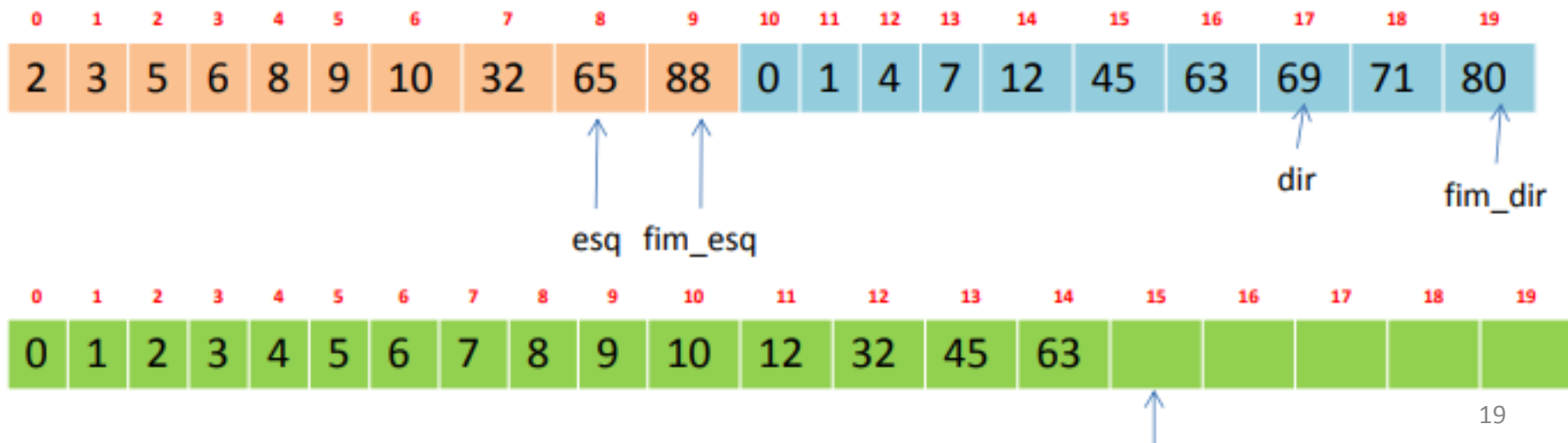
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



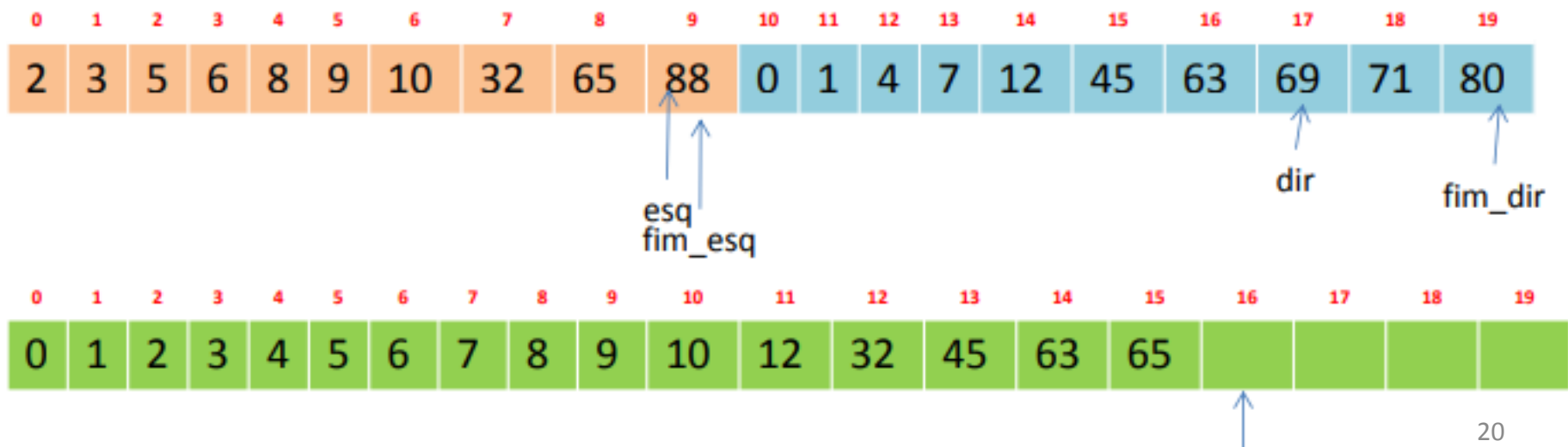
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



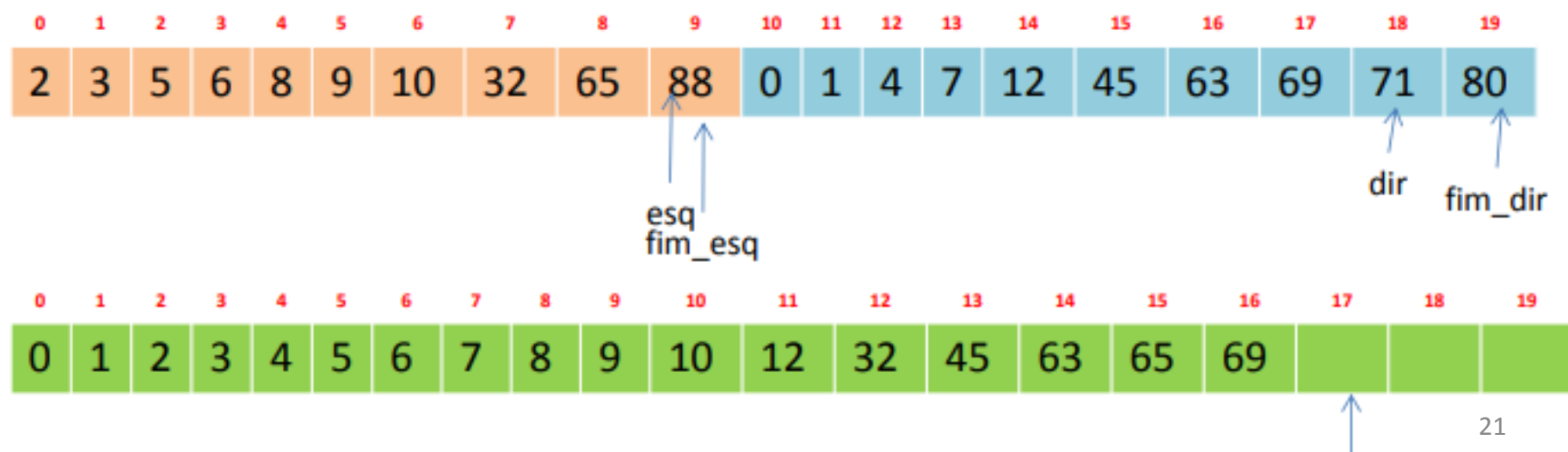
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



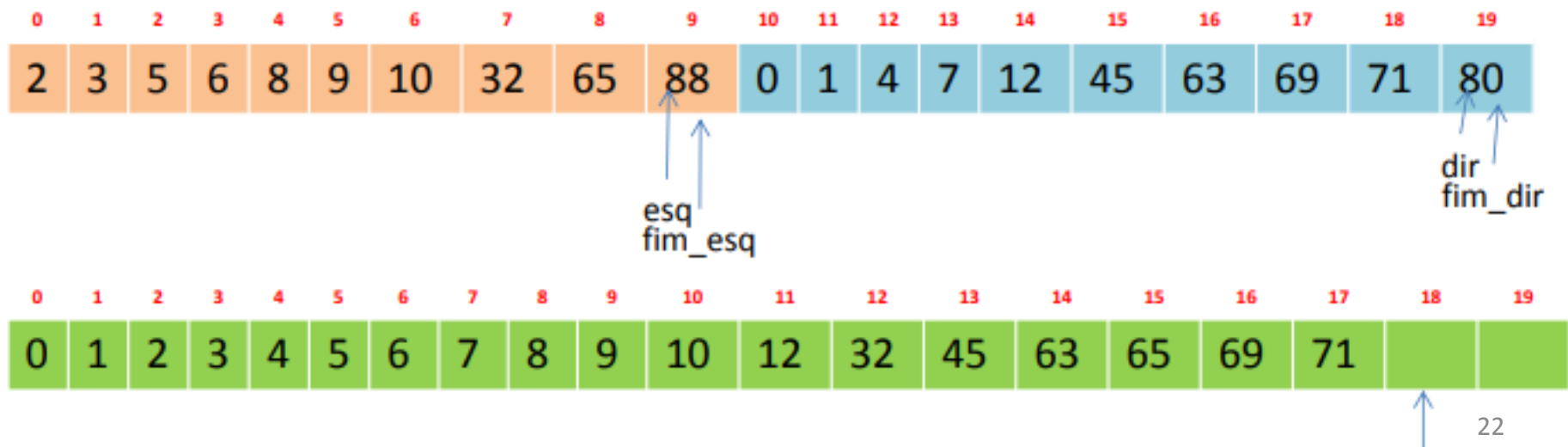
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



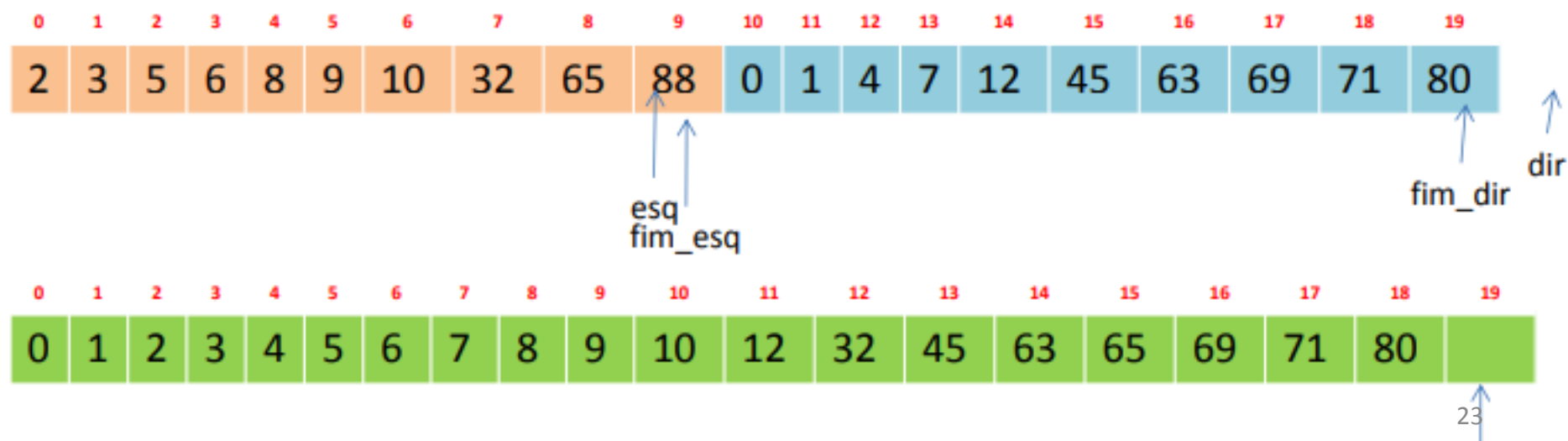
Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:



Intercalação

- Dado um vetor de inteiro de tamanho n , bipartidos em dois subvetores, ambos ordenados. Fazer um procedimento que retorne o vetor intercalado também ordenado.
- Exemplo:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	3	5	6	8	9	10	32	65	88	0	1	4	7	12	45	63	69	71	80

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	2	3	4	5	6	7	8	9	10	12	32	45	63	65	69	71	80	88

Intercala

```
01. void intercala(int *v, int e, int m, int d)
    {
02.     int *temp, i, fim_esq = m-1;
03.     temp = (int*) malloc(d*sizeof(int));
04.     for(i=0; e<=fim_esq && m<d; i++)
05.         if(v[e] < v[m])
06.             {
07.                 temp[i] = v[e];
08.                 e++;
09.             }
10.         else
11.             {
12.                 temp[i] = v[m];
13.                 m++;
14.             }
15.     for(; e<=fim_esq; e++, i++)
16.         temp[i] = v[e];
17.     for(; m<d; m++, i++)
18.         temp[i] = v[m];
19.     for(i=0; i<d; i++)
20.         v[i] = temp[i];
21.     free(temp);
    }
```

Intercala

```
01. void intercala(int *v, int e, int m, int d)
    {
02.     int *temp, i, fim_esq = m-1;
03.     temp = (int*) malloc(d*sizeof(int));
04.     for(i=0; e<=fim_esq && m<d; i++)
05.     {
06.         if(v[e] < v[m])
07.         {
08.             temp[i] = v[e];
09.             e++;
10.         }
11.         else
12.         {
13.             temp[i] = v[m];
14.             m++;
15.         }
16.     }
17.     for(; e<=fim_esq; e++, i++)
18.         temp[i] = v[e];
19.     for(; m<d; m++, i++)
20.         temp[i] = v[m];
21.     for(i=0; i<d; i++)
22.         v[i] = temp[i];
23.     free(temp);
    }
```

```
// Consumo de Tempo
// Rascunho
```

```
// 02. O(1)
// 03. O(1)
// 04. O(n)
// 05. O(1) x O(n)

// 06. O(1) x O(n)
// 07. O(1) x O(n)

// 08. O(1) x O(n)

// 09. O(1) x O(n)
// 10. O(1) x O(n)

// 11. O(n)
// 12. O(1) x O(n)
// 13. O(n)
// 14. O(1) x O(n)
// 15. O(n)
// 16. O(1) x O(n)
// 17. O(1)
```

```
// Total: O(n)
```

Intercala

```
01. void intercala(int *v, int e, int m, int d)
    {
02.     int *temp, i, fim_esq = m-1;
03.     temp = (int*) malloc(d*sizeof(int));
04.     for(i=0; e<=fim_esq && m<d; i++)
05.         if(v[e] < v[m])
06.             {
07.                 temp[i] = v[e];
08.                 e++;
09.             }
10.         else
11.             {
12.                 temp[i] = v[m];
13.                 m++;
14.             }
15.     for(; e<=fim_esq; e++, i++)
16.         temp[i] = v[e];
17.     for(; m<d; m++, i++)
18.         temp[i] = v[m];
19.     for(i=0; i<d; i++)
20.         v[i] = temp[i];
21.     free(temp);
    }
```

```
// Consumo de Tempo
// Rascunho
```

```
// 02. O(1)
// 03. O(1)
// 04. O(n)
// 05. O(1) x O(n)

// 06. O(1) x O(n)
// 07. O(1) x O(n)

// 08. O(1) x O(n)

// 09. O(1) x O(n)
// 10. O(1) x O(n)

// 11. O(n)
// 12. O(1) x O(n)
// 13. O(n)
// 14. O(1) x O(n)
// 15. O(n)
// 16. O(1) x O(n)
// 17. O(1)

// Total: O(n)
```

Intercala

```
01. void intercala(int *v, int e, int m, int d)
    {
02.     int *temp, i, fim_esq = m-1;
03.     temp = (int*) malloc(d*sizeof(int));
04.     for(i=0; e<=fim_esq && m<d; i++)
05.         if(v[e] < v[m])
06.             {
07.                 temp[i] = v[e];
08.                 e++;
09.             }
10.         else
11.             {
12.                 temp[i] = v[m];
13.                 m++;
14.             }
15.     for(; e<=fim_esq; e++, i++)
16.         temp[i] = v[e];
17.     for(; m<d; m++, i++)
18.         temp[i] = v[m];
19.     for(i=0; i<d; i++)
20.         v[i] = temp[i];
21.     free(temp);
    }
```

```
// Consumo de Tempo
// Rascunho
```

```
// 02. O(1)
// 03. O(1)
// 04. O(n)
// 05. O(1) x O(n)
// 06. O(1) x O(n)
// 07. O(1) x O(n)
// 08. O(1) x O(n)
// 09. O(1) x O(n)
// 10. O(1) x O(n)
// 11. O(n)
// 12. O(1) x O(n)
// 13. O(n)
// 14. O(1) x O(n)
// 15. O(n)
// 16. O(1) x O(n)
// 17. O(1)
// Total: O(n)
```

Intercala

```
01. void intercala(int *v, int e, int m, int d)
    {
02.     int *temp, i, fim_esq = m-1;
03.     temp = (int*) malloc(d*sizeof(int));
04.     for(i=0; e<=fim_esq && m<d; i++)
05.         if(v[e] < v[m])
06.             {
07.                 temp[i] = v[e];
08.                 e++;
09.             }
10.         else
11.             {
12.                 temp[i] = v[m];
13.                 m++;
14.             }
15.     for(; e<=fim_esq; e++, i++)
16.         temp[i] = v[e];
17.     for(; m<d; m++, i++)
18.         temp[i] = v[m];
19.     for(i=0; i<d; i++)
20.         v[i] = temp[i];
21.     free(temp);
22. }
```

```
// Consumo de Tempo
// Rascunho
```

```
// 02.  $O(1)$ 
// 03.  $O(1)$ 
// 04.  $O(n)$ 
// 05.  $O(1) \times O(n)$ 
// 06.  $O(1) \times O(n)$ 
// 07.  $O(1) \times O(n)$ 
// 08.  $O(1) \times O(n)$ 
// 09.  $O(1) \times O(n)$ 
// 10.  $O(1) \times O(n)$ 
// 11.  $O(n)$ 
// 12.  $O(1) \times O(n)$ 
// 13.  $O(n)$ 
// 14.  $O(1) \times O(n)$ 
// 15.  $O(n)$ 
// 16.  $O(1) \times O(n)$ 
// 17.  $O(1)$ 
```

```
// Total:  $O(n)$ 
```

Merge Sort

- Segue a técnica de divisão e conquista. Intuitivamente opera da seguinte forma:
 - Divisão: Quebra a seqüência de n elementos a serem ordenados em duas subsequências de $n/2$ cada.
 - Conquista: Classifica-se ambas subsequências recursivamente, utilizando a ordenação por intercalação.
 - Combinação: Intercala-se ambas subsequências para formar a solução do problema original
- Prós e Contras:
 - Pró: Ligeiramente melhor que outros algoritmos de ordenação para entradas suficientemente grandes.
 - Contra: Requer no mínimo o dobro de memória em comparação com os demais algoritmos de ordenação.

Merge Sort

Idéia:



Não Ordenado



Ordenado



Em Consideração

Merge Sort

Idéia:



Merge Sort

Idéia:



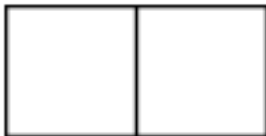
Merge Sort

Idéia:



Merge Sort

Idéia:



Merge Sort

Idéia:



Merge Sort

Idéia:



Não Ordenado



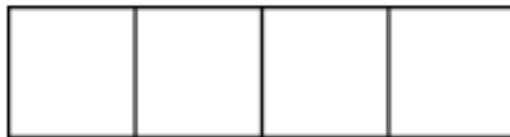
Ordenado



Em Consideração

Merge Sort

Idéia:



Merge Sort

Idéia:



Não Ordenado



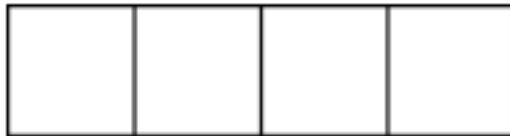
Ordenado



Em Consideração

Merge Sort

Idéia:



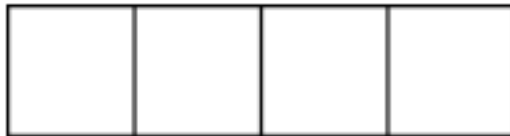
Merge Sort

Idéia:



Merge Sort

Idéia:



Merge Sort

Idéia:



Merge Sort

Idéia:



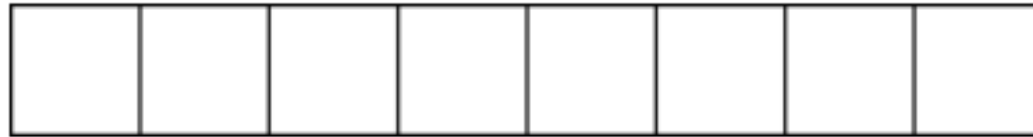
Merge Sort

Idéia:



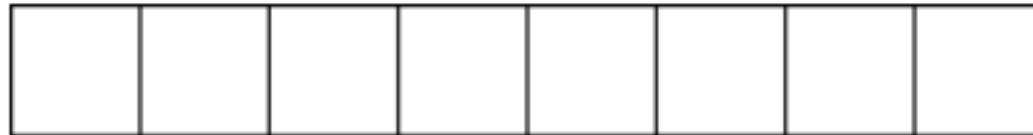
Merge Sort


Idéia:




Merge Sort

Idéia:



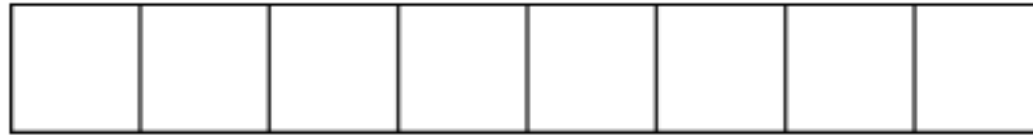
 Não Ordenado


 Ordenado


 Em Consideração


Merge Sort

Idéia:



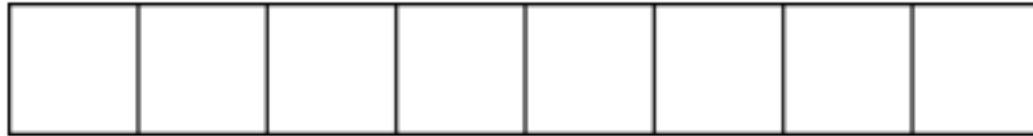
 Não Ordenado


 Ordenado

 Em Consideração


Merge Sort

Idéia:



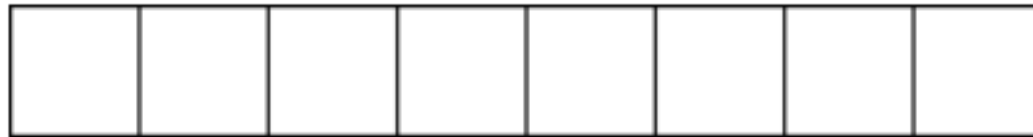
 Não Ordenado

 Ordenado

 Em Consideração

Merge Sort

Idéia:



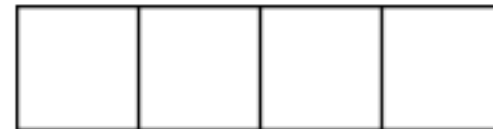
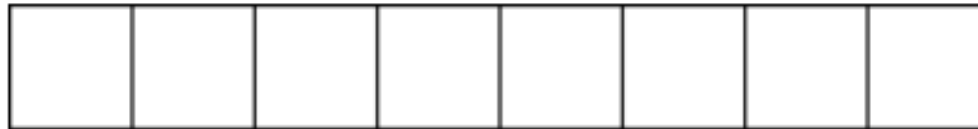
Merge Sort

Idéia:



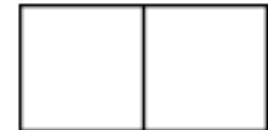
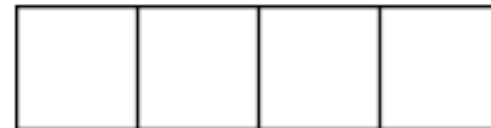
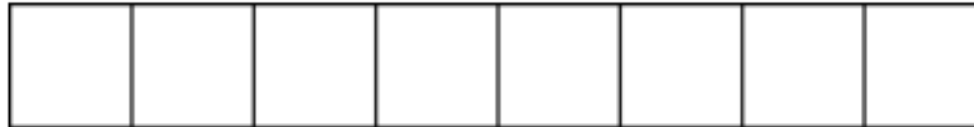
Merge Sort

Idéia:



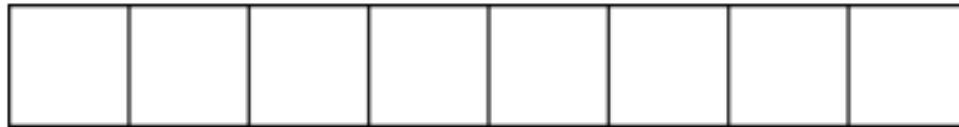
Merge Sort

Idéia:



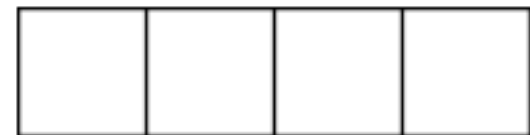
Merge Sort

Idéia:



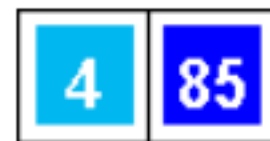
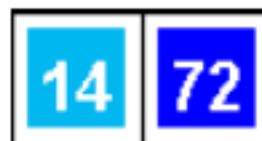
Merge Sort

Idéia:



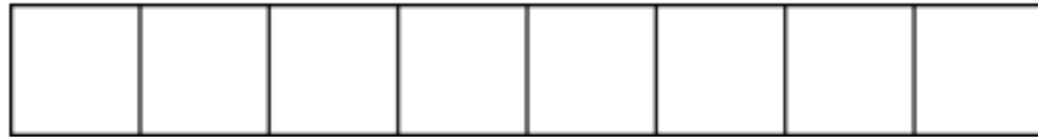
Merge Sort

Idéia:



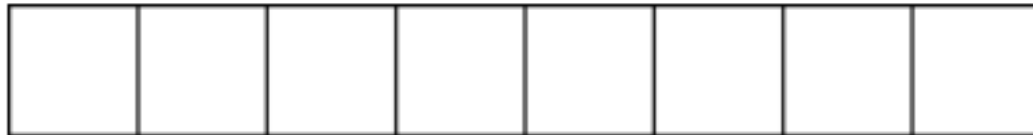
Merge Sort

Idéia:



Merge Sort

Idéia:



Não Ordenado



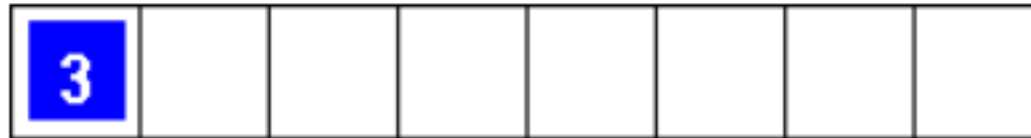
Ordenado



Em Consideração

Merge Sort

Idéia:



Não Ordenado



Ordenado



Em Consideração

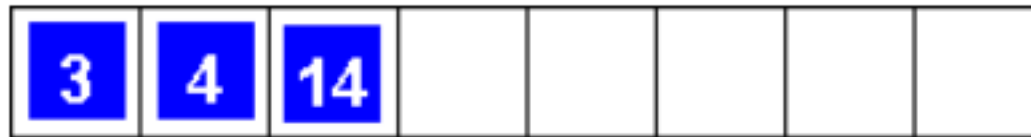
Merge Sort

Idéia:



Merge Sort

Idéia:



Não Ordenado



Ordenado



Em Consideração

Merge Sort

Idéia:



Merge Sort

Idéia:



Não Ordenado



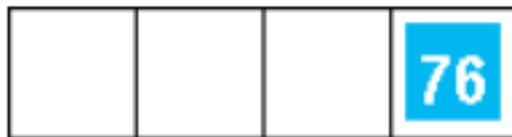
Ordenado



Em Consideração

Merge Sort

Idéia:



Não Ordenado



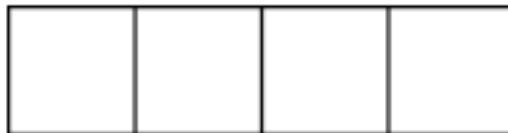
Ordenado



Em Consideração

Merge Sort

Idéia:



Não Ordenado



Ordenado



Em Consideração

Merge Sort

Idéia:



Não Ordenado



Ordenado



Em Consideração

Merge Sort

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
20	43	23	54	87	12	03	45	01	98	05	52	69	07	21	04

0	1	2	3	4	5	6	7
20	43	23	54	87	12	03	45

8	9	10	11	12	13	14	15
01	98	05	52	69	07	21	04

0	1	2	3	4	5	6	7
20	43	23	54	87	12	03	45

8	9	10	11	12	13	14	15
01	98	05	52	69	07	21	04

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
20	43	23	54	87	12	03	45	01	98	05	52	69	07	21	04

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
20	43	23	54	87	12	03	45	01	98	05	52	69	07	21	04

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
20	43	23	54	12	87	03	45	01	98	05	52	07	69	04	21

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
20	23	43	54	03	12	45	87	01	05	52	98	04	07	21	69

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
03	12	20	23	43	45	54	87	01	04	05	07	21	52	69	98

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
01	03	04	05	07	12	20	21	23	43	45	52	54	69	87	98

Intercala

- O algoritmo de intercalação do Merge Sort usa como vetor subvetores do vetor original. Por este motivo, ele é ligeiramente diferente da implementação usando um vetor bipartido.
 - Pois agora o vetor será n-partido, então deve-se preocupar com o tamanho do subvetor.
 - Deve-se ter cuidado ao transferir o resultado do vetor temporário para o original, pois os índices do original devem ser mantidos.

Merge Sort

```
void mergeSort(int *v, int e, int d)
{
01.     int meio;
02.     if(e < d)
03.     {
04.         meio = (d+e)/2;
05.         mergeSort(v, e, meio);
06.         mergeSort(v, meio+1, d);
07.         intercala(v, e, meio+1, d);
08.     }
09. }
```

Merge Sort

```
void mergeSort(int *v, int e, int d)    // Consumo de Tempo
{                                        // Rascunho
01.    int meio;                        // 01. O(1)
02.    if(e < d)                        // 02. O(1)
    {
03.        meio = (d+e)/2;              // 03. O(1)
04.        mergeSort(v, e, meio);       // 04. T("n/2")
05.        mergeSort(v, meio+1, d);     // 05. T("n/2")
06.        intercala(v, e, meio+1, d);  // 06. O(n)
    }
}
```

```
// Total: T(n) = 2 x T(n/2) + O(n) + 3 x O(1)
              = 2 x T(n/2) + O(n)
```

Análise de Algoritmo

```
void mergeSort(int *v, int e, int d)
{
01.     int meio;
02.     if(e < d)
03.     {
04.         meio = (d+e)/2;
05.         mergeSort(v, e, meio); → T(n/2)
06.         mergeSort(v, meio+1, d); → T(n/2)
07.         intercala(v, e, meio+1, d); → O(n)
08.     }
09. }
```

$$T(n) = \begin{cases} 1 & , \text{ para } n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + n & , \text{ para } n > 1 \end{cases}$$

Recorrência

- Exemplo: considere $n = 1, 2, 4, 8, 16, 32, 64, \dots$
 - $T(1) = 1$
 - $T(n) = 2 * T(n/2) + n$

$T[2]$	$= 2 * T[1] + 2$	$= 2 * 1 + 2$	$= 4$
$T[4]$	$= 2 * T[2] + 4$	$= 2 * 4 + 4$	$= 12$
$T[8]$	$= 2 * T[4] + 8$	$= 2 * 12 + 8$	$= 32$
$T[16]$	$= 2 * T[8] + 16$	$= 2 * 32 + 16$	$= 80$
$T[32]$	$= 2 * T[16] + 32$	$= 2 * 80 + 32$	$= 192$
$T[64]$	$= 2 * T[32] + 64$	$= 2 * 192 + 64$	$= 448$
$T[128]$	$= 2 * T[64] + 128$	$= 2 * 448 + 128$	$= 1024$
$T[256]$	$= 2 * T[128] + 256$	$= 2 * 1024 + 256$	$= 2304$
$T[512]$	$= 2 * T[256] + 512$	$= 2 * 2304 + 512$	$= 5120$
$T[1024]$	$= 2 * T[512] + 1024$	$= 2 * 5120 + 1024$	$= 11264$

Comportamento assintótico...

- ... para **n** "grande".

n	$T(n) = 2 * T(n/2) + n$	$n * \log(n)$
2	4	2
4	12	8
8	32	24
16	80	64
32	192	160
64	448	384
128	1024	896
256	2304	2048
512	5120	4608
1024	11264	10240
...

Comportamento assintótico...

- ... para **n** "grande".

n	$T(n) = 2 * T(n/2) + n$	$n * \log(n)$
1024	11264	10240
2048	24576	22528
4096	53248	49152
8192	114688	106496
16384	245760	229376
32768	524288	491520
65536	1114112	1048576
131072	2359296	2228224
262144	4980736	4718592
524288	10485760	9961472
1048576	22020096	20971520

Comportamento assintótico...

- ... para **n** "grande".

n	$T(n) = 2 * T(n/2) + n$	$n * \log(n)$
1048576	22020096	20971520
2097152	46137344	44040192
4194304	96468992	92274688
8388608	201326592	192937984
16777216	419430400	402653184
33554432	872415232	838860800
67108864	1811939328	1744830464
...
274877906944	10720238370816	10445360463872
549755813888	21990232555520	21440476741632
1099511627776	45079976738816	43980465111040

Comportamento assintótico...

- ... para **n** "grande".

n	$T(n) = 2 * T(n/2) + n$	$n * \log(n)$
281474976710656	13792273858822144	13510798882111488
562949953421312	28147497671065600	27584547717644288
1125899906842624	57420895248973824	56294995342131200
...
2251799813685248	117093590311632896	114841790497947648
4503599627370496	238690780250636288	234187180623265792
9007199254740992	486388759756013568	477381560501272576
18014398509481984	990791918021509120	972777519512027136
36028797018963968	2017612633061982208	1981583836043018240
72057594037927936	4107282860161892352	4035225266123964416
144115188075855872	8358680908399640576	8214565720323784704

Recorrência

- Exemplo: considere $n = 1, 2, 4, 8, 16, 32, 64, \dots$
 - $T(1) = 1$
 - $T(n) = 2 * T(n/2) + n$
- Fórmula "fechada":
 - $T(n) = n * \log(n) + n = O(n * \log n)$

(Verifique esta igualdade!)

Análise de Algoritmo

Portanto, o Merge Sort ordena um arranjo com n elementos, consumindo tempo proporcional a:

$$n \cdot \log_2 n + n$$

Deste modo, sua complexidade é:

$$O(n \cdot \log_2 n)$$

E de acordo com a análise de Lower Bound do Problema da Ordenação, o Merge Sort é um Algoritmo Assintoticamente Ótimo.