Usando tudo

Problema *Ordenação*

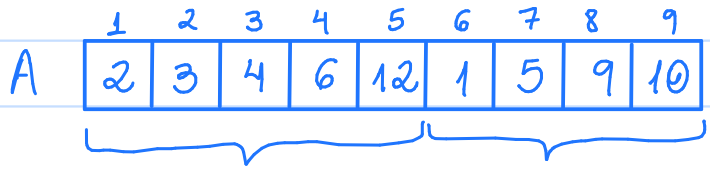
Entrada: $\langle A, n\rangle$, onde $A = (a_1, a_2, \ldots, a_n)$ é um vetor com n números.
Saída: Permutação $(a'_1, a'_2, \ldots, a'_n)$ dos números de A de modo que $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

→ Vimos que o Insertion Sort resolve esse problema em tempo $O(n^2)$.

→ Será que dá para fazer melhor?

Um algoritmo recursivo para Ordenação

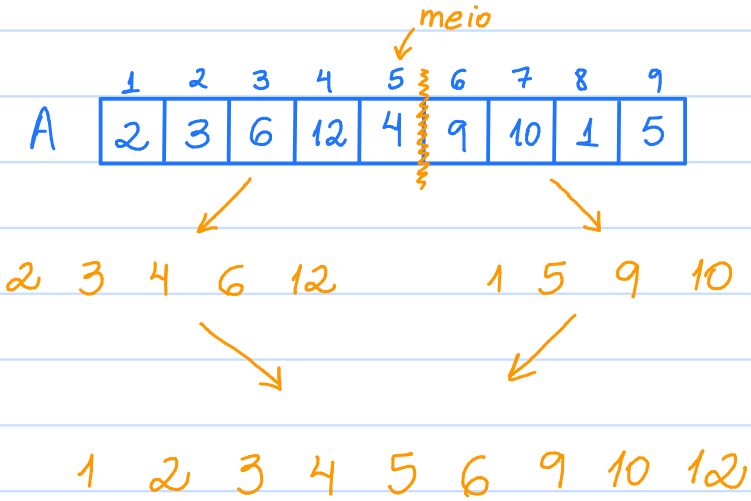
→ Se partes do vetor estão ordenadas, fica mais fácil ordenar o vetor todo?



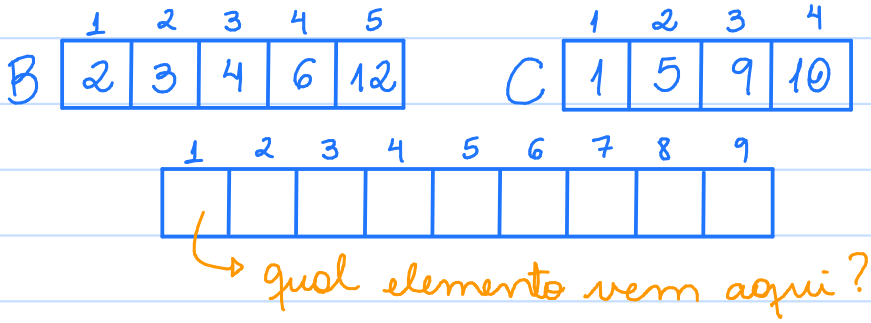
Ordenação por intercalação

```
MERGESORT(A, ini, fim)
1 se inicio < fim então
2     meio = ⌊(ini + fim)/2⌋
3     MERGESORT(A, ini, meio)
4     MERGESORT(A, meio + 1, fim)
5     COMBINA(A, ini, meio, fim)
```

Ideia: ordene as duas metades do vetor e intercale os dois subvetores.



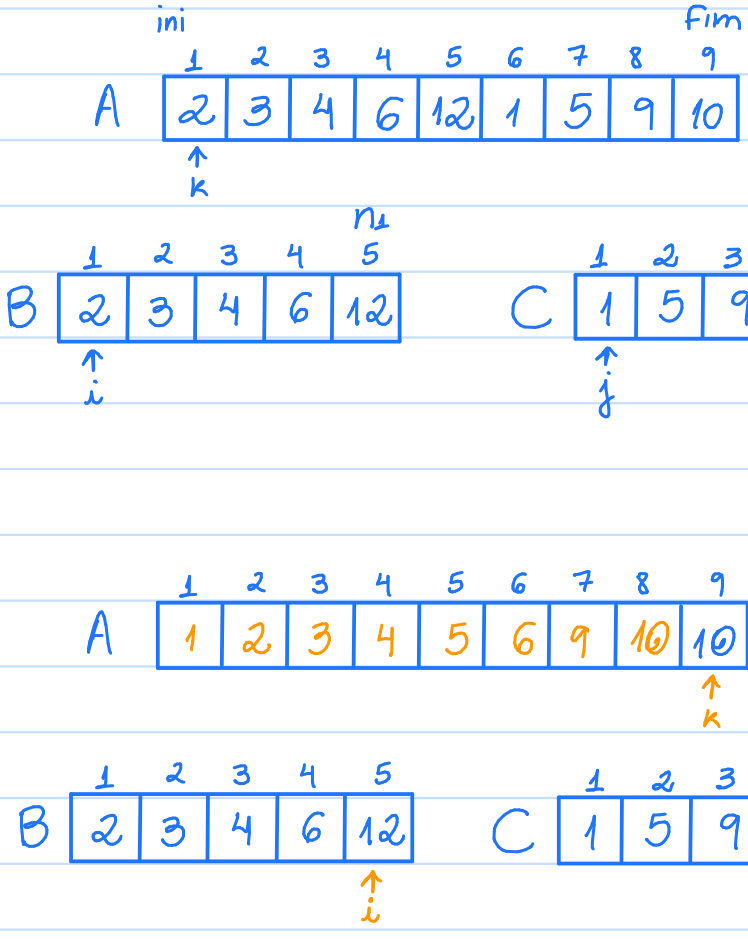
Combinando dois vetores ordenados em um



Combinando dois vetores ordenados em um

```
COMBINA(A, ini, meio, fim)
1  n1 = meio − ini + 1
2  n2 = fim − meio
3  Crie vetores auxiliares B[1..n1] e C[1..n2]
4  para i = 1 até n1, incrementando faça
5      B[i] = A[ini + i − 1]
6  para j = 1 até n2, incrementando faça
7      C[j] = A[meio + j]
8  i = 1
9  j = 1
10 k = ini
11 enquanto i ≤ n1 e j ≤ n2 faça
12     se B[i] ≤ C[j] então
13         A[k] = B[i]
14         i = i + 1
15     senão
16         A[k] = C[j]
17         j = j + 1
18     k = k + 1
19 enquanto i ≤ n1 faça
20     A[k] = B[i]
21     i = i + 1
22     k = k + 1
23 enquanto j ≤ n2 faça
24     A[k] = C[j]
25     j = j + 1
26     k = k + 1
```

→ considera A[ini .. meio] e A[meio+1 .. fim] ordenados



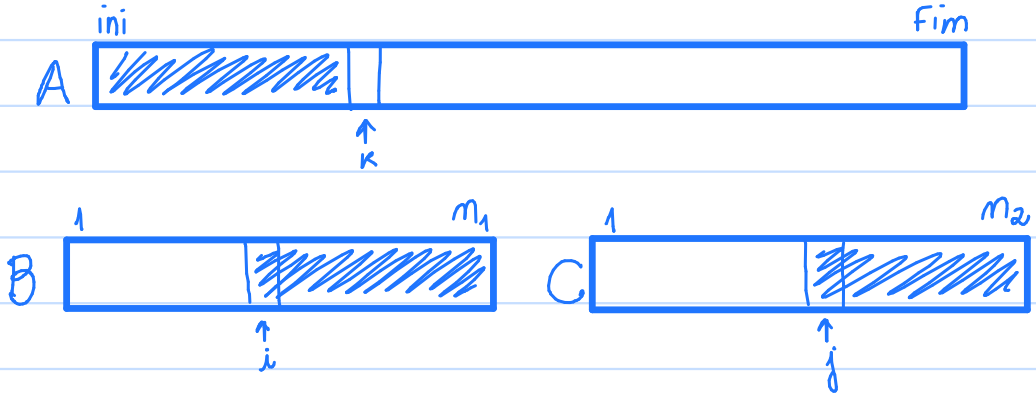
Combinando dois vetores ordenados em um: corretude

→ Queremos provar que Combina(A, ini, meio, fim) ordena A[ini..fim] se A[ini..meio] e A[meio+1..fim] estiverem ordenados, para qualquer tamanho de vetor (fim - ini + 1).

→ Pelo funcionamento do algoritmo, uma vez que um elemento de B ou C é copiado para A, ele não mudará de lugar.

→ Então temos que garantir que $A[ini..k-1] \leq A[k] \leq A[k+1..fim]$.

→ Vamos fazer isso mostrando que
P(t) = "Antes da t-ésima iteração começar, vale que $k = ini + t - 1$, o vetor $A[ini..k-1]$ está ordenado e os elementos de $B[i..n_1]$ e $C[j..n_2]$ são maiores ou iguais aos elementos de $A[ini..k-1]$."
é uma invariante do primeiro laço enquanto, por indução no nº t de vezes que o teste da condição de parada é executado.

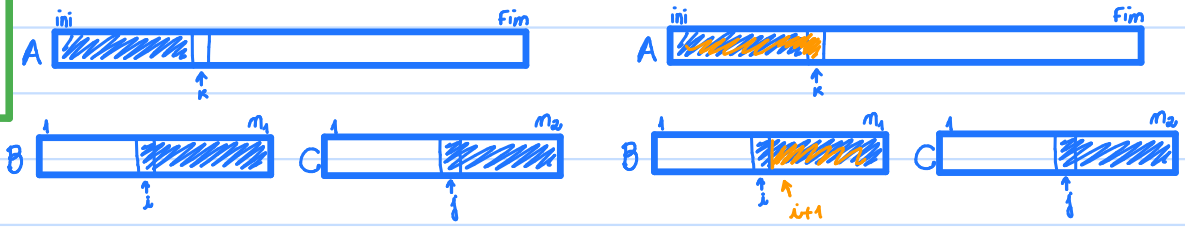


Quando t=1, o teste executou uma vez. Temos $k = ini$, $i = 1$ e $j = 1$, de forma que P(1) claramente vale.

Combinando dois vetores ordenados em um: corretude

```
8  i = 1
9  j = 1
10 k = ini
11 enquanto i ≤ n1 e j ≤ n2 faça
12     se B[i] ≤ C[j] então
13         A[k] = B[i]
14         i = i + 1
15     senão
16         A[k] = C[j]
17         j = j + 1
18     k = k + 1
```

P(t) = "Antes da t-ésima iteração começar, vale que k = ini + t - 1, o vetor A[ini .. k-1] está ordenado e os elementos de B[i .. n1] e C[j .. n2] são maiores ou iguais aos elementos de A[ini .. k-1]."



Seja $t \geq 1$ e suponha que P(t) vale. Então $k = ini + t - 1$.
Se $B[i] \leq C[j]$, copiamos B[i] para A[ini + t - 1]. Como P(t) vale, B[i] é maior do que A[ini .. ini + t - 2], então acabamos de verificar que A[ini .. ini + t - 1] está ordenado. Como B e C estão em ordem e $B[i] \leq C[j]$, verificamos que B[i+1 .. n1] e C[j .. n2] são maiores ou iguais a A[ini .. ini + t - 1]. Como incrementamos apenas i e k, temos que P(t+1) vale nesse caso.
Se B[i] > C[j], a análise é similar e P(t+1) também vale.
Então P é invariante.

Suponha que o laço termina com $k = k_f$, $i = i_f$ e $j = j_f$. Então o teste executou k_f vezes e $P(k_f)$ nos dá informações ao fim do laço: A[ini .. k_f - 1] está ordenado e seus elementos são menores do que os elem. em B[i_f .. n1] e C[j_f .. n2].

Se o laço acabou porque $i_f = n_1 + 1$, então B[i_f .. n1] é vazio. O teste da linha 19 falha e o laço enquanto da linha 23 será executado, copiando C[j_f .. n2] para A a partir da posição k_f. Como esses elementos são maiores do que A[ini .. k_f - 1], ao fim teremos A[ini .. fim] em ordem.

```
19 enquanto i ≤ n1 faça
20     A[k] = B[i]
21     i = i + 1
22     k = k + 1
23 enquanto j ≤ n2 faça
24     A[k] = C[j]
25     j = j + 1
26     k = k + 1
```

Se o laço acabou porque $j_f = n_2 + 1$, a análise é similar.

Então o algoritmo funciona corretamente.

Mergesort : corretude

```
MERGESORT(A, inicio, fim)
1 se inicio < fim então
2     meio = ⌊(inicio + fim)/2⌋
3     MERGESORT(A, inicio, meio)
4     MERGESORT(A, meio + 1, fim)
5     COMBINA(A, inicio, meio, fim)
```

Queremos provar que MergeSort(A, inicio, fim) ordena A[inicio..fim] para qualquer vetor. Vamos provar por indução no tamanho $n = fim - inicio + 1$ do vetor.

Se $n = 0$, $0 = fim - inicio + 1 \Rightarrow inicio = fim + 1$, ou $inicio > fim$

Se $n = 1$, $1 = fim - inicio + 1 \Rightarrow inicio = fim$

Veja que quando $inicio \geq fim$, o algoritmo não faz nada.
Mas um vetor vazio ou com um elemento já está ordenado.
Então MergeSort funciona no caso base.

Mergesort : corretude

```
MERGESORT(A, inicio, fim)
1 se inicio < fim então
2     meio = ⌊(inicio + fim)/2⌋
3     MERGESORT(A, inicio, meio)
4     MERGESORT(A, meio + 1, fim)
5     COMBINA(A, inicio, meio, fim)
```

Agora seja $n > 1$ e suponha que MergeSort ordena vetores com tamanho k, para $0 \leq k < n$.

Note que $n = fim - inicio + 1 > 1 \Rightarrow fim > inicio$.

Nesse caso, o algoritmo calcula $meio = \left\lfloor \frac{inicio + fim}{2} \right\rfloor$ e faz duas chamadas recursivas. A primeira é sobre A[inicio..meio] e veja que

$$meio - inicio + 1 = \left\lfloor \frac{inicio + fim}{2} \right\rfloor - inicio + 1 \leq \frac{inicio + fim}{2} - inicio + 1 = \frac{fim - inicio + 2}{2} = \frac{n+1}{2} < n$$

quando $n > 1$. Então essa chamada, por hipótese, funciona
Como $fim - (meio + 1) + 1 < n$, a segunda chamada também funciona.
Já vimos que Combina funciona quando A[inicio..meio] e A[meio+1..fim] estão ordenados. Então a chamada atual ordena A[inicio..fim]. CQD

Mergesort : tempo de execução

```
MERGESORT(A, inicio, fim)
1  se inicio < fim então
2      meio = ⌊(inicio + fim)/2⌋
3      MERGESORT(A, inicio, meio)
4      MERGESORT(A, meio + 1, fim)
5      COMBINA(A, inicio, meio, fim)
```

$T(\star)$ = tempo que Mergesort leva para ordenar vetores de tamanho $\star$

$T(0) = T(1) = \Theta(1)$

$$T(n) = T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + \Theta(n)$$
$$= T(\tfrac{n}{2}) + T(\tfrac{n}{2}) + \Theta(n)$$
$$= 2T(\tfrac{n}{2}) + \Theta(n) \quad ?$$

Combina : tempo de execução

```
COMBINA(A, inicio, meio, fim)
1  n1 = meio − inicio + 1                    Θ(1)
2  n2 = fim − meio                           Θ(1)
3  Crie vetores auxiliares B[1..n1] e C[1..n2]
4  para i = 1 até n1, incrementando faça
5      B[i] = A[inicio + i − 1]              Θ(n1)
6  para j = 1 até n2, incrementando faça
7      C[j] = A[meio + j]                    Θ(n2)
8  i = 1
9  j = 1                                     Θ(1)
10 k = inicio
11 enquanto i ≤ n1 e j ≤ n2 faça
12     se B[i] ≤ C[j] então
13         A[k] = B[i]
14         i = i + 1
15     senão
16         A[k] = C[j]
17         j = j + 1
18     k = k + 1
19 enquanto i ≤ n1 faça
20     A[k] = B[i]
21     i = i + 1
22     k = k + 1
23 enquanto j ≤ n2 faça
24     A[k] = C[j]
25     j = j + 1                             Θ(n)
26     k = k + 1
```

$\Theta(n_1) + \Theta(n_2) = \Theta(n)$

Combina (A, inicio, meio, fim) leva tempo $\Theta(n)$, onde $n = fim - inicio + 1$, porque ...

Divisão e conquista

→ É uma técnica de projeto de algoritmos que envolve recursão.

→ A ideia é dividir a instância do problema em 2 ou mais instâncias menores (divisão), resolvê-las de forma recursiva e combinar as soluções em uma solução da instância original (conquista).
  ↳ Exemplo clássico: MergeSort (ordenação).

→ Outros exemplos: Quicksort (ordenação), Karatsuba (multiplicação de inteiros) Strassen (multiplicação de matrizes), Contagem de inversões em vetor.