Análise de Algoritmos

Parte destes slides são adaptações de slides dos Profs.

Paulo Feofiloff, José Coelho de Pina e Cristina G. Fernandes

7 de agosto de 2025

Análise de Algoritmos

CLRS 2.3, 3.1 e 3.2

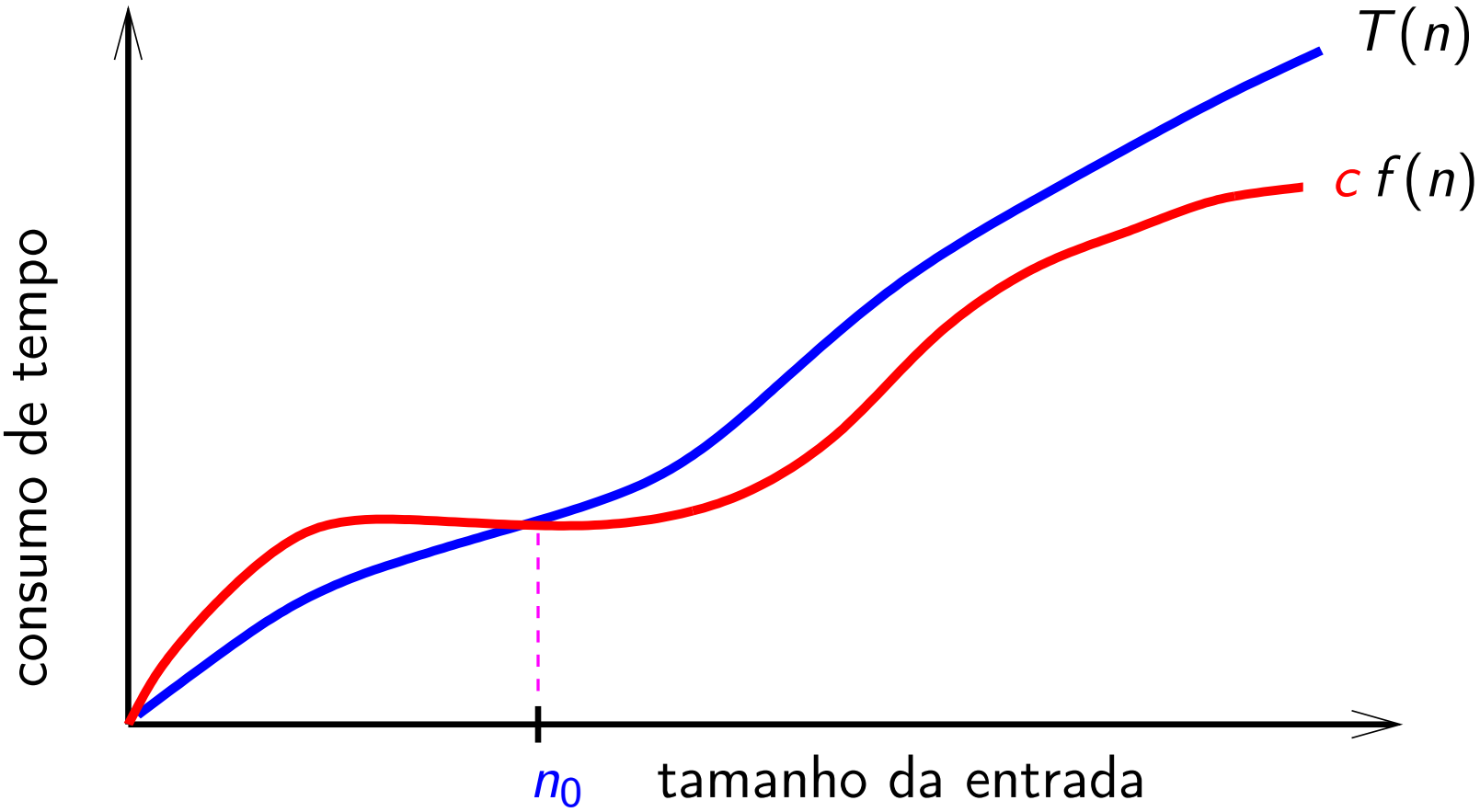
AU 3.3, 3.4 e 3.6

Notação Omega

Dizemos que $T(n)$ é $\Omega(f(n))$ se
existem constantes positivas c e n_0 tais que

$$c\,f(n) \leq T(n)$$

para todo $n \geq n_0$.

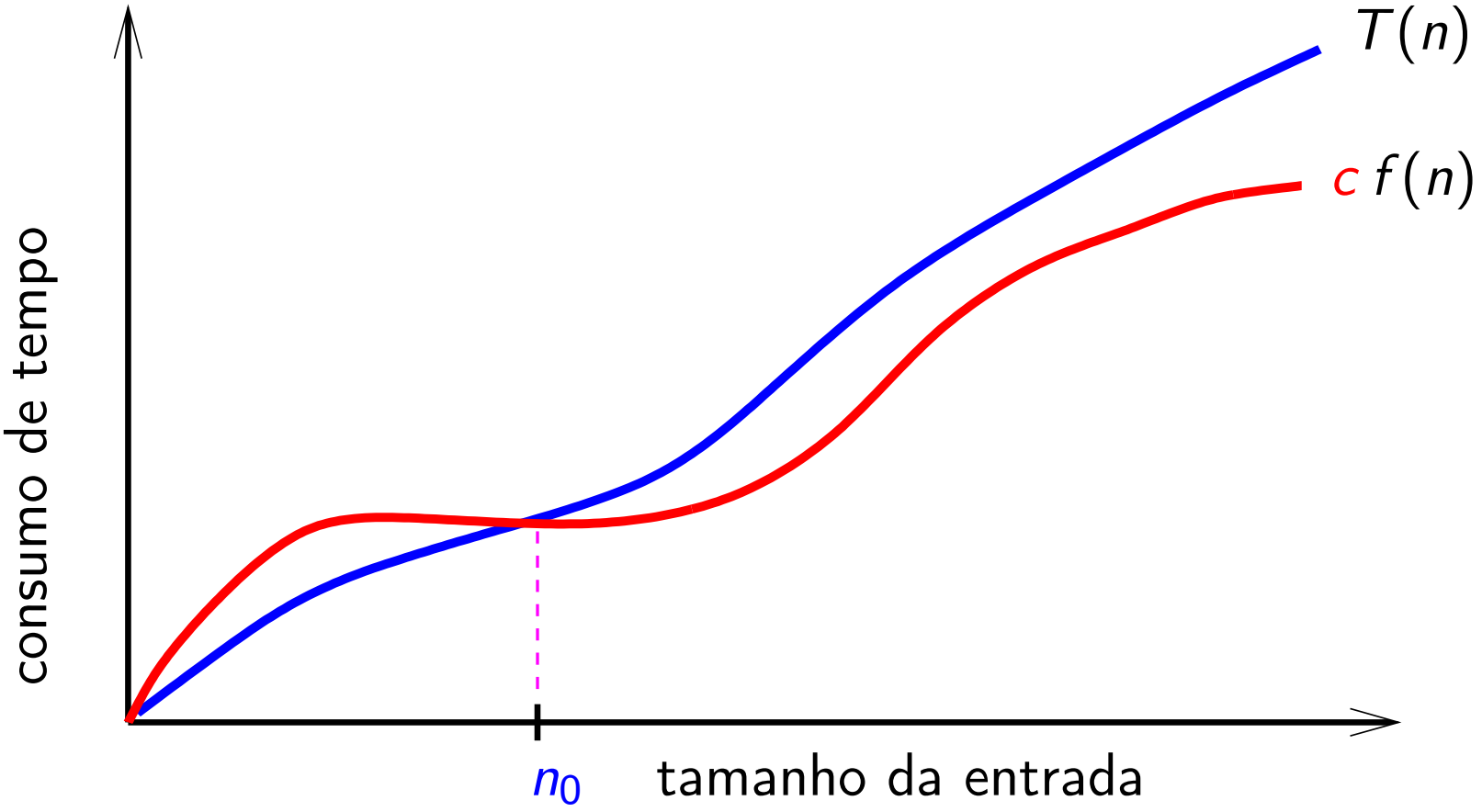


Mais informal

$T(n) = \Omega(f(n))$ se existe $c > 0$ tal que

$$c\, f(n) \leq T(n)$$

para todo n suficientemente GRANDE.



Exemplos

Exemplo 1

Se $T(n) \geq 0.001n^2$ para todo $n \geq 8$, então $T(n)$ é $\Omega(n^2)$.

Exemplos

Exemplo 1
Se $T(n) \geq 0.001n^2$ para todo $n \geq 8$, então $T(n)$ é $\Omega(n^2)$.

Prova: Aplique a definição com $c = 0.001$ e $n_0 = 8$.

Exemplo 2

O consumo de tempo do ORDENA-POR-INSERÇÃO é $\mathrm{O}(n^2)$ e $\Omega(n)$.

```
ORDENA-POR-INSERÇÃO (A, n)
1   para j ← 2 até n faça
2       chave ← A[j]
3       i ← j − 1
4       enquanto i ≥ 1 e A[i] > chave faça
5           A[i + 1] ← A[i]      ▷ desloca
6           i ← i − 1
7       A[i + 1] ← chave         ▷ insere
```

Exemplo 2

O consumo de tempo do ORDENA-POR-INSERÇÃO é $\mathrm{O}(n^2)$ e $\Omega(n)$.

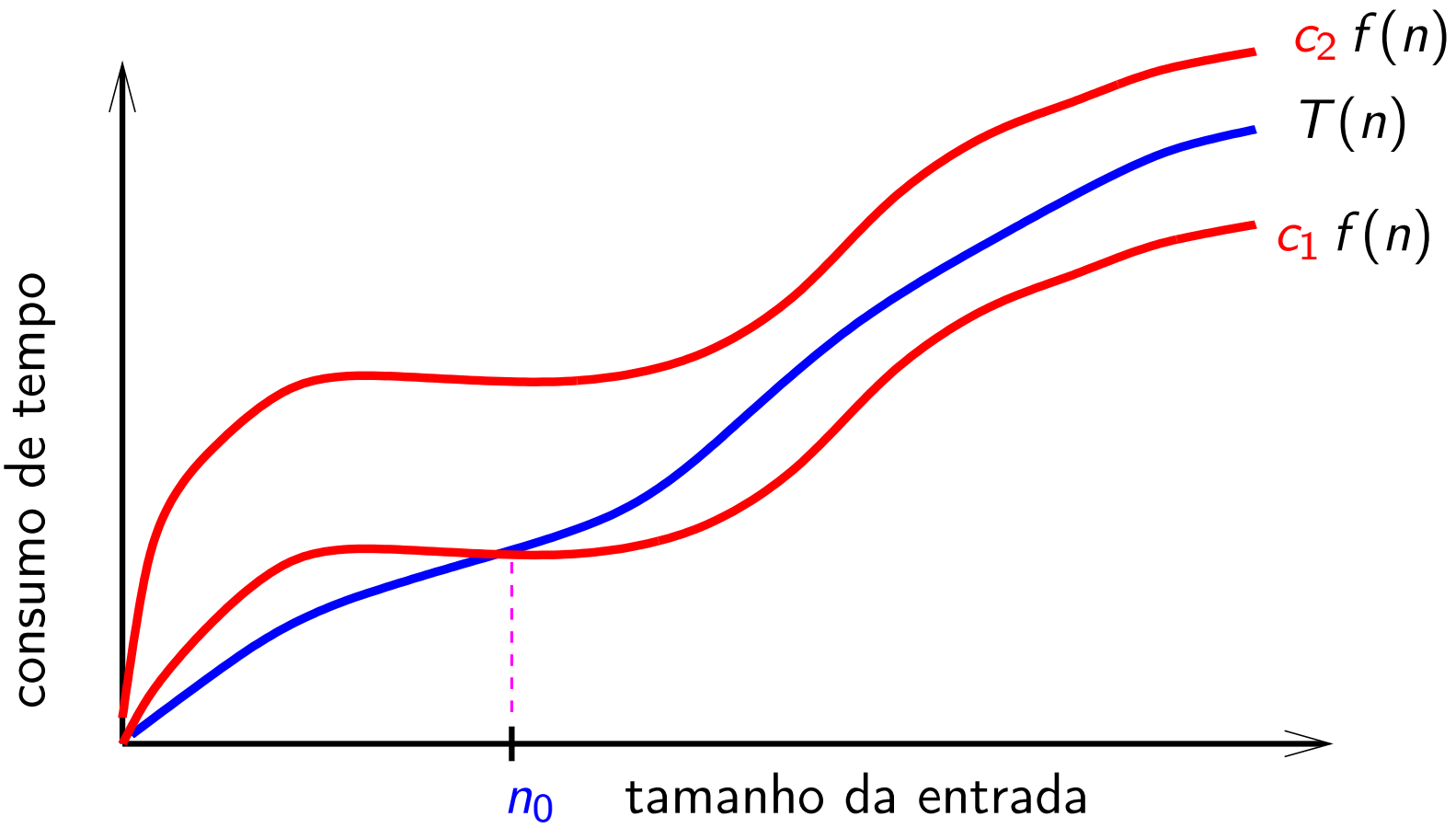
```
ORDENA-POR-INSERÇÃO (A, n)
1   para j ← 2 até n faça
2       chave ← A[j]
3       i ← j − 1
4       enquanto i ≥ 1 e A[i] > chave faça
5           A[i + 1] ← A[i]      ▷ desloca
6           i ← i − 1
7       A[i + 1] ← chave         ▷ insere
```

linha	todas as execuções da linha	
1	$=$	n
2-3	$=$	$n-1$
4	$\geq$	$n-1$
5-6	$\geq$	0
7	$=$	$n-1$
total	$\geq$	$4n-3 = \Omega(n)$

Notação Theta

Sejam $T(n)$ e $f(n)$ funções dos inteiros no reais.
Dizemos que $T(n)$ é $\Theta(f(n))$ se

$T(n)$ é $\mathrm{O}(f(n))$ e $T(n)$ é $\Omega(f(n))$.

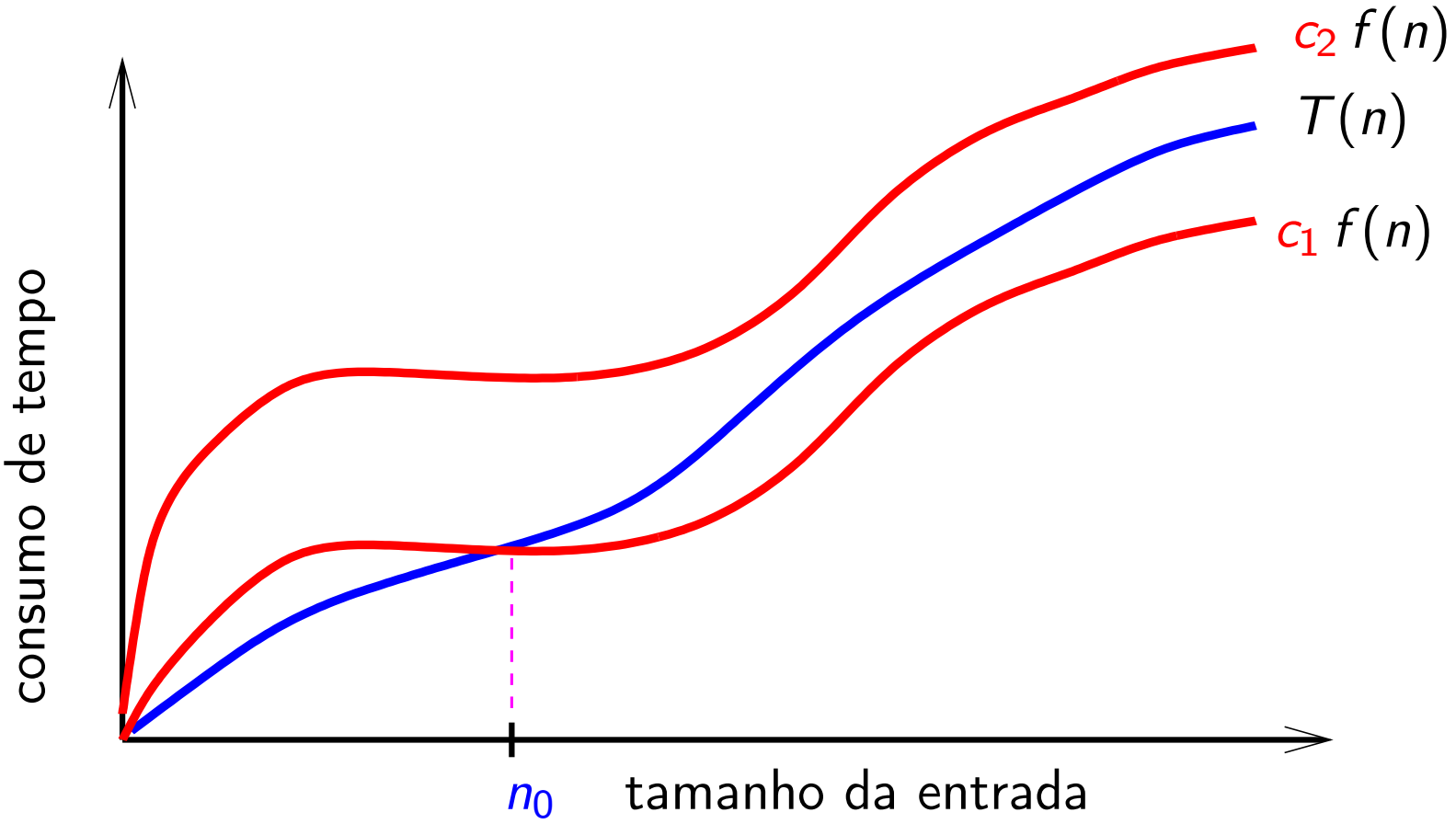


Notação Theta

Dizemos que $T(n)$ é $\Theta(f(n))$ se
existem constantes positivas c_1, c_2 e n_0 tais que

$$c_1\, f(n) \;\leq\; T(n) \;\leq\; c_2\, f(n)$$

para todo $n \geq n_0$.



Intuitivamente

Comparação assintótica, ou seja, para n ENORME.

comparação	comparação assintótica
$T(n) \leq f(n)$	$T(n)$ é $\mathrm{O}(f(n))$
$T(n) \geq f(n)$	$T(n)$ é $\Omega(f(n))$
$T(n) = f(n)$	$T(n)$ é $\Theta(f(n))$

Tamanho máximo de problemas

Suponha que cada operação consome 1 microsegundo ($1\mu s$).

consumo de tempo (μs)	Tamanho máximo de problemas (n)		
	1 segundo	1 minuto	1 hora
$400n$	2500	150000	9000000
$20\,n\lceil \lg n \rceil$	4096	166666	7826087
$2n^2$	707	5477	42426
n^4	31	88	244
2^n	19	25	31

Michael T. Goodrich e Roberto Tamassia,
Projeto de Algoritmos, Bookman.

Crescimento de algumas funções

n	$\lg n$	$\sqrt{n}$	$n \lg n$	n^2	n^3	2^n
2	1	1,4	2	4	8	4
4	2	2	8	16	64	16
8	3	2,8	24	64	512	256
16	4	4	64	256	4096	65536
32	5	5,7	160	1024	32768	4294967296
64	6	8	384	4096	262144	$1{,}8\ 10^{19}$
128	7	11	896	16384	2097152	$3{,}4\ 10^{38}$
256	8	16	1048	65536	16777216	$1{,}1\ 10^{77}$
512	9	23	4608	262144	134217728	$1{,}3\ 10^{154}$
1024	10	32	10240	1048576	$1{,}1\ 10^{9}$	$1{,}7\ 10^{308}$

Nomes de classes Θ

classe	nome
$\Theta(1)$	constante
$\Theta(\log n)$	logarítmica
$\Theta(n)$	linear
$\Theta(n \log n)$	$n \log n$
$\Theta(n^2)$	quadrática
$\Theta(n^3)$	cúbica
$\Theta(n^k)$ com $k \geq 1$	polinomial
$\Theta(2^n)$	exponencial
$\Theta(a^n)$ com $a > 1$	exponencial

Palavras de Cautela

Suponha que $\mathcal{A}$ e $\mathcal{B}$ são algoritmos para um mesmo problema.
Suponha que o consumo de tempo de $\mathcal{A}$ é "essencialmente" $100\,n$
e que o consumo de tempo de $\mathcal{B}$ é "essencialmente" $n \log_{10} n$.

Palavras de Cautela

Suponha que $\mathcal{A}$ e $\mathcal{B}$ são algoritmos para um mesmo problema.
Suponha que o consumo de tempo de $\mathcal{A}$ é "essencialmente" $100\,n$
e que o consumo de tempo de $\mathcal{B}$ é "essencialmente" $n \log_{10} n$.

$100\,n$ é $\Theta(n)$ e $n \log_{10} n$ é $\Theta(n \lg n)$.
Logo, $\mathcal{A}$ é assintoticamente mais eficiente que $\mathcal{B}$.

Palavras de Cautela

Suponha que $\mathcal{A}$ e $\mathcal{B}$ são algoritmos para um mesmo problema.
Suponha que o consumo de tempo de $\mathcal{A}$ é "essencialmente" $100\,n$
e que o consumo de tempo de $\mathcal{B}$ é "essencialmente" $n \log_{10} n$.

$100\,n$ é $\Theta(n)$ e $n \log_{10} n$ é $\Theta(n \lg n)$.
Logo, $\mathcal{A}$ é assintoticamente mais eficiente que $\mathcal{B}$.

$\mathcal{A}$ é mais eficiente que $\mathcal{B}$ para $n \geq 10^{100}$.

10^{100} = um *googol*
$>$ número de átomos no universo observável
= um número ENORME

Crescimento de algumas funções

11:24 Wed 9 Aug

en.wikipedia.org

93%

Googol

74 languages

Article Talk Read View source View history Tools

From Wikipedia, the free encyclopedia

Not to be confused with Google or Nikolai Gogol.

A **googol** is the large number $\mathbf{10^{100}}$. In decimal notation, it is written as the digit 1 followed by one hundred zeroes: **10,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000**. Its systematic name is **10 duotrigintillion**.

Etymology

The term was coined in 1920 by 9-year-old Milton Sirotta (1911–1981), nephew of U.S. mathematician Edward Kasner.[1] He may have been inspired by the contemporary comic strip character Barney Google.[2] Kasner popularized the concept in his 1940 book *Mathematics and the Imagination*.[3] Other names for this quantity include *ten duotrigintillion* on the short scale,[4] *ten thousand sexdecillion* on the long scale, or *ten sexdecilliard* on the Peletier long scale.

11:23 Wed 9 Aug
en.wikipedia.org
93%

Eddington number

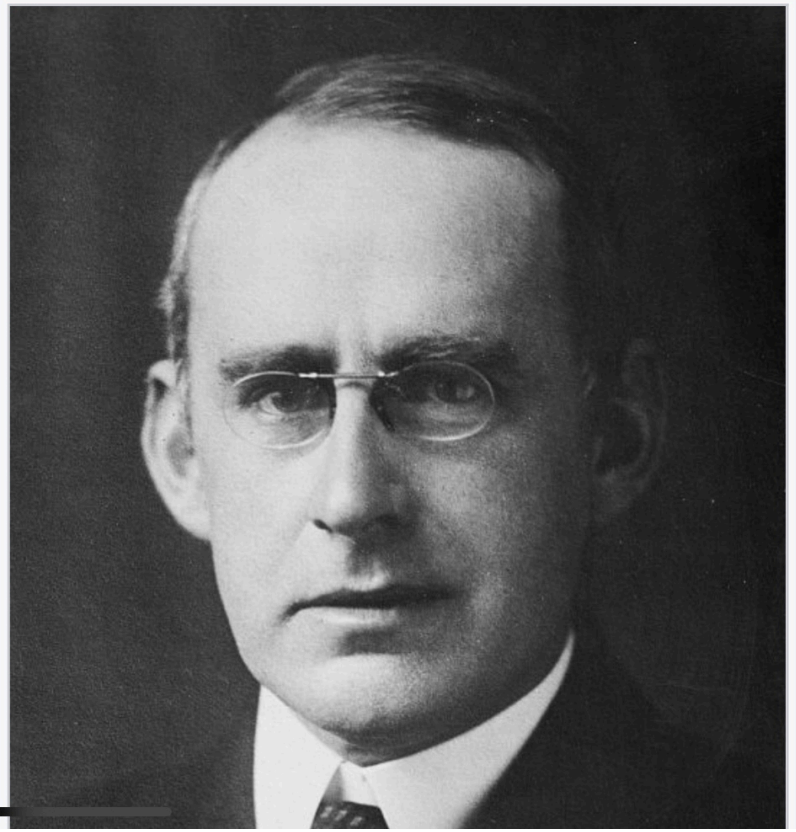
7 languages

Article Talk Read Edit View history Tools

From Wikipedia, the free encyclopedia

This article is about astrophysics. For the measure in cycling, see Arthur Eddington § Eddington number for cycling.

In astrophysics, the **Eddington number**, N_{Edd}, is the number of protons in the observable universe. Eddington originally calculated it as about 1.57×10^{79}; current estimates make it approximately 10^{80}.



The term is named for British astrophysicist Arthur Eddington, who in 1940 was the first to propose a value of N_{Edd} and to explain why this number might be important for physical cosmology and the foundations of physics.

History [edit]

Eddington argued that the value of the fine-structure constant, α, could be obtained by pure deduction. He related α to the Eddington number, which was his

Palavras de Cautela

Conclusão:

> Lembre das constantes e termos de baixa ordem
> que estão "escondidos" na notação assintótica.

Em geral um algoritmo que consome tempo $\Theta(n \lg n)$,
e com fatores constantes razoáveis, é bem eficiente.

Um algoritmo que consome tempo $\Theta(n^2)$ pode,
algumas vezes, ser satisfatório.

Um algoritmo que consome tempo $\Theta(2^n)$ é dificilmente aceitável.

Do ponto de vista de AA, eficiente = polinomial.

Análise do caso médio

Suponha que a entrada do algoritmo é uma permutação de 1 a n escolhida com probabilidade uniforme.

Análise do caso médio

Suponha que a entrada do algoritmo é uma permutação de 1 a n escolhida com probabilidade uniforme.

Qual é o número esperado de execuções da linha 4?

```
ORDENA-POR-INSERÇÃO (A, n)
1  para j ← 2 até n faça
2      chave ← A[j]
3      i ← j − 1
4      enquanto i ≥ 1 e A[i] > chave faça
5          A[i + 1] ← A[i]      ▷ desloca
6          i ← i − 1
7      A[i + 1] ← chave         ▷ insere
```

Análise do caso médio

Suponha que a entrada do algoritmo é uma permutação de 1 a n escolhida com probabilidade uniforme.

```
ORDENA-POR-INSERÇÃO (A, n)
1  para j ← 2 até n faça
2      chave ← A[j]
3      i ← j − 1
4      enquanto i ≥ 1 e A[i] > chave faça
5          A[i + 1] ← A[i]        ▷ desloca
6          i ← i − 1
7      A[i + 1] ← chave           ▷ insere
```

Para cada valor de j, a linha 4 é executada de 1 a j vezes.

Análise do caso médio

Suponha que a entrada do algoritmo é uma permutação de 1 a n escolhida com probabilidade uniforme.

```
ORDENA-POR-INSERÇÃO (A, n)
1  para j ← 2 até n faça
2      chave ← A[j]
3      i ← j − 1
4      enquanto i ≥ 1 e A[i] > chave faça
5          A[i + 1] ← A[i]        ▷ desloca
6          i ← i − 1
7      A[i + 1] ← chave           ▷ insere
```

Para cada valor de j, a linha 4 é executada de 1 a j vezes.

Qual é a probabilidade de ela ser executada t vezes, para cada $t \in \{1, \ldots, j\}$?

Análise do caso médio

Suponha que a entrada do algoritmo é uma permutação de 1 a n escolhida com probabilidade uniforme.

```
ORDENA-POR-INSERÇÃO (A, n)
1  para j ← 2 até n faça
2      chave ← A[j]
3      i ← j − 1
4      enquanto i ≥ 1 e A[i] > chave faça
5          A[i + 1] ← A[i]        ▷ desloca
6          i ← i − 1
7      A[i + 1] ← chave           ▷ insere
```

Para cada valor de j, a linha 4 é executada de 1 a j vezes.

Qual é a probabilidade de ela ser executada t vezes, para cada $t \in \{1, \ldots, j\}$?

Qual é a probabilidade de ela ser executada $t + 1$ vezes, para cada $t \in \{0, \ldots, j - 1\}$?

Análise do caso médio

Suponha que a entrada do algoritmo é uma permutação de 1 a n escolhida com probabilidade uniforme.

```
ORDENA-POR-INSERÇÃO (A, n)
1  para j ← 2 até n faça
2      chave ← A[j]
3      i ← j − 1
4      enquanto i ≥ 1 e A[i] > chave faça
5          A[i + 1] ← A[i]        ▷ desloca
6          i ← i − 1
7      A[i + 1] ← chave           ▷ insere
```

Para cada valor de j, a linha 4 é executada de 1 a j vezes.

Qual é a probabilidade de ela ser executada $t + 1$ vezes, para cada $t \in \{0, \ldots, j - 1\}$?

Ocorre sse $A[1\,..\,j-1]$ tem exatamente t números que são $> A[j]$.

prob(A[1..j] tem exatamente t elementos maiores que A[j])

Para construir qualquer (e toda!) permutação A[1..n] de 1 a n tq A[1..j] tem exatamente t elementos maiores que A[j], podemos:

↑ $\in \{0, \ldots, j-1\}$

prob(A[1..j] tem exatamente t elementos maiores que A[j])

Para construir qualquer (e toda!) permutação A[1..n] de 1 a n tq A[1..j] tem exatamente t elementos maiores que A[j], podemos:

↑ $\in \{0, \ldots, j-1\}$

1. escolher uniformemente um conjunto $S \subseteq \{1, \ldots, n\}$ de j elementos para preencher as células de A[1..j]



estas células serão preenchidas com elementos de S

prob(A[1..j] tem exatamente t elementos maiores que A[j])

Para construir qualquer (e toda!) permutação A[1..n] de 1 a n tq A[1..j] tem exatamente t elementos maiores que A[j], podemos: ← $\in \{0, \ldots, j-1\}$

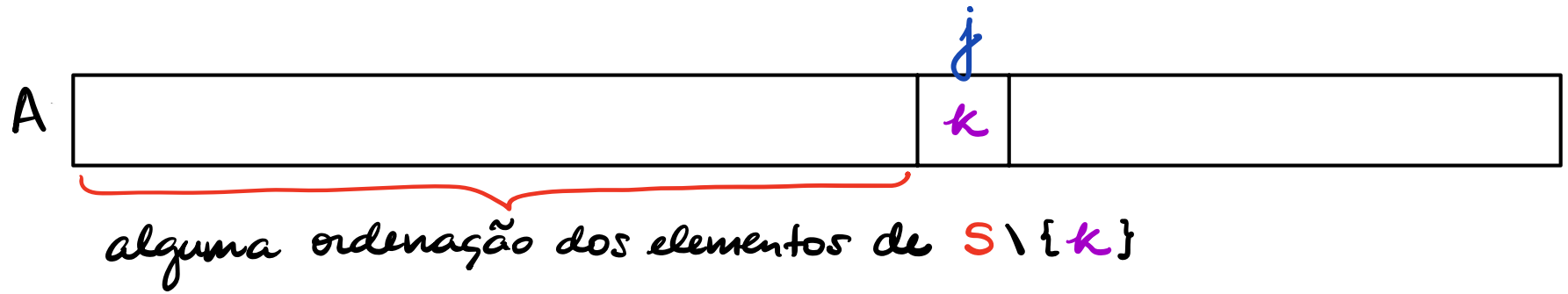
1. escolher uniformemente um conjunto $S \subseteq \{1, \ldots, n\}$ de j elementos para preencher as células de A[1..j]
2. escolher o único $k \in S$ t.q. S tem exatamente t elementos maiores que k, e tomar $A[j] \leftarrow k$



prob(A[1..j] tem exatamente t elementos maiores que A[j])

Para construir qualquer (e toda!) permutação A[1..n] de 1 a n tq A[1..j] tem exatamente t elementos maiores que A[j], podemos: ← $\in \{0, \ldots, j-1\}$

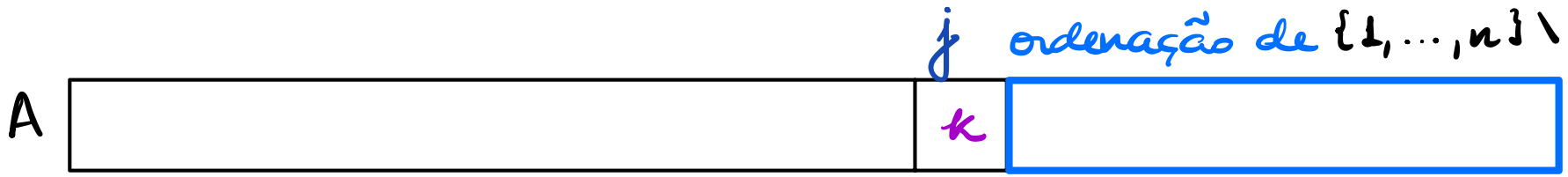
1. escolher uniformemente um conjunto $S \subseteq \{1, \ldots, n\}$ de j elementos para preencher as células de A[1..j]
2. escolher o único $k \in S$ t.q. S tem exatamente t elementos maiores que k, e tomar $A[j] \leftarrow k$
3. preencher A[1..j−1] com os elementos de $S \setminus \{k\}$



prob(A[1..j] tem exatamente t elementos maiores que A[j])

Para construir qualquer (e toda!) permutação A[1..n] de 1 a n tq A[1..j] tem exatamente t elementos maiores que A[j], podemos: ← $\in \{0, \ldots, j-1\}$

1. escolher uniformemente um conjunto $S \subseteq \{1, \ldots, n\}$ de j elementos para preencher as células de A[1..j]
2. escolher o único $k \in S$ t.q. S tem exatamente t elementos maiores que k, e tomar $A[j] \leftarrow k$
3. preencher A[1..j−1] com os elementos de $S \setminus \{k\}$
4. preencher A[j+1..n] com os elementos de $\{1, \ldots, n\} \setminus S$



prob(A[1..j] tem exatamente t−1 elementos maiores que A[j])

1. escolher uniformemente um conjunto $S \subseteq \{1,\ldots,n\}$ de j elementos para preencher as células de $A[1..j]$
2. escolher o único $k \in S$ t.q. S tem exatamente $t-1$ elementos maiores que k, e tomar $A[j] \leftarrow k$
3. preencher $A[1..j-1]$ com os elementos de $S \setminus \{k\}$
4. preencher $A[j+1..n]$ com os elementos de $\{1,\ldots,n\} \setminus S$

$$\text{probabilidade} = \frac{\binom{n}{j} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!}$$

prob(A[1..j] tem exatamente t elementos maiores que A[j])

1. escolher uniformemente um conjunto $S \subseteq \{1, \ldots, n\}$ de j elementos para preencher as células de $A[1..j]$
2. escolher o único $k \in S$ t.q. S tem exatamente t elementos maiores que k, e tomar $A[j] \leftarrow k$
3. preencher $A[1..j-1]$ com os elementos de $S \setminus \{k\}$
4. preencher $A[j+1..n]$ com os elementos de $\{1, \ldots, n\} \setminus S$

$$\text{probabilidade} = \frac{\binom{n}{j} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!}$$

$$= \frac{\frac{n!}{j!\,(n-j)!} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!}$$

prob(A[1..j] tem exatamente t elementos maiores que A[j])

1. escolher uniformemente um conjunto $S \subseteq \{1,\ldots,n\}$ de j elementos para preencher as células de $A[1..j]$
2. escolher o único $k \in S$ t.q. S tem exatamente t elementos maiores que k, e tomar $A[j] \leftarrow k$
3. preencher $A[1..j-1]$ com os elementos de $S \setminus \{k\}$
4. preencher $A[j+1..n]$ com os elementos de $\{1,\ldots,n\} \setminus S$

- $$\text{probabilidade} = \frac{\binom{n}{j} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!}$$

$$= \frac{\frac{n!}{j!\,(n-j)!} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!}$$

prob(A[1..j] tem exatamente t elementos maiores que A[j])

1. escolher uniformemente um conjunto $S \subseteq \{1, \ldots, n\}$ de j elementos para preencher as células de $A[1..j]$
2. escolher o único $k \in S$ t.q. S tem exatamente t elementos maiores que k, e tomar $A[j] \leftarrow k$
3. preencher $A[1..j-1]$ com os elementos de $S \setminus \{k\}$
4. preencher $A[j+1..n]$ com os elementos de $\{1, \ldots, n\} \setminus S$

$$\text{probabilidade} = \frac{\binom{n}{j} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!}$$

$$= \frac{\frac{n!}{j!\,(n-j)!} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!}$$

prob(A[1..j] tem exatamente t elementos maiores que A[j])

1. escolher uniformemente um conjunto $S \subseteq \{1, \ldots, n\}$ de j elementos para preencher as células de $A[1..j]$
2. escolher o único $k \in S$ t.q. S tem exatamente t elementos maiores que k, e tomar $A[j] \leftarrow k$
3. preencher $A[1..j-1]$ com os elementos de $S \setminus \{k\}$
4. preencher $A[j+1..n]$ com os elementos de $\{1, \ldots, n\} \setminus S$

$$\text{probabilidade} = \frac{\binom{n}{j} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!}$$

$$= \frac{\frac{n!}{j!\,(n-j)!} \cdot 1 \cdot (j-1)!\,(n-j)!}{n!} = \frac{1}{j}$$

Análise do caso médio

Para cada valor de j, a linha 4 é executada de 1 a j vezes.

Qual é a probabilidade de ela ser executada t vezes, para cada $t \in \{1, \ldots, j\}$?

Esta probabilidade é $1/j$.

Análise do caso médio

Para cada valor de j, a linha 4 é executada de 1 a j vezes.

Qual é a probabilidade de ela ser executada t vezes, para cada $t \in \{1, \ldots, j\}$?

Esta probabilidade é $1/j$.

Portanto o número esperado de execuções da linha 4 é

$$\sum_{t=1}^{j} t \frac{1}{j} = \frac{1}{j} \sum_{t=1}^{j} t = \frac{1}{j} \frac{(j+1)j}{2} = \frac{j+1}{2}.$$

Análise do caso médio

Para cada valor de j, a linha 4 é executada de 1 a j vezes.

Qual é a probabilidade de ela ser executada t vezes, para cada $t \in \{1, \ldots, j\}$?

Esta probabilidade é $1/j$.

Portanto o número esperado de execuções da linha 4 é

$$\sum_{t=1}^{j} t \frac{1}{j} = \frac{1}{j} \sum_{t=1}^{j} t = \frac{1}{j} \frac{(j+1)j}{2} = \frac{j+1}{2}.$$

E o número esperado de execuções da linha 4 no total é

$$\sum_{j=2}^{n} \frac{j+1}{2} = \frac{1}{2} \sum_{j=2}^{n} (j+1) = \frac{(n+4)(n-1)}{4} = \Theta(n^2).$$

Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique crescente.

Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique crescente.

ORDENA-POR-INSERÇÃO consome tempo $O(n^2)$ no pior caso e $\Theta(n^2)$ no caso médio.

Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique crescente.

ORDENA-POR-INSERÇÃO consome tempo $O(n^2)$ no pior caso e $\Theta(n^2)$ no caso médio.

Conseguimos fazer melhor?

Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique crescente.

ORDENA-POR-INSERÇÃO consome tempo $O(n^2)$ no pior caso e $\Theta(n^2)$ no caso médio.

Conseguimos fazer melhor?

Sim! Usando divisão e conquista!

Divisão e conquista

Esse paradigma envolve os seguintes passos:

Divisão e conquista

Esse paradigma envolve os seguintes passos:

Divisão: dividir a instância do problema em instâncias menores do problema.

Divisão e conquista

Esse paradigma envolve os seguintes passos:

Divisão: dividir a instância do problema em instâncias menores do problema.

Conquista: resolver o problema nas instâncias menores recursivamente (ou diretamente, se elas forem pequenas o suficiente).

Divisão e conquista

Esse paradigma envolve os seguintes passos:

Divisão: dividir a instância do problema em instâncias menores do problema.

Conquista: resolver o problema nas instâncias menores recursivamente (ou diretamente, se elas forem pequenas o suficiente).

Combinação: combinar as soluções das instâncias menores para gerar uma solução da instância original.

Mergesort

Rearranja $A[p\,..\,r]$, com $p \leq r$, em ordem crescente.

Método: Divisão e conquista.

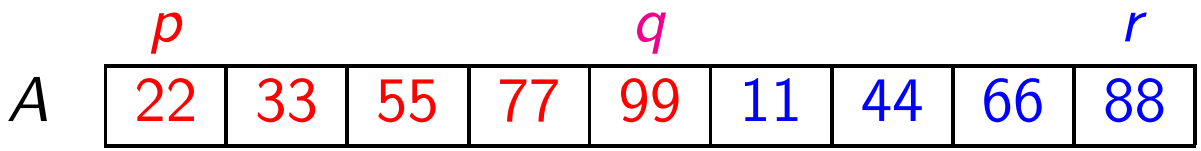
```
MERGESORT (A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT (A, p, q)
4           MERGESORT (A, q + 1, r)
5           INTERCALA (A, p, q, r)
```

Intercalação

Problema: Dados $A[p\,..\,q]$ e $A[q{+}1\,..\,r]$ crescentes,
rearranjar $A[p\,..\,r]$ de modo que ele fique em ordem crescente.

Para que valores de q o problema faz sentido?

Entra:

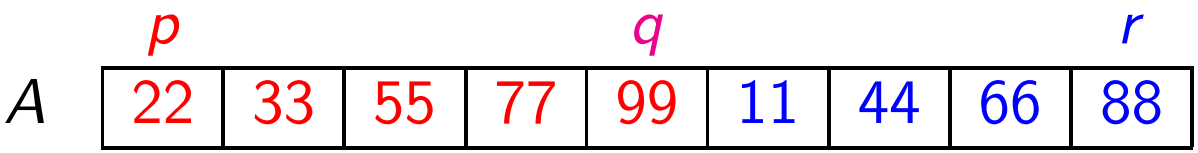


Intercalação

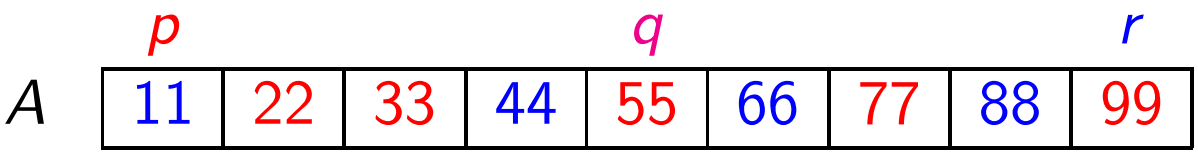
Problema: Dados $A[p\,..\,q]$ e $A[q{+}1\,..\,r]$ crescentes,
rearranjar $A[p\,..\,r]$ de modo que ele fique em ordem crescente.

Para que valores de q o problema faz sentido?

Entra:



Sai:

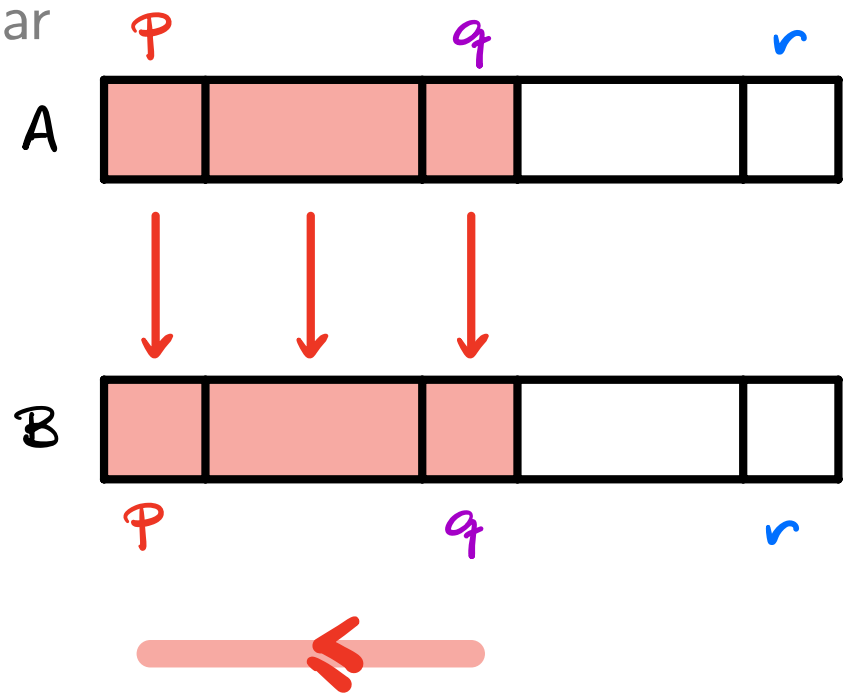


Intercalação

```
INTERCALA (A, p, q, r)
 0   ▷ B[p..r] é um vetor auxiliar
 1   para i ← p até q faça
 2       B[i] ← A[i]
 3   para j ← q + 1 até r faça
 4       B[r + q + 1 − j] ← A[j]
 5   i ← p
 6   j ← r
 7   para k ← p até r faça
 8       se B[i] ≤ B[j]
 9           então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```

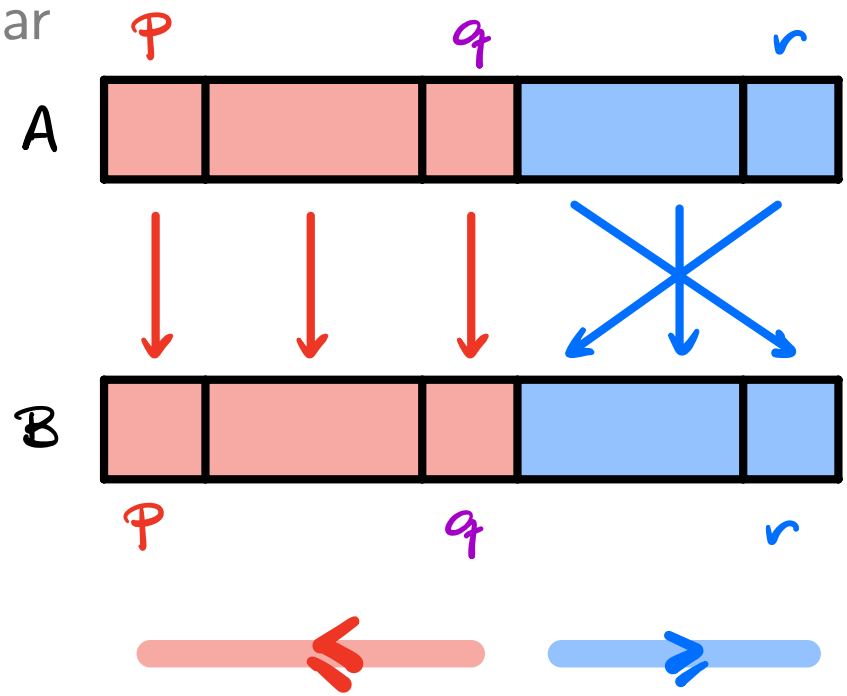
Intercalação

```
INTERCALA (A, p, q, r)
 0  ▷ B[p..r] é um vetor auxiliar
 1  para i ← p até q faça
 2      B[i] ← A[i]
 3  para j ← q + 1 até r faça
 4      B[r + q + 1 − j] ← A[j]
 5  i ← p
 6  j ← r
 7  para k ← p até r faça
 8      se B[i] ≤ B[j]
 9          então A[k] ← B[i]
10                i ← i + 1
11          senão A[k] ← B[j]
12                j ← j − 1
```



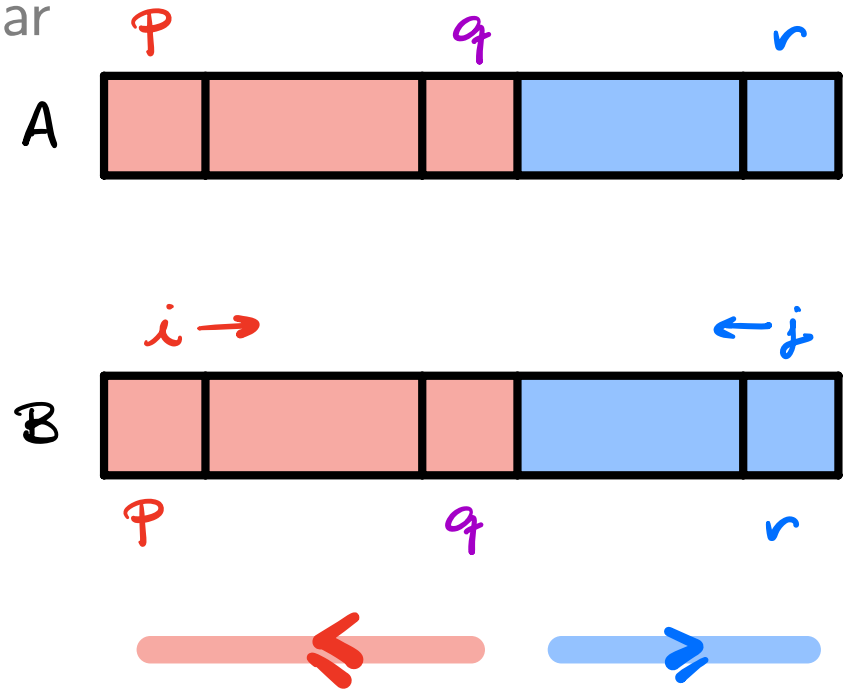
Intercalação

```
INTERCALA (A, p, q, r)
 0   ▷ B[p..r] é um vetor auxiliar
 1   para i ← p até q faça
 2       B[i] ← A[i]
 3   para j ← q + 1 até r faça
 4       B[r + q + 1 − j] ← A[j]
 5   i ← p
 6   j ← r
 7   para k ← p até r faça
 8       se B[i] ≤ B[j]
 9           então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```



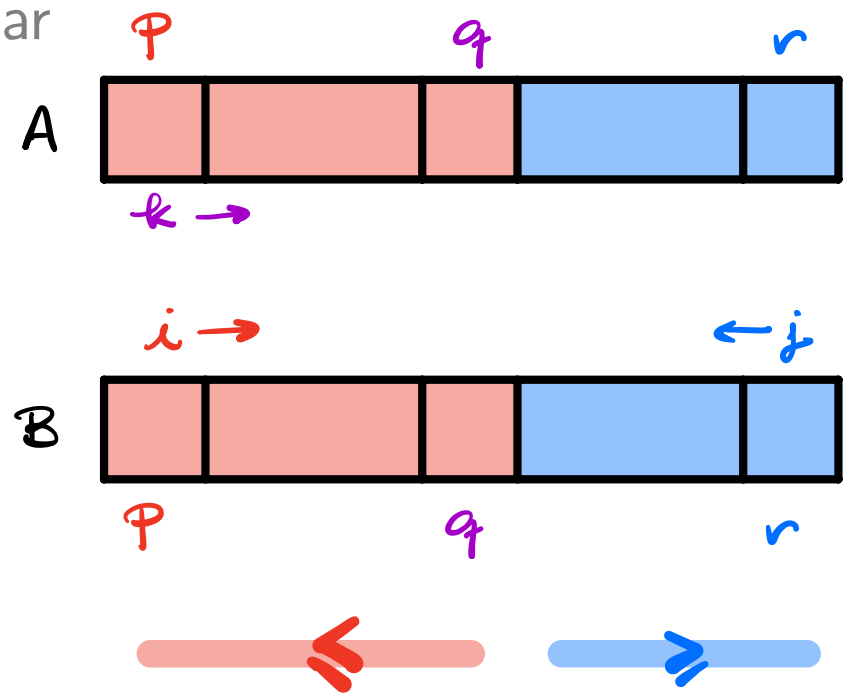
Intercalação

```
INTERCALA (A, p, q, r)
 0   ▷ B[p..r] é um vetor auxiliar
 1   para i ← p até q faça
 2       B[i] ← A[i]
 3   para j ← q + 1 até r faça
 4       B[r + q + 1 − j] ← A[j]
 5   i ← p
 6   j ← r
 7   para k ← p até r faça
 8       se B[i] ≤ B[j]
 9           então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```

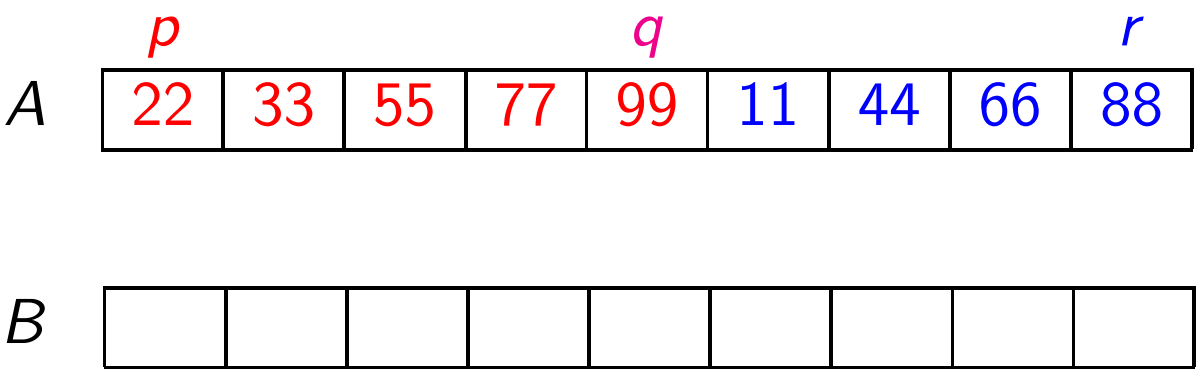


Intercalação

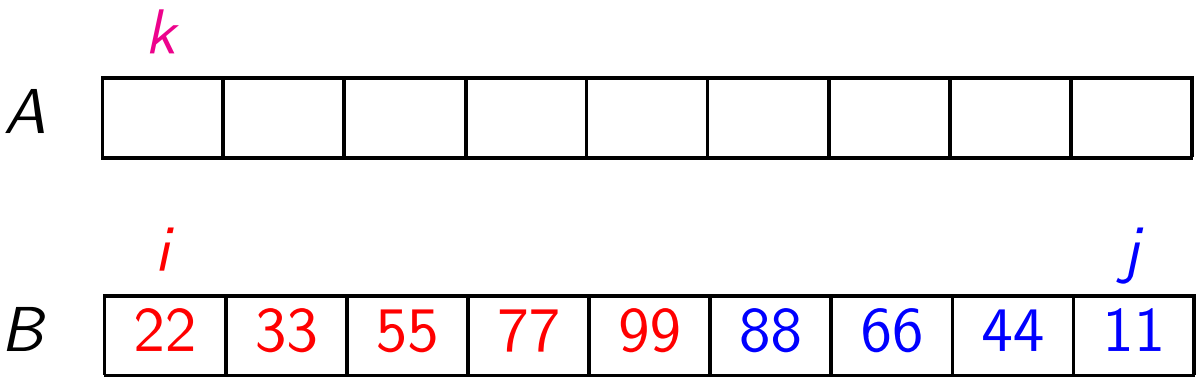
```
INTERCALA (A, p, q, r)
 0   ▷ B[p..r] é um vetor auxiliar
 1   para i ← p até q faça
 2       B[i] ← A[i]
 3   para j ← q + 1 até r faça
 4       B[r + q + 1 − j] ← A[j]
 5   i ← p
 6   j ← r
 7   para k ← p até r faça
 8       se B[i] ≤ B[j]
 9           então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```



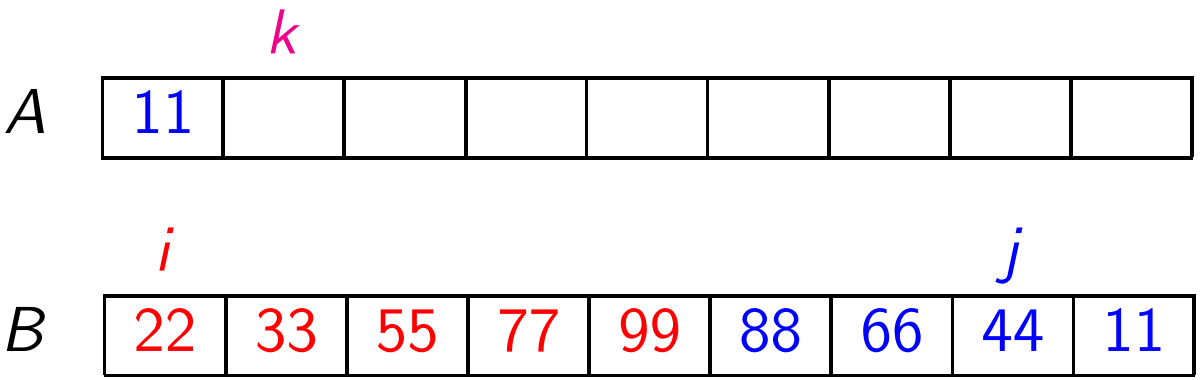
Simulação



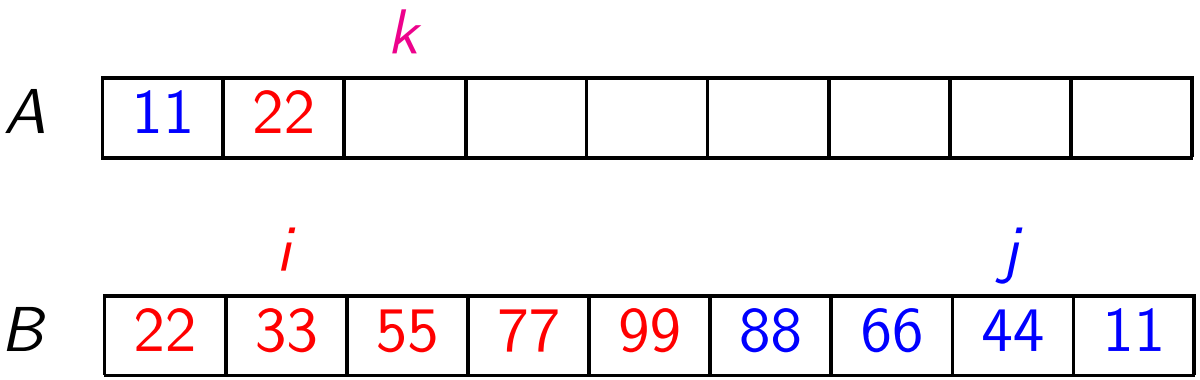
Simulação



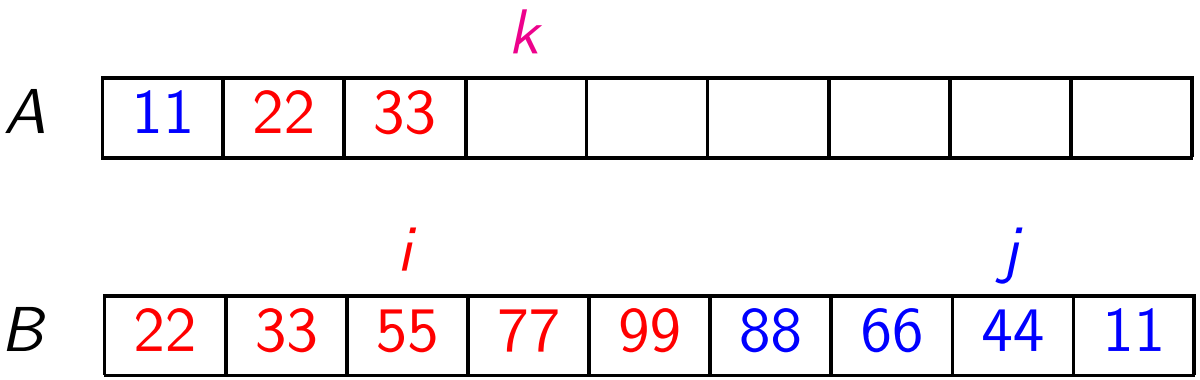
Simulação



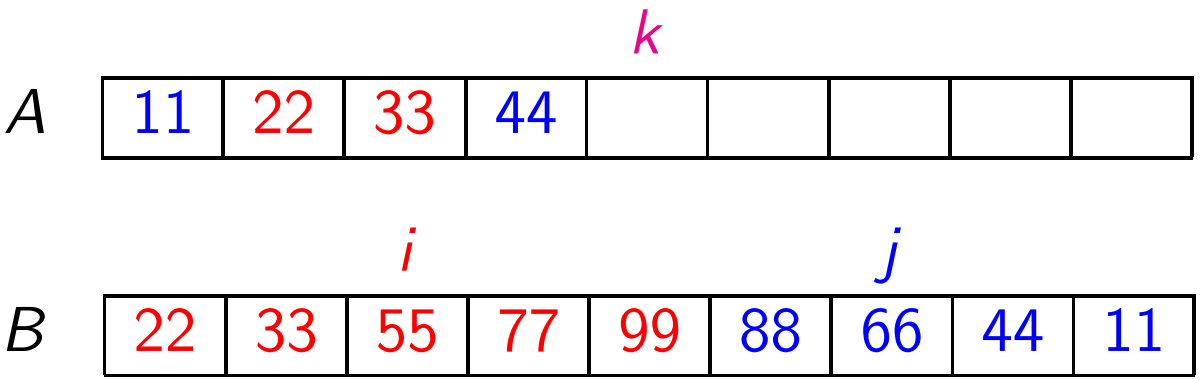
Simulação



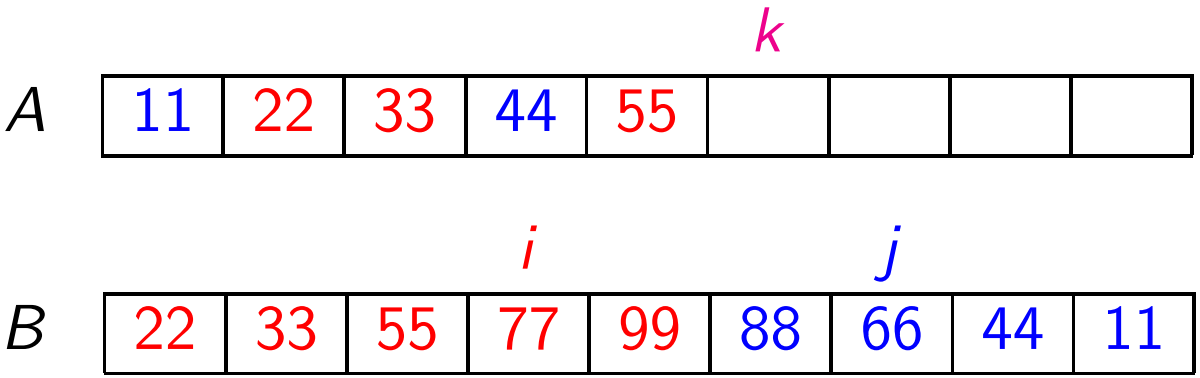
Simulação



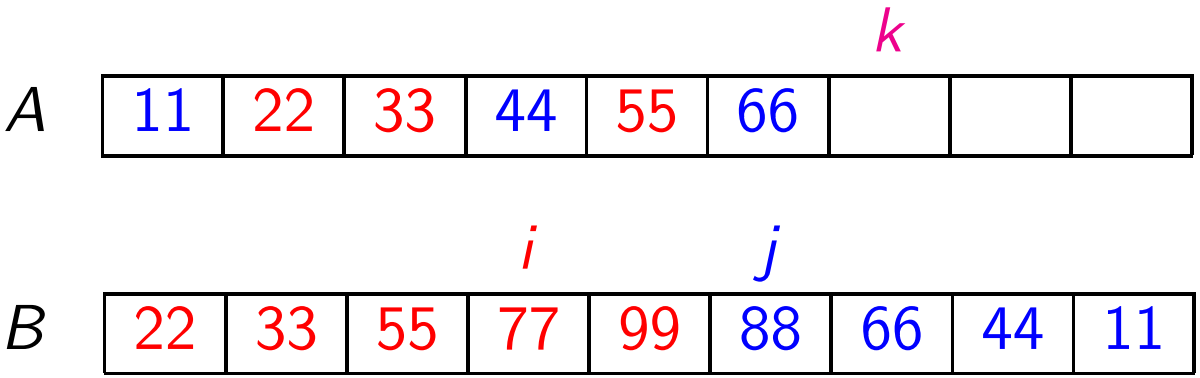
Simulação



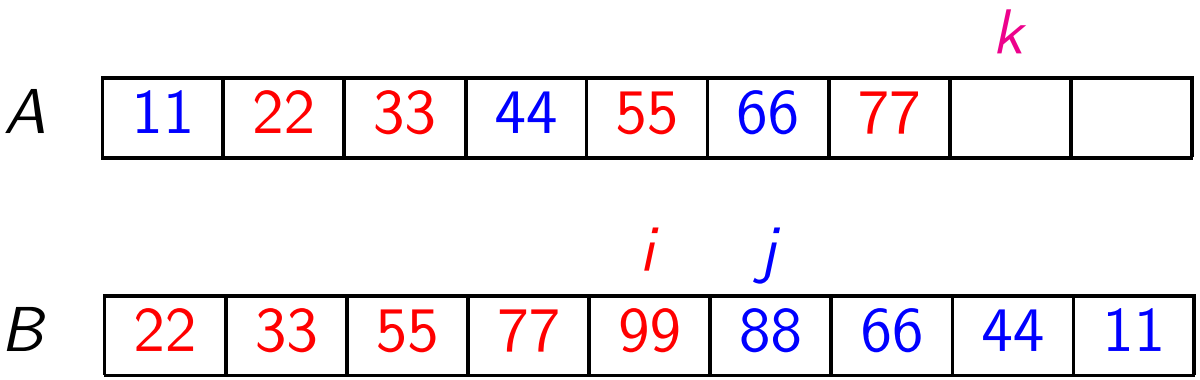
Simulação



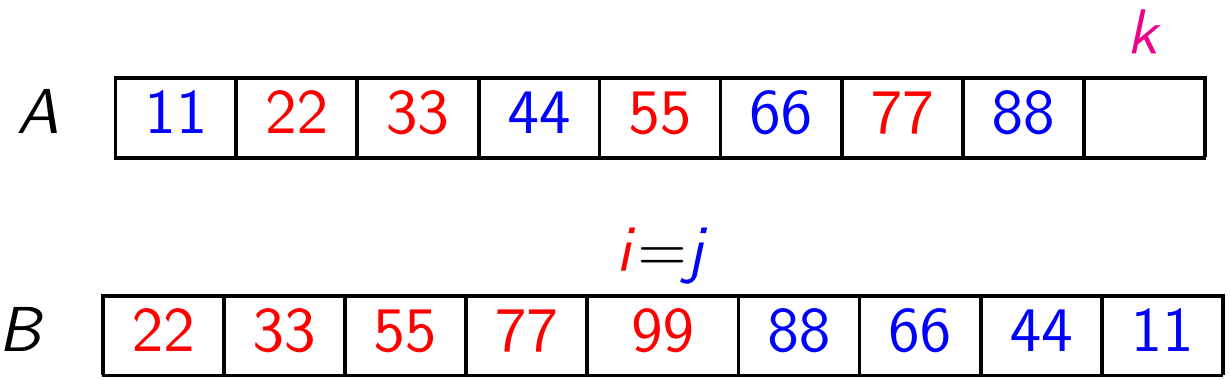
Simulação



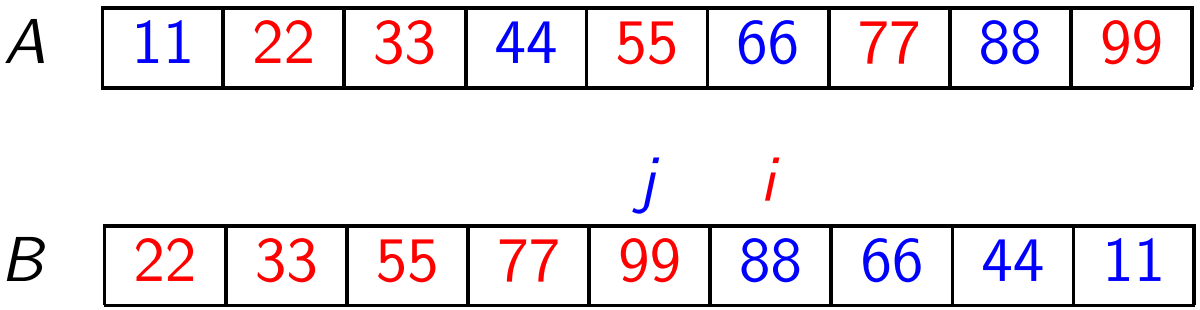
Simulação



Simulação



Simulação

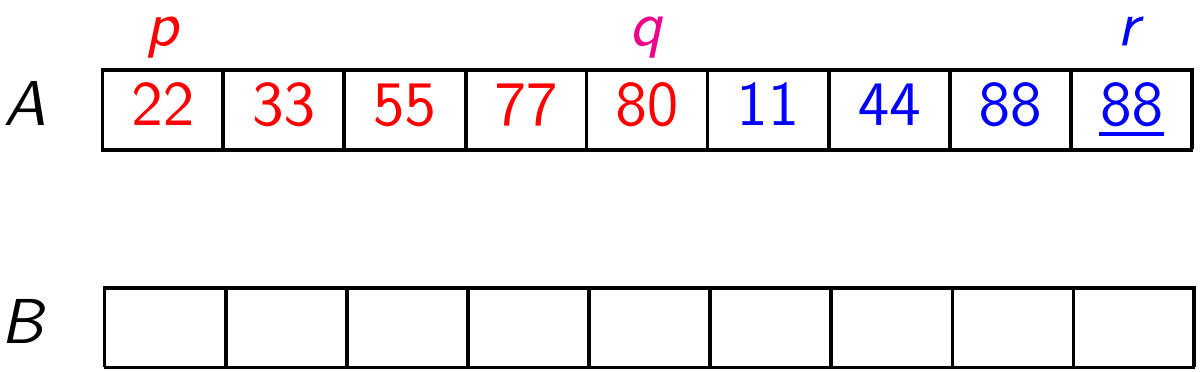


Intercalação

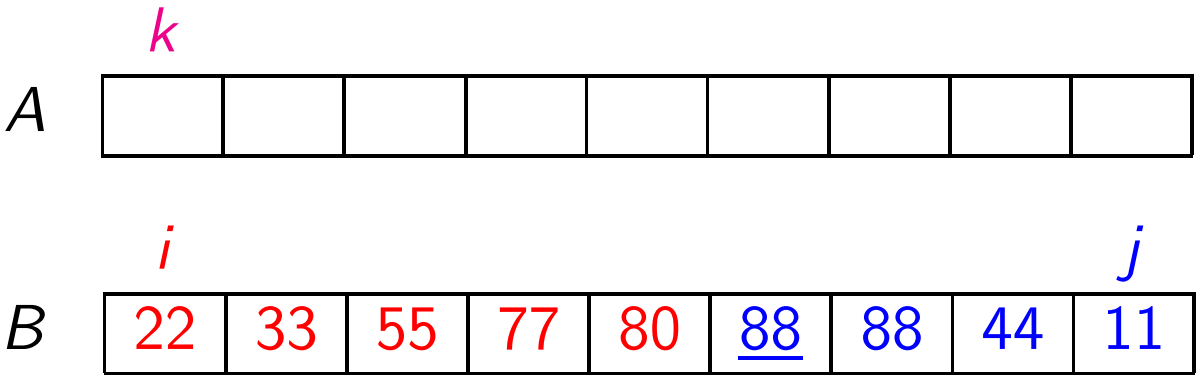
```
INTERCALA (A, p, q, r)
 0   ▷ B[p..r] é um vetor auxiliar
 1   para i ← p até q faça
 2       B[i] ← A[i]
 3   para j ← q + 1 até r faça
 4       B[r + q + 1 − j] ← A[j]
 5   i ← p
 6   j ← r
 7   para k ← p até r faça
 8       se B[i] ≤ B[j]
 9           então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```

Essa versão do intercala não é estável. Por quê?

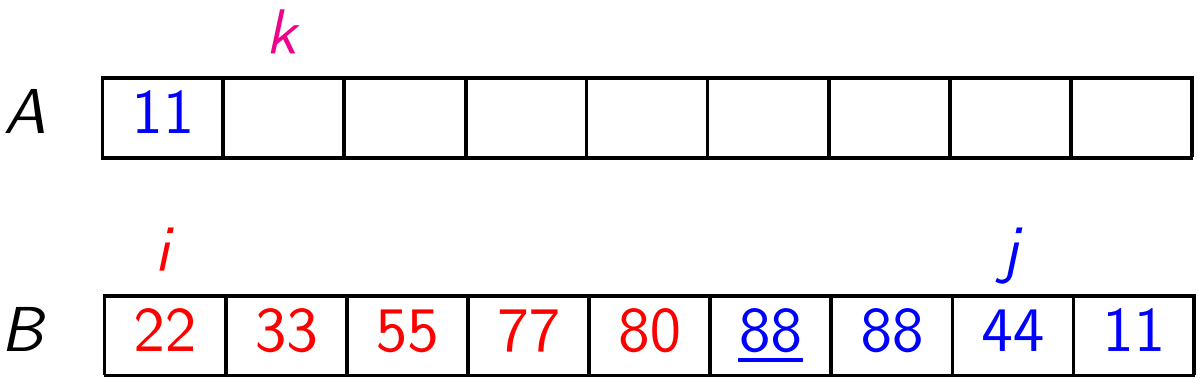
Simulação



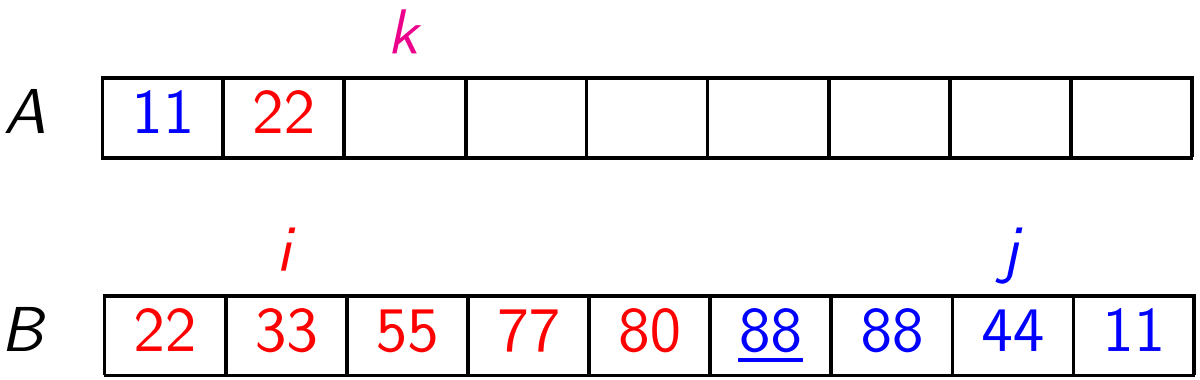
Simulação



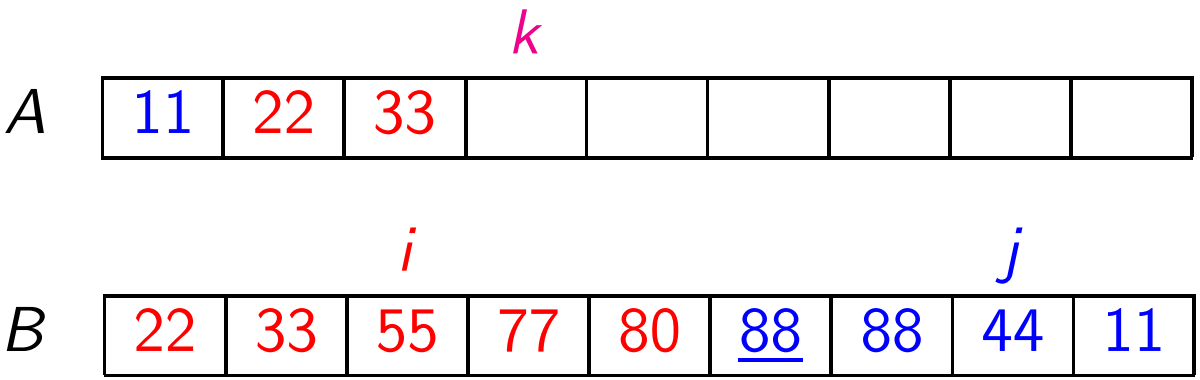
Simulação



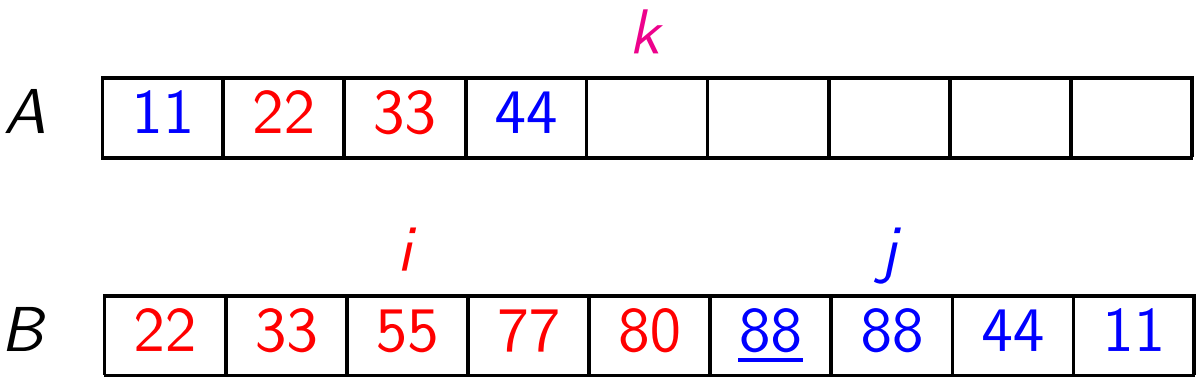
Simulação



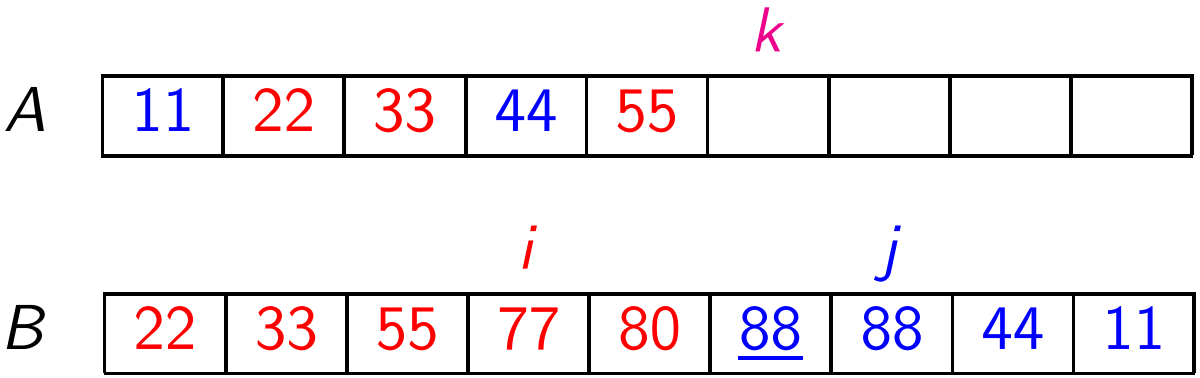
Simulação



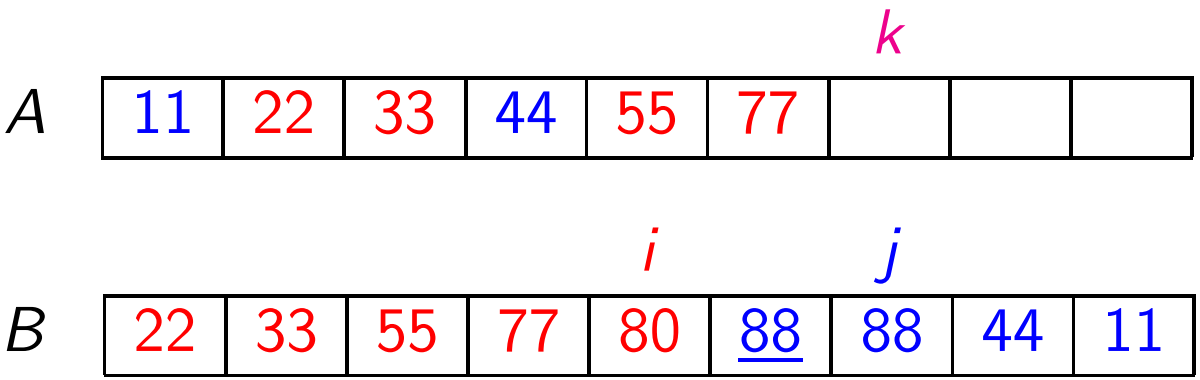
Simulação



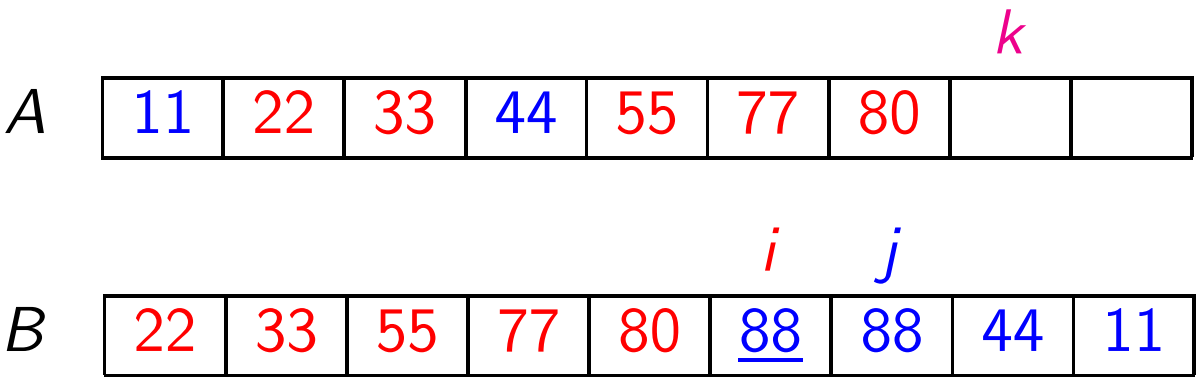
Simulação



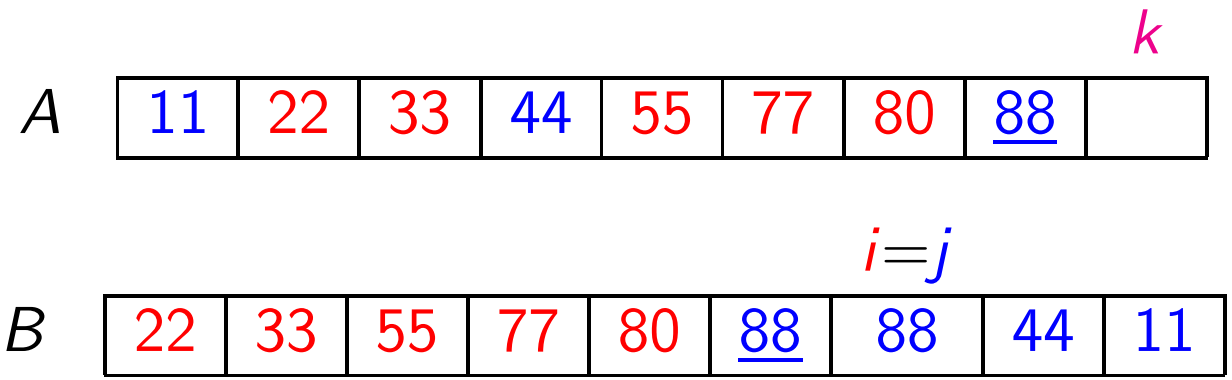
Simulação



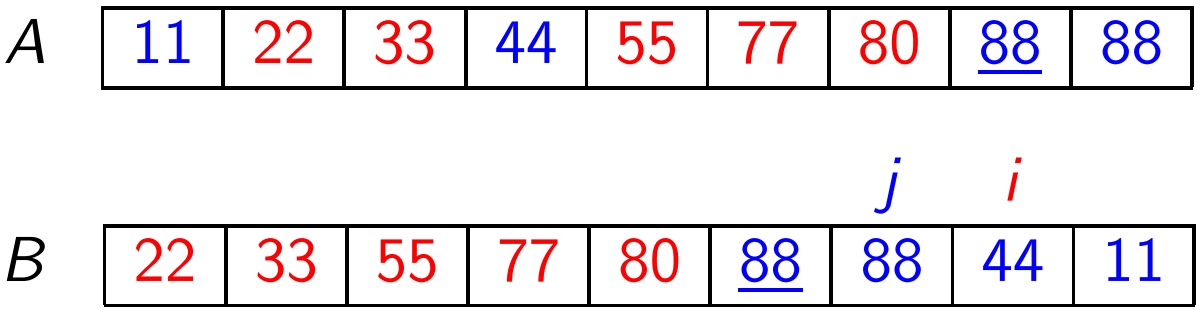
Simulação



Simulação



Simulação



Intercalação

```
INTERCALA (A, p, q, r)
 0   ▷ B[p..r] é um vetor auxiliar
 1   para i ← p até q faça
 2       B[i] ← A[i]
 3   para j ← q + 1 até r faça
 4       B[r + q + 1 − j] ← A[j]
 5   i ← p
 6   j ← r
 7   para k ← p até r faça
 8       se B[i] ≤ B[j] e i ≤ q          ▷ alteração
 9           então A[k] ← B[i]
10                 i ← i + 1
11           senão A[k] ← B[j]
12                 j ← j − 1
```

Essa versão do intercala é estável.

Consumo de tempo

Quanto tempo consome em função de $n := r - p + 1$?

linha	consumo de todas as execuções da linha
1	$\Theta(n)$
2	$\Theta(n)$
3	$\Theta(n)$
4	$\Theta(n)$
5–6	$\Theta(1)$
7	$\Theta(n)$
8	$\Theta(n)$
9–12	$\Theta(n)$
total	$7\Theta(n) + \Theta(1) = \Theta(n)$

Conclusão

O algoritmo INTERCALA consome
$\Theta(n)$ unidades de tempo.

Também escreve-se

O algoritmo INTERCALA consome
tempo $\Theta(n)$.

Aula que vem

Como analisar o MERGESORT?

Recorrências: um jeito de analisar algoritmos recursivos.