



# **ESTRUTURAS DE DADOS**

## **Complexidade**

# Algoritmo

- Algoritmo é qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.
- Portanto, um algoritmo é uma sequência de passos computacionais que transformam a entrada na saída.

# Algoritmo

- Ex.: Entrada: Uma sequência de  $n$  números  
 $\langle a_1, a_2, \dots, a_n \rangle$
- Saída: Uma permutação  $\langle x_1, x_2, \dots, x_n \rangle$   
da sequência de entrada, tal que  
 $x_1 \leq x_2 \leq \dots \leq x_n$

# Algoritmo

- Um algoritmo é dito correto se, para cada instância de entrada, ele para com a saída correta.
- A medida habitual de eficiência é a velocidade, isto é, quanto tempo um algoritmo leva para produzir seu resultado. Porém, existem alguns problemas para os quais não se conhece nenhuma solução eficiente. Um subconjunto desses problemas denomina-se NP-Completo.

# Algoritmo

- Por que os problemas NP-Completo são interessantes?
- Embora não tenha sido encontrado nenhum algoritmo eficiente para um problema NP-Completo, ninguém jamais provou que não é possível existir um algoritmo eficiente para esse fim.

# Algoritmo

- O conjunto de problemas NP-Completo tem a propriedade notável de que, se existe um algoritmo eficiente para qualquer um deles, então existem algoritmos eficientes para todos.
  - Ex.: Caixeiro Viajante.

# Algoritmo

- Os computadores podem ser rápidos, mas não são infinitamente rápidos. A memória pode ser de baixo custo, mas não é gratuita. Assim, o tempo de computação é um recurso limitado bem como o espaço na memória.
- Esses recursos devem ser usados de forma sensata e algoritmos eficientes em termos de tempo ou espaço ajudarão nesse sentido.

# Algoritmo

- Eficiência:
  - Algoritmos criados para resolver o mesmo problema muitas vezes diferem de forma drástica em sua eficiência.
  - Ex.: ordenação por inserção leva um tempo aproximadamente igual a  $c_1 n^2$  para ordenar  $n$  itens, em que  $c_1$  é uma constante que não depende de  $n$ .
  - Ex.: ordenação por intercalação leva um tempo aproximadamente igual  $c_2 n \log(n)$



# Algoritmo

- A ordenação por inserção normalmente tem um fator constante menor que a ordenação por intercalação e assim  $c_1 < c_2$ .
- Os fatores constantes podem ser muito menos significativos no tempo de execução que a dependência do tamanho da entrada  $n$ .

# Algoritmo

- Ex.: O problema consiste em ordenar um arranjo de um milhão de números
- Suponha que o computador A seja o mais rápido e execute a ordenação por inserção, e o computador B, mais lento, execute a ordenação por intercalação.

# Algoritmo

- O computador A executa um bilhão de instruções por segundo e o computador B executa apenas dez milhões de instruções por segundo; assim o computador A é 100 vezes mais rápido que o computador B em capacidade bruta de computação.

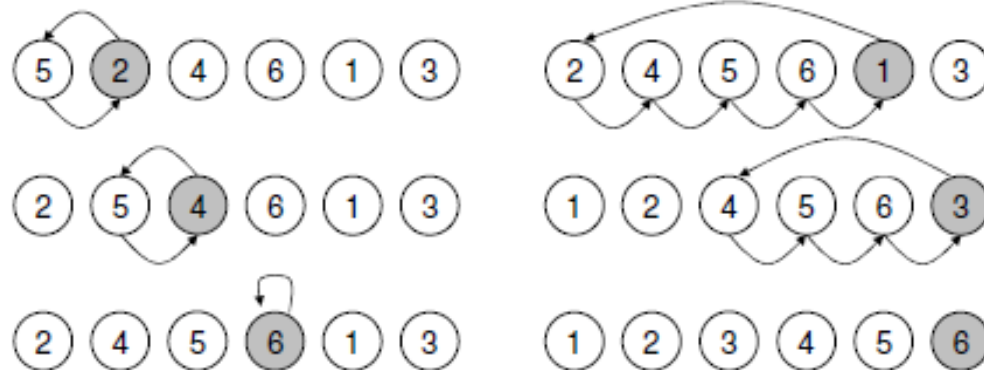
# Algoritmo

- A codificação por inserção possui  $2n^2$  instruções para ordenar  $n$  números.
- A ordenação por intercalação tem um código de  $50n \log(n)$  instruções.
- $A \rightarrow 2 \times (10^6)^2 / (10^9)$  instruções-segundo = 2000 segundos
- $B \rightarrow 50 \times 10^6 \log_2(10^6) / 10^7 = 100$  segundos

# Algoritmo

## Ordenação por Inserção

```
para j ← 2 até tamanho[A] faça  
    chave ← A[j];  
    i ← j - 1;  
    enquanto i > 0 e A[i] > chave faça  
        A[i + 1] ← A[i];  
        i ← i - 1;  
    A[i + 1] ← chave;
```



# Algoritmo

- Análise do Algoritmo

Algoritmo de Inserção (A)	<i>custo</i>	<i>vezes</i>
<b>for</b> j = 2 <b>to</b> comprimento de A	$C_1$	$n$
chave = A[j];	$C_2$	$n - 1$
i = j - 1;	$C_3$	$n - 1$
<b>while</b> i > 0 e A[i] > chave	$C_4$	$\sum_{j=2}^n t_j$
A[i + 1] = A[i];	$C_5$	$\sum_{j=2}^n (t_j - 1)$
i = i - 1;	$C_6$	$\sum_{j=2}^n (t_j - 1)$
A[i+1] = chave;	$C_7$	$n - 1$

# Algoritmo

- Melhor caso?
- Pior caso?

# Algoritmo

- Análise do algoritmo no melhor caso

$$T(n) = c_1 \cdot n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^n t_j + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_7(n-1)$$

$$T(n) = c_1 \cdot n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$$

- Complexidade:  $\Omega(n)$



# Algoritmo

- Análise do algoritmo no pior caso

$$T(n) = c_1 \cdot n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^n t_j + c_5 \sum_{j=2}^n (t_j - 1) + c_6 \sum_{j=2}^n (t_j - 1) + c_7(n-1)$$

Sendo:

$$\sum_{j=2}^n t_j = [n(n+1)/2] - 1 \quad \text{e} \quad \sum_{j=2}^n (t_j - 1) = n(n-1)/2$$

Logo temos:

$$T(n) = C_1 \cdot n + C_2 \cdot (n-1) + C_3 \cdot (n-1) + C_4 \cdot ((n(n+1)/2) - 1) + C_5 \cdot (n(n-1)/2) + C_6 \cdot (n(n-1)/2) + C_7 \cdot (n-1)$$

# Algoritmo

- Análise do algoritmo no pior caso

$$T(n) = (C_4/2 + c_5/2 + c_6/2) n^2 + (c_1 + c_2 + c_3 + c_4/2 - c_5/2 - c_6/2 + c_7) n - (c_2 + c_3 + c_4 + c_7)$$

- Complexidade:  $O(n^2)$

# Slides baseados no livro

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, **Algoritmos**, Campus, 3ª edição, 2012.