

# Estruturas de Dados 2

Prof. Silvia Brandão

2024.1

# Bora lá brincar?...

The screenshot shows the Visualgo website (visualgo.net/en) in a web browser. The browser's address bar shows the URL. The website has a dark header with a 'LOGIN' button. Below the header, there's a 'Do You Know?' section with a 'Next Random Tip' button. The main content area features the Visualgo logo and tagline, a search bar, and a 'NUS Computing' logo. Below this, there's a 'Featured story' section. The bottom of the page displays a row of five interactive visualizations: 'Sorting' (a bar chart), 'Bitmask' (a grid with 'AND' operation), 'Linked List' (a diagram of a linked list), 'Binary Heap' (a tree diagram), and 'Hash Table' (a diagram of a hash table). Each visualization has a 'Training' button. The Windows taskbar is visible at the bottom, showing the time as 19:36 on 12/03/2024.

Do You Know? Next Random Tip

To compare 2 related algorithms, e.g., *Kruskal's vs Prim's* on the same graph, or 2 related operations of the same data structure, e.g., visualizing *Binary (Max) Heap* as a Binary Tree or as a Compact Array, open 2 VisuAlgo pages in 2 windows and juxtapose them. Click [here](#) to see the screenshot. This juxtaposition technique can be used anytime you want to compare two similar data structures or algorithms.

**VISUALGO**.NET/EN  
visualising data structures and algorithms through animation

Search...

**NUS** Computing  
National University of Singapore

Featured story: Visualizing Algorithms with a Click

**Optiver**

VisuAlgo project is funded by Optiver for 2023-2024. We now open VisuAlgo account registration to every Computer Science students/teachers worldwide and have started various upgrading (sub)-projects.

Sorting Training Bitmask Training Linked List Training Binary Heap Training Hash Table Training

<https://visualgo.net/en/sorting>

# Método *Insertion Sort* (ordenação por inserção)

- ▶ Também conhecido como ordenação por inserção direta.
  - Similar a ordenação de cartas de baralho com as mãos
  - Pegue uma carta de cada vez e a insira em seu devido lugar, sempre deixando as cartas da mão em ordem.



- ▶ Funcionamento
  - O algoritmo percorre o array e para cada posição  $X$  verifica se o seu valor está na posição correta
    - Isso é feito andando para o começo do array a partir da posição  $X$  e movimentando uma posição para frente os valores que são maiores do que o valor da posição  $X$
    - Desse modo, teremos uma posição livre para inserir o valor da posição  $X$  em seu devido lugar

# *Insertion Sort* – algoritmo

ALGORITMO INSERÇÃO

ENTRADA: UM VETOR V COM N POSIÇÕES

SAÍDA: O VETOR V EM ORDEM CRESCENTE

PARA  $i = 1$  até  $n - 1$

    PIVO =  $V[i]$

$j = i - 1$

    ENQUANTO  $j \geq 0$  e  $V[j] > \text{PIVO}$

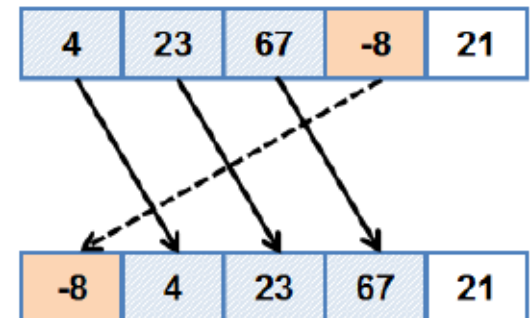
$V[j+1] = V[j]$

$j = j - 1$

$V[j+1] = \text{PIVO}$

FIM {INSERÇÃO}

**Move as cartas maiores para frente e insere na posição vaga**



# *Insertion Sort* – algoritmo

## ➤ Vantagens:

- Fácil implementação.
- Na prática, é mais eficiente que a maioria dos algoritmos de ordem quadrática, como o selection sort e o bubble sort.
- É um bom método quando se desejar adicionar poucos elementos em um arquivo já ordenado, pois seu custo é linear.
- Um dos mais rápidos algoritmos de ordenação para conjuntos pequenos de dados; **superando inclusive o quick sort**.
- Estável: não altera a ordem dos dados iguais.
- Online
  - Pode ordenar elementos a medida que os recebe (tempo real).
  - Não precisa ter todo o conjunto de dados para colocá-los em ordem.

## ➤ Desvantagens:

- Alto custo de movimentação de elementos no vetor.

# *Insertion Sort* – algoritmo

## ► Complexidade:

- Considerando um array com  $N$  elementos, o tempo de execução é:
  - $O(n)$ , melhor caso: os elementos já estão ordenados.
  - $O(n^2)$ , pior caso: os elementos estão ordenados na ordem inversa.
  - $O(n^2)$ , caso médio.

Algoritmo	Tempo		
	Melhor	Médio	Pior
Bubble sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$
Merge sort	$O(n \log_2 n)$	$O(n \log_2 n)$	$O(n \log_2 n)$
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$

# Resumo dos métodos simples de ordenação

- ▶ Não existe um algoritmo de ordenação que seja o melhor em todas as possíveis situações.
- ▶ Para escolher o algoritmo mais adequado para uma dada situação, precisamos **verificar as características específicas dos elementos que devem ser ordenados.**
- ▶ Por exemplo:
  - Se os elementos a serem ordenados forem grandes, por exemplo, registros acadêmicos de alunos, o Selection Sort pode ser uma boa escolha, já que ele efetuará, no pior caso, muito menos trocas que o Insertion Sort ou o Bubble Sort.
  - Se os elementos a serem ordenados estiverem quase ordenados (situação relativamente comum), o Insertion Sort realizará muito menos operações (comparações e trocas) do que o Selection Sort ou o Bubble Sort.

# Bubble Sort - Análise de Complexidade

```
1 def bubbleSort(lista):  
2     n = len(lista)  
3     for i in range(n - 1, 0, -1):  
4         for j in range(i):  
5             if lista[j] > lista[j + 1]:  
6                 (lista[j], lista[j + 1]) = (lista[j + 1], lista[j])
```

- Número máximo de comparações entre elementos da lista:

$$f(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = (n-1) \frac{n}{2} = \frac{n^2 - n}{2}$$

- Número máximo de trocas entre elementos da lista:

$$f(n) = \sum_{i=1}^{n-1} \sum_{i=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = (n-1) \frac{n}{2} = \frac{n^2 - n}{2}$$

- Número mínimo de comparações entre elementos da lista:

$$f(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = (n-1) \frac{n}{2} = \frac{n^2 - n}{2}$$

- Número mínimo de trocas entre elementos da lista:

$$f(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 0 = 0$$



# Selection Sort - Análise de Complexidade

```
1 def selectionSort(lista):  
2     n = len(lista)  
3     for i in range(n - 1):  
4         minimo = indiceMenor(lista, i)  
5         (lista[i], lista[minimo]) = (lista[minimo], lista[i])
```

- Número máximo de comparações entre elementos da lista:

$$f(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} n - i - 1 = \sum_{i=1}^{n-1} i = (n-1) \frac{n}{2} = \frac{n^2 - n}{2}$$

- Número máximo de trocas entre elementos da lista:

$$f(n) = \sum_{i=0}^{n-2} 1 = n - 1$$

- Número mínimo de comparações entre elementos da lista:

$$f(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} n - i - 1 = \sum_{i=1}^{n-1} i = (n-1) \frac{n}{2} = \frac{n^2 - n}{2}$$

- Número mínimo de trocas entre elementos da lista:

$$f(n) = \sum_{i=0}^{n-2} 1 = n - 1$$

# Insertion Sort - Análise de Complexidade

```
1 def insertionSort(lista):  
2     n = len(lista)  
3     for i in range(1, n):  
4         insertion(lista, i)
```

```
1 def insertion(lista, i):  
2     aux = lista[i]  
3     j = i - 1  
4     while (j >= 0) and (lista[j] > aux):  
5         lista[j + 1] = lista[j]  
6         j = j - 1  
7     lista[j + 1] = aux
```

função insertion de forma ainda mais simples

```
1 def insertion(lista, i):  
2     j = i - 1  
3     while (j >= 0) and (lista[j] > lista[i]):  
4         j = j - 1  
5     lista[j + 1:i + 1] = [lista[i]] + lista[j + 1:i]
```

- Número máximo de comparações entre elementos da lista:

$$f(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1 = \sum_{i=1}^{n-1} i = (n-1) \frac{n}{2} = \frac{n^2 - n}{2}$$

- Número máximo de modificações realizadas na lista:

$$f(n) = \sum_{i=1}^{n-1} \sum_{j=0}^i 1 = \sum_{i=1}^{n-1} (i+1) = (n-1) \frac{n+2}{2} = \frac{n^2 + n}{2} - 1$$

- Número mínimo de comparações entre elementos da lista:

$$f(n) = \sum_{i=1}^{n-1} 1 = n - 1$$

- Número mínimo de modificações realizadas na lista:

$$f(n) = \sum_{i=1}^{n-1} 1 = n - 1$$

# Prática

- ▶ Implemente o método *Insertion Sort* junto aos códigos anteriores. Compare o tempo de cada ordenação para cada estrutura de dados.

## Próxima aula

- ▶ Avaliação contínua da implementação dos métodos de ordenação simples (Bubble, Selection e Insertion).
- 