

# Complexidade de Algoritmos

Paulino Ng

2020-05-29

# Plano da aula

Esta aula apresenta alguns algoritmos de ordenação e uma análise de sua complexidade.

1. Problema de ordenação.
2. Ordenação por seleção
3. Ordenação por inserção
4. *Mergesort*, ordenação por intercalação
5. *Quicksort*

# Problema da Ordenação

*Entrada:*

*Saída:*

# Ordenação por seleção

Procedimento ordena\_selecao(A,n)

Entradas:

A: um vetor.

n: número de elementos de A a serem ordenados.

Saída:

Os elementos de A ordenados em ordem não decrescente

1. Para  $i = 0$  até  $n-2$ :

A. Faça menor ser o índice do menor elemento do  
subvetor  $A[i..n-1]$

B. Troque  $A[i]$  com  $A[\text{menor}]$

*Complexidade:  $\Theta(n^2)$ , por que?*

## Análise do ordena\_selecao()

- ▶ temos dois laços, um dentro do outro:
  - ▶ o mais interno procura o índice do menor valor do subvetor  $\text{vec}[i..n-1] \rightarrow \Theta(n)$
  - ▶ o externo, depois de encontrado o menor valor do subvetor, troca o valor de  $\text{vec}[i]$  com o valor de  $\text{vec}[\text{menor}]$  para cada  $i$  do primeiro ao penúltimo  $\rightarrow (n-1) \cdot \Theta(n) = \Theta(n^2)$
- ▶ Logo, a complexidade do ordena\_selecao() é  $\Theta(n^2)$

## Código Python para implementar este algoritmo

```
def select_sort(vec,n):  
    for i in range(0,n-1):  
        menor = i  
        for j in range(i+1,n):  
            if vec[j] < vec[menor]: menor = j  
        vec[i], vec[menor] = vec[menor], vec[i]
```

# Ordenação por inserção

Procedimento ordena\_insercao(A,n)

Entradas e Saída como no ordena\_selecao()

1. Para  $i=1$  até  $n-1$ :

A. Faça chave =  $A[i]$  e  $j = i - 1$

B. Enquanto  $j > 0$  e  $A[j] > \text{chave}$  faça:

i.  $A[j+1] = A[j]$

ii.  $j = j - 1$

C.  $A[j+1] = \text{chave}$

## Análise do `insert_sort()`

- ▶ A ideia básica do funcionamento do `insert_sort()` é semelhante ao que faz um jogador de cartas que recebe as cartas uma a uma e vai colocando elas na ordem. As cartas já recebidas estão em ordem na mão do jogador, a carta nova é inserida na sua posição deslocando todas as cartas maiores para a direita de uma posição.
- ▶ O algoritmo, de novo, tem dois laços, um dentro do outro;
  - ▶ o laço externo percorre o vetor do segundo elemento para o último, isto é, executa as instruções A, B e C  $n - 1$  vezes;
  - ▶ o laço interno desloca para a direita os valores maiores do que  $A[i]$  para a direita para abrir espaço para colocar o  $A[i]$  na sua posição correta (instrução C)
- ▶ Se a chave (valor de  $A[i]$ ) for maior do que o valor mais a direita dos elementos já ordenados em ordem crescente, então, não há a necessidade de deslocar nenhum elemento já ordenado. Isto quer dizer que se o vetor já estiver ordenado, o laço interno não precisa executar as suas instruções e a complexidade do algoritmo é  $\mathcal{O}(n)$  (melhor caso).
- ▶ Se o vetor estiver ordenado na ordem decrescente, o laço



## Código Python para implementar este algoritmo

```
def insert_sort(vec,n):  
    for i in range(1,n-1):  
        chave = vec[i]  
        j = i - 1  
        while j > 0 and A[j] > chave:  
            vec[j+1] = vec[j]  
            j = j - 1  
        vec[j+1] = chave
```

# Mergesort

# Quicksort