

Liniker Jardel de Oliveira – **RGM:** 1782703-5

**ANÁLISE DE DESEMPENHO DE ALGORITMOS DE ORDENAÇÃO POR COMPARAÇÃO**

Ciência da Computação – 6 Semestre.

Teoria dos Grafos

Prof.º Leandro B. Marques

Salto/SP

2019

**INTRODUÇÃO**

Este trabalho tem como objetivo analisar o desempenho de seis algoritmos distintos de ordenação. A análise será dividida em duas partes: Na primeira parte, serão executados os métodos de ordem de complexidade O(n²), como o Bubble Sort, Selection Sort, Insertion Sort. Na segunda parte, serão feitas as análises sobre os métodos de complexidade logarítmicos O(log n), como o ShellSort, Merge Sort, QuickSort. A linguagem utilizada para a codificação dos algoritmos foi implementada em Python 3.7 cujo código se encontra no repositório do GitHub. <https://github.com/linikerunk/Algoritmo-Ordenacao-Python/blob/master/Principal_Ordenacoes.py>

A criação dos gráficos foi realizada com a linguagem Python com a biblioteca matplotlib e os dados foram registrados em Excel.

As configurações de máquinas utilizadas para os devidos testem foram.

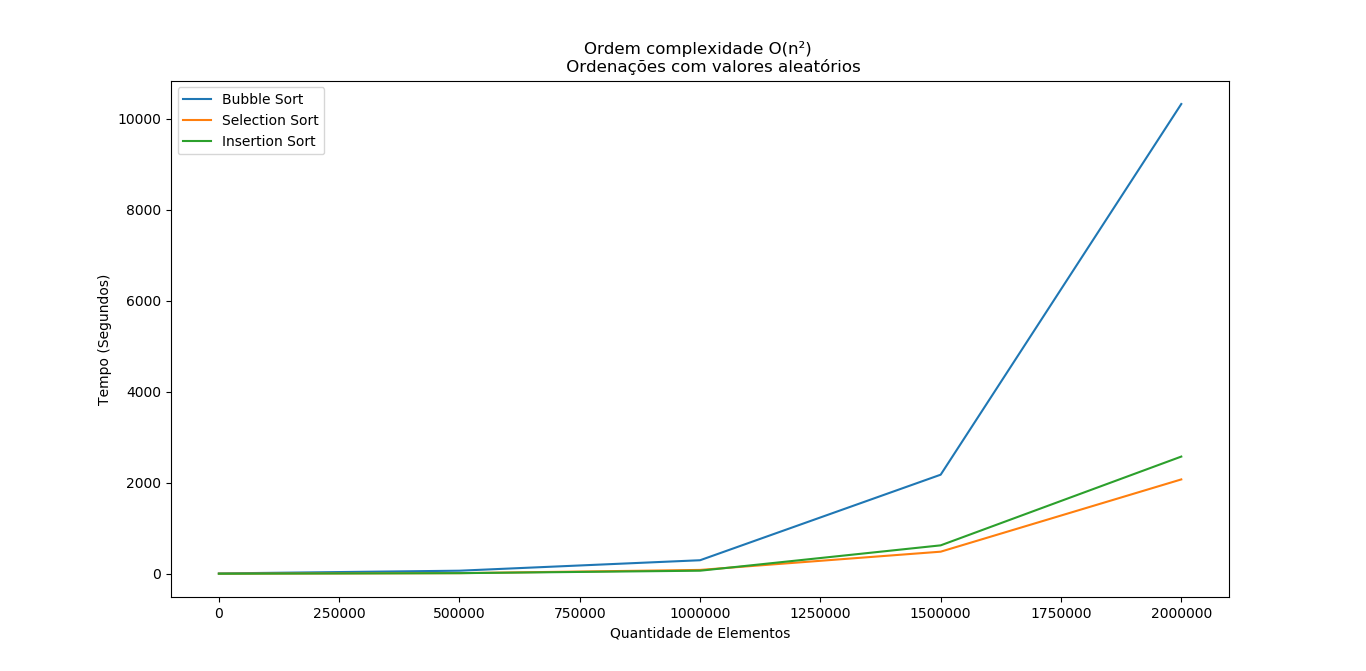


**Figura 1:** Configurações do sistema.

**Fonte:** Próprio autor.

**Gráficos de desempenho dos métodos**

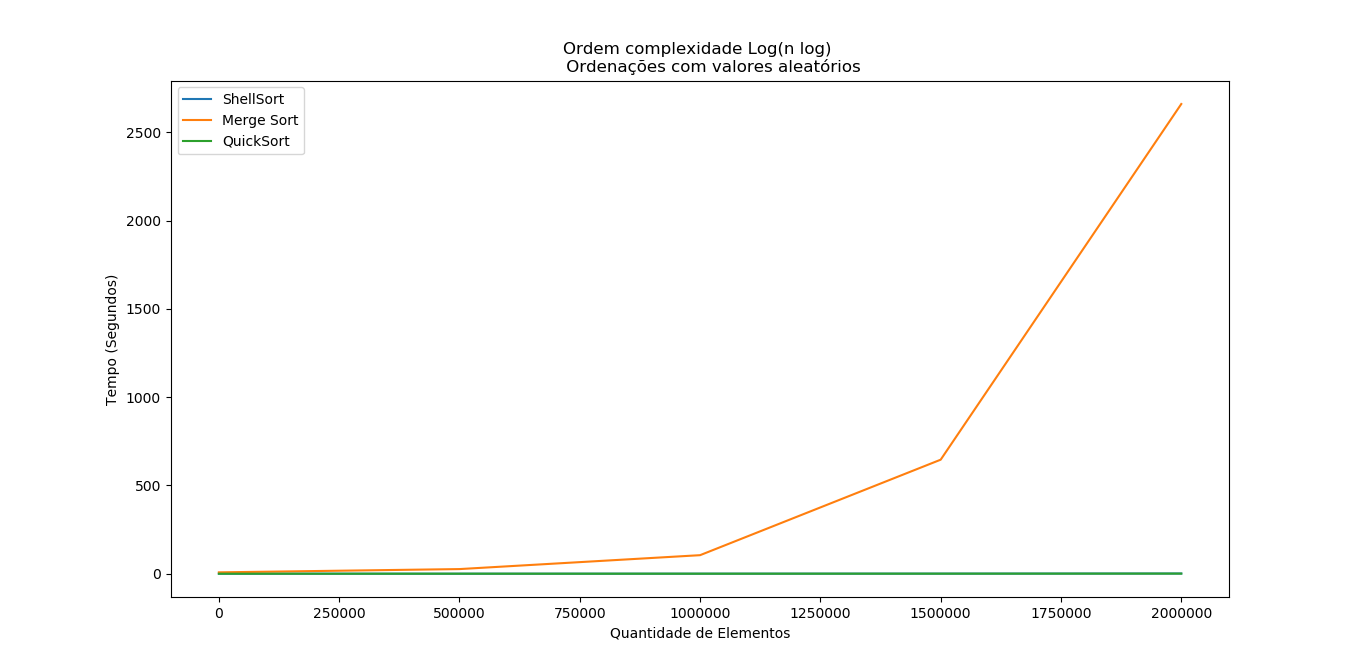
A seguir, é possível observar o desempenho de cada ordenação quadrática, com elementos gerados de forma aleatória, conforme a figura 2.

****

**Figura 2:** Ordenações Quadráticas Aleatórias

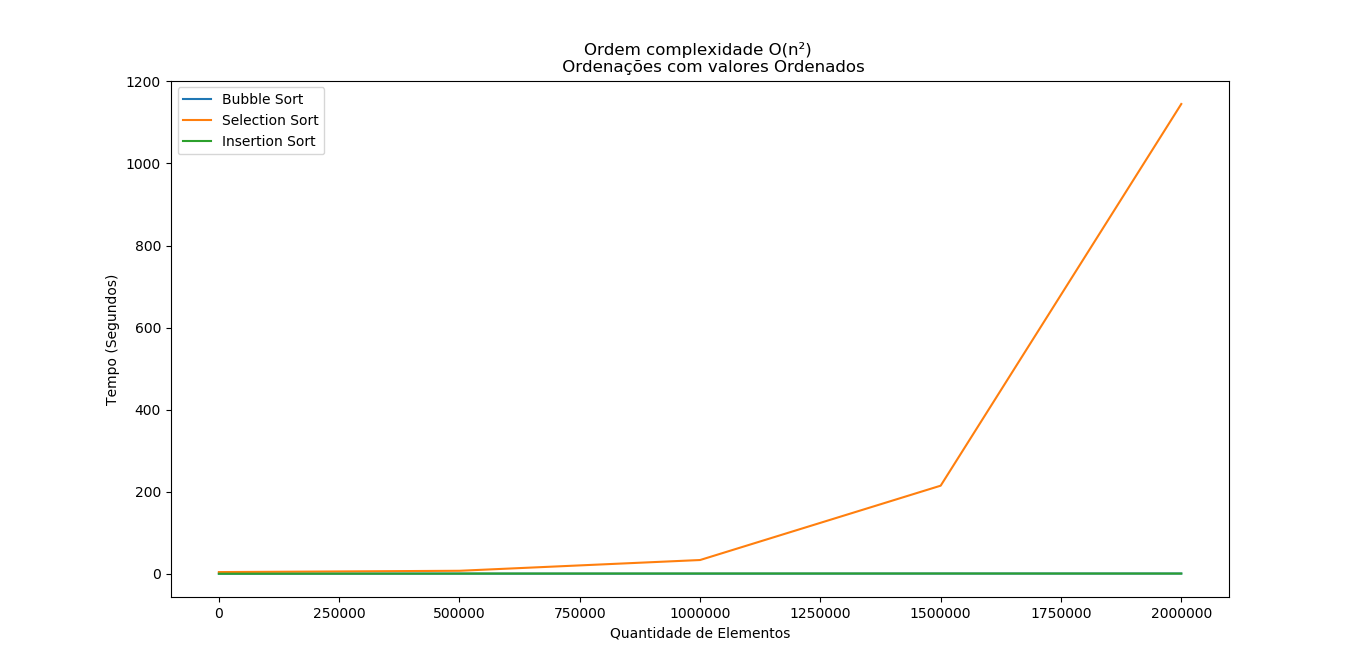
**Fonte:** Próprio autor.

A figura 3 representa as ordenações logarítmicas, com elementos gerados de forma aleatória.

****

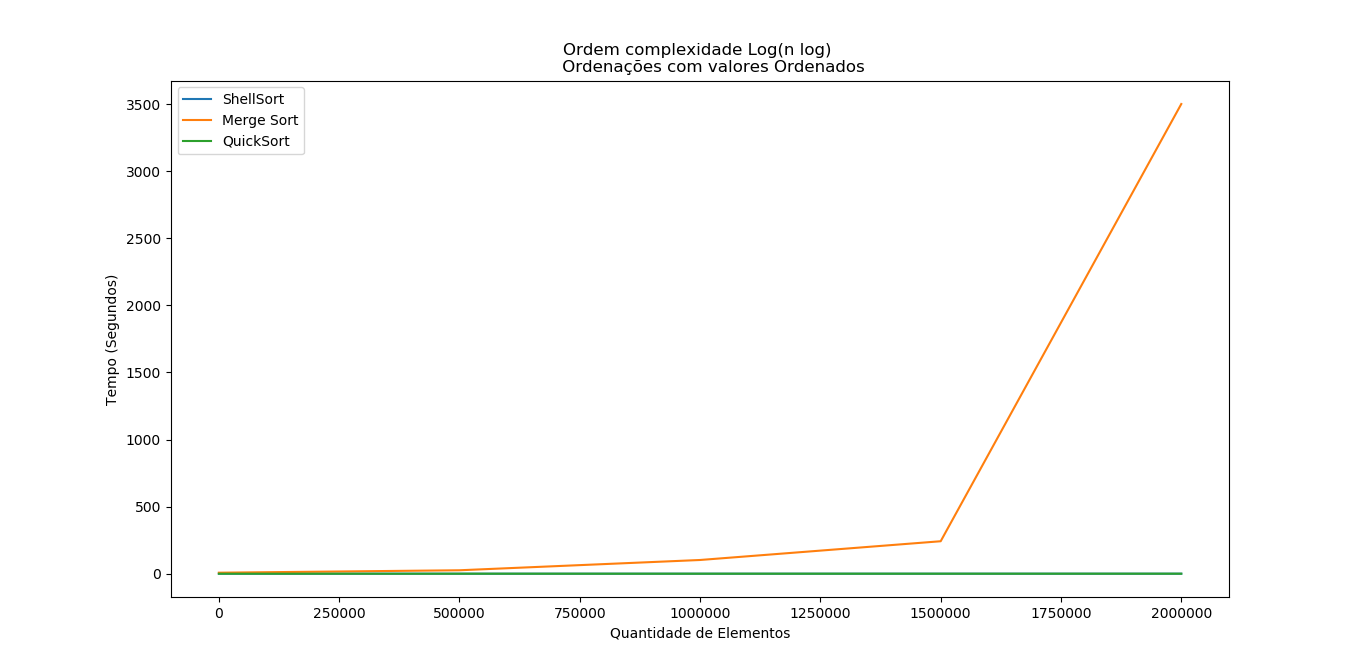
**Figura 3:** Ordenações Logarítmicas Aleatórias

**Fonte:** Próprio autor.

As figuras 4 e 5 representam as ordenações de elementos ordenados pelo índice, nas complexidades quadrática e logarítmica, respectivamente.**

**Figura 4:** Complexidade Quadráticas Ordenadas

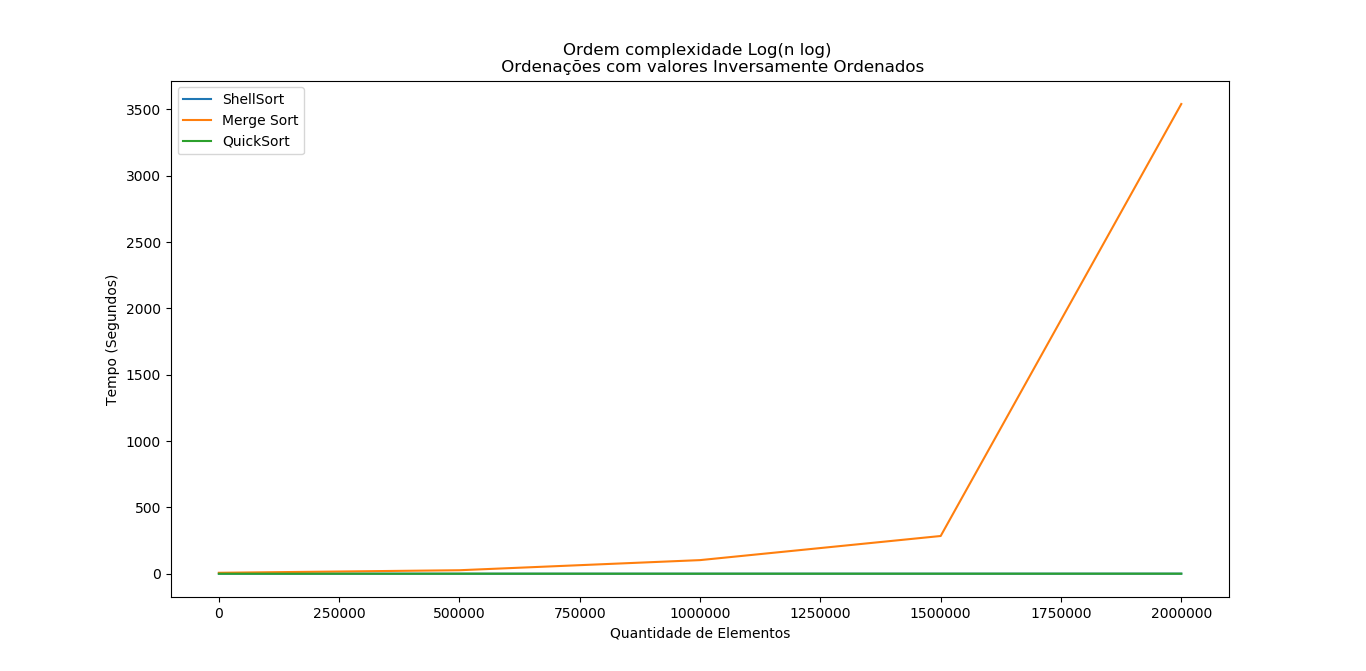
**Fonte:** Próprio autor.

****

**Figura 5:** Complexidade Logarítmicas Ordenadas

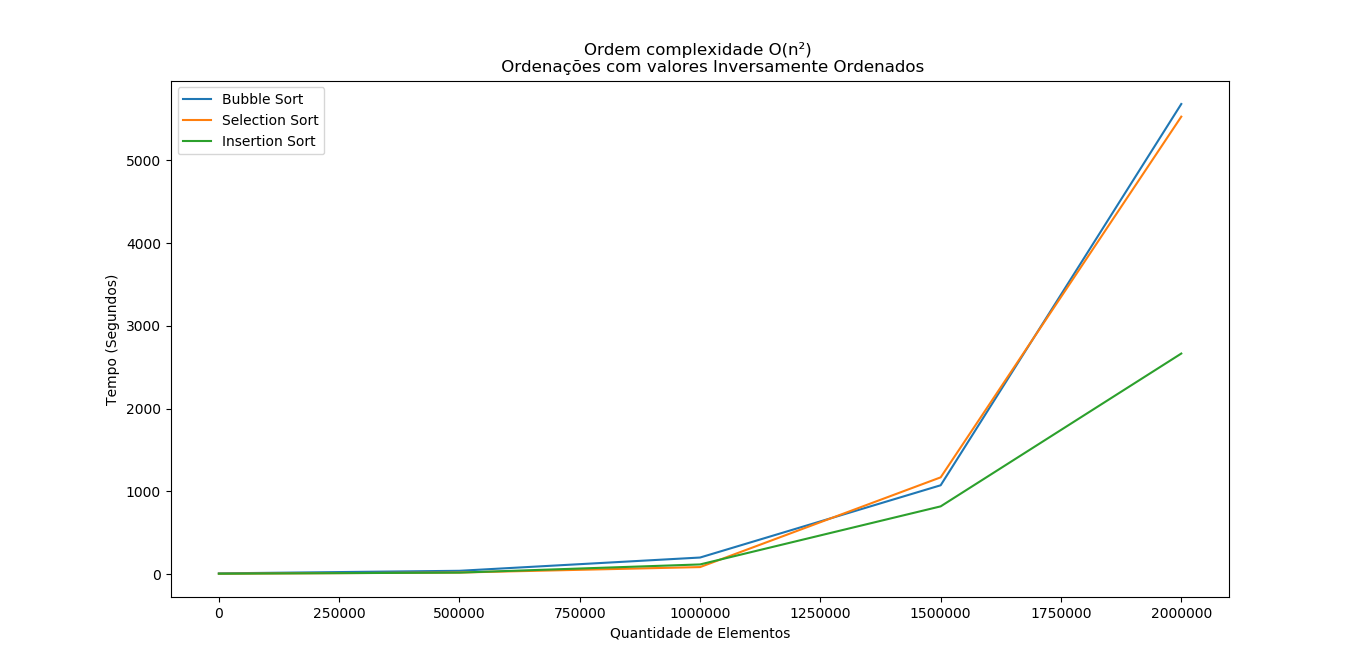
**Fonte:** Próprio autor.

Por fim, tem-se os gráficos das ordenações inversamente ordenadas, sendo a figura 6 de complexidade logarítmica e a figura 7 de complexidade quadrática.

**

**Figura 6:** Complexidade Logarítmica Inversamente Ordenada

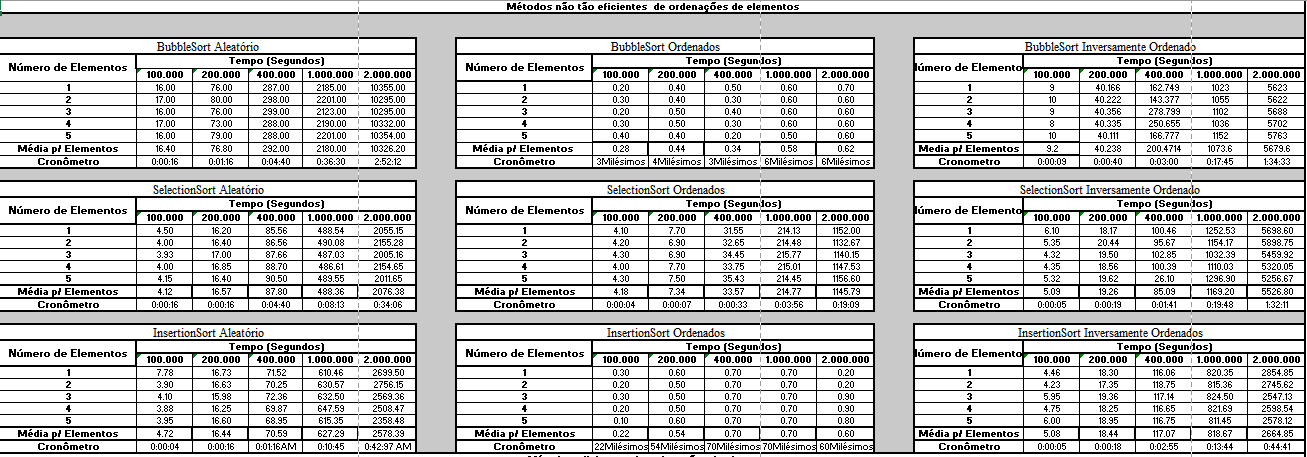
**Fonte:** Próprio autor.

****

**Figura 7:** Complexidade Quadrática Inversamente Ordenada

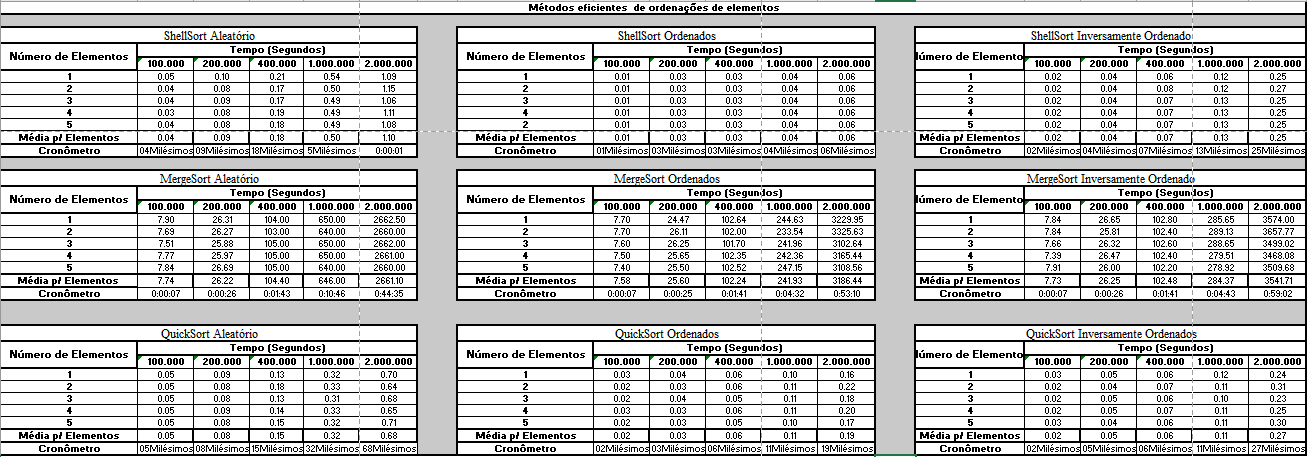
**Fonte:** Próprio autor.

**Resultados dos métodos de ordenações**

****

**Figura 8:** Resultado dos métodos não eficientes de ordenação

**Fonte:** Próprio autor.

****

**Figura 9:** Resultado dos métodos eficientes de ordenação

**Fonte:** Próprio autor.

**Código Fonte (Python)**

#ALGORITMO DE ORDENAÇÕES PARA O TRABALHO DE COMPUTABILIDADE E COMPLEXIDADE DE ALGORITMOS

# AUTHOR: LINIKER OLIVEIRA  - CEUNSP (SALTO)

import random

import time

import sys

import sys

x=1500

sys.setrecursionlimit(x)

def BubbleSort(*vetor*):

    for final in range(len(vetor), 0, -1):

        troca = False

        for atual in range(0, final - 1):

            if vetor[atual] > vetor[atual + 1]:

                vetor[atual + 1], vetor[atual] = vetor[atual], vetor[atual + 1]

                troca = True

        if not troca:

            break

def SelectionSort(*vetor*):

    for indice in range(0, len(vetor)):

        min\_indice = indice

        for direita in range(indice + 1, len(vetor)):

            if vetor[direita] < vetor[min\_indice]: # pois precisamos encontrar o menor elemento da direita

                min\_indice =  direita # se ele for menor ele virá o menor indice

        vetor[indice], vetor[min\_indice] = vetor[min\_indice], vetor[indice] # Aqui eu faço a troca...

def InsertionSort(*vetor*):

    for posicao in range(0, len(vetor)):

        elemento\_atual = vetor[posicao]

        while posicao > 0 and vetor[posicao - 1] > elemento\_atual:

            vetor[posicao] = vetor[posicao - 1]

            posicao -= 1

        vetor[posicao] = elemento\_atual

def MergeSort(*array*):

    ordena\_metade(array, 0, len(array) - 1)

def ordena\_metade(*array*, *inicio*, *fim*):

    if inicio >= fim:

        return

    meio = (inicio + fim) // 2 # divisão de Inteiros ...

    ordena\_metade(array, inicio, meio)# do inicio até o meio...

    ordena\_metade(array, meio + 1, fim)# do meio + 1 até o final...

    merge(array, inicio, fim)

def merge(*array*, *inicio*, *fim*):

    array[inicio: fim + 1] = sorted(array[inicio: fim + 1])

def ShellSort(*lista*):

    sublista = len(lista)//2

    while sublista > 0:

      for posicao\_inicial in range(sublista):

        faz\_InsertionSort(lista, posicao\_inicial, sublista)

      sublista = sublista // 2

def faz\_InsertionSort(*nlist*,*start*,*gap*):

    for i in range(start+gap,len(nlist),gap):

        valor\_atual = nlist[i]

        posicao = i

        while posicao>=gap and nlist[posicao-gap]>valor\_atual:

            nlist[posicao]=nlist[posicao-gap]

            posicao = posicao-gap

        nlist[posicao]=valor\_atual

def BucketSort(*array*):

    k = max(array)

    bucket = [0]\* (k+1)

    for j in range(len(array)):

        bucket[array[j]] = bucket[array[j]] + 1

    indice = 0

    for i in range(k+1):

        for j in range(bucket[i]):

            array[indice] = i

            indice = indice + 1

def QuickSort(*array*):

    less = []

    equal = []

    greater = []

    # print array

    if len(array) <= 1:

        return array

    else:

        pivot = array[0]

        for x in array:

            if x < pivot:

                less.append(x)

            elif x > pivot:

                greater.append(x)

            else:

                equal.append(x)

        less = QuickSort(less)

        greater = QuickSort(greater)

        return greater + equal + less

vetor = []

numeros\_gerados = False

opcao = True

numeros\_dados = *int*(input(" Digite o tamanho de dados que vc quer ordenar : "))

while (opcao != 0):

    print('\t','\*' \* 20, '\

    \t MENU','\t','\*'\*20,)

    opcao = *int*(input('\t 0 - Sair  \n \

    \t 1 - Imprimir Vetor  \n\

    \t 2 - Gerar Vetor  \n \

    \t 3 - Organizar Vetor Ordem Decrescente \n \

    \t 4 - Organizar o Vetor Ordem Crescente \n \

    \t 5 - BubbleSort \n \

    \t 6 - SelectionSort \n \

    \t 7 - InsertionSort \n \

    \t 8 - MergeSort  \n \

    \t 9 - ShellSort \n \

    \t 10 - BucketSort \n \

    \t 11 - QuickSort \n \

    \t Opção : '))

    if *int*(opcao) == 0:

        exit(1)

    elif *int*(opcao) == 1:

        if numeros\_gerados == True:

            print('\n IMPRIMIR O VETOR \n')

            for v in vetor:

                print("Vetor ["+ *str*(v) +"]")

        else:

            print("Antes de imprimir o vetor, gere os números aleatórios através da opção '2'")

    elif *int*(opcao) == 2:

        vetor = [random.randint(0, numeros\_dados) for i in range(numeros\_dados)]

        numeros\_gerados = True

    elif *int*(opcao) == 3:

        if numeros\_gerados == True:

            vetor = (sorted(vetor, *reverse* = True))

        else:

            print(' Erro. Primeiro gere o vetor.')

    elif *int*(opcao) == 4:

        if numeros\_gerados == True:

            vetor = (sorted(vetor))

        else:

            print(' Erro. Primeiro gere o vetor.')

    elif *int*(opcao) == 5:

        if numeros\_gerados == True:

            print("\t [Método BubbleSort] Ordenando Vetores \t")

            inicio = time.time()

            BubbleSort(vetor)

            fim = time.time()

            tempototal = fim - inicio

            print(f"Início = {inicio:.2f} Segundos")

            print(f"Fim = {fim:.2f} Segundos")

            print(f"Tempo Total = {tempototal:.2f}")

            print(f" A Ordenação levou {tempototal:.2f} Segundos")

        else:

            print(' Erro. Primeiro gere o vetor.')

    elif *int*(opcao) == 6:

        if numeros\_gerados == True:

            print("\t [Método SelectionSort] Ordenando Vetores \t")

            inicio = time.time()

            SelectionSort(vetor)

            fim = time.time()

            tempototal = fim - inicio

            print(f"Início = {inicio:.2f} Segundos")

            print(f"Fim = {fim:.2f} Segundos")

            print(f"Tempo Total = {tempototal:.2f}")

            print(f" A Ordenação levou {tempototal:.2f} Segundos")

        else:

            print(' Erro. Primeiro gere o vetor.')

    elif *int*(opcao) == 7:

        if numeros\_gerados == True:

            print("\t [Método InsertionSort] Ordenando Vetores \t")

            inicio = time.time()

            InsertionSort(vetor)

            fim = time.time()

            tempototal = fim - inicio

            print(f"Início = {inicio:.2f} Segundos")

            print(f"Fim = {fim:.2f} Segundos")

            print(f"Tempo Total = {tempototal:.2f}")

            print(f" A Ordenação levou {tempototal:.2f} Segundos")

        else:

            print(' Erro. Primeiro gere o vetor.')

    elif *int*(opcao) == 8:

        if numeros\_gerados == True:

            print("\t [Método MergeSort] Ordenando Vetores \t")

            inicio = time.time()

            MergeSort(vetor)

            fim = time.time()

            tempototal = fim - inicio

            print(f"Início = {inicio:.2f} Segundos")

            print(f"Fim = {fim:.2f} Segundos")

            print(f"Tempo Total = {tempototal:.2f}")

            print(f" A Ordenação levou {tempototal:.2f} Segundos")

        else:

            print(' Erro. Primeiro gere o vetor.')

    elif *int*(opcao) == 9:

        if numeros\_gerados == True:

            print("\t [Método ShellSort] Ordenando Vetores \t")

            inicio = time.time()

            ShellSort(vetor)

            fim = time.time()

            tempototal = fim - inicio

            print(f"Início = {inicio:.2f} Segundos")

            print(f"Fim = {fim:.2f} Segundos")

            print(f"Tempo Total = {tempototal:.2f}")

            print(f" A Ordenação levou {tempototal:.2f} Segundos")

        else:

            print(' Erro. Primeiro gere o vetor.')

    elif *int*(opcao) == 10:

        if numeros\_gerados == True:

            print("\t [Método BucketSort] Ordenando Vetores \t")

            inicio = time.time()

            BucketSort(vetor)

            fim = time.time()

            tempototal = fim - inicio

            print(f"Início = {inicio:.2f} Segundos")

            print(f"Fim = {fim:.2f} Segundos")

            print(f"Tempo Total = {tempototal:.2f}")

            print(f" A Ordenação levou {tempototal:.2f} Segundos")

        else:

            print(' Erro. Primeiro gere o vetor.')

    elif *int*(opcao) == 11:

        if numeros\_gerados == True:

            print("\t [Método QuickSort] Ordenando Vetores \t")

            inicio = time.time()

            QuickSort(vetor)

            fim = time.time()

            tempototal = fim - inicio

            print(f"Início = {inicio:.2f} Segundos")

            print(f"Fim = {fim:.2f} Segundos")

            print(f"Tempo Total = {tempototal:.2f}")

            print(f" A Ordenação levou {tempototal:.2f} Segundos")

        else:

            print(' Erro. Primeiro gere o vetor.')

    else:

        print("Opção inválida, digite novamente...")