CEUNSP
Centro Universitário
N. Sra. do Patrocínio

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 $O(n^2)$, 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.

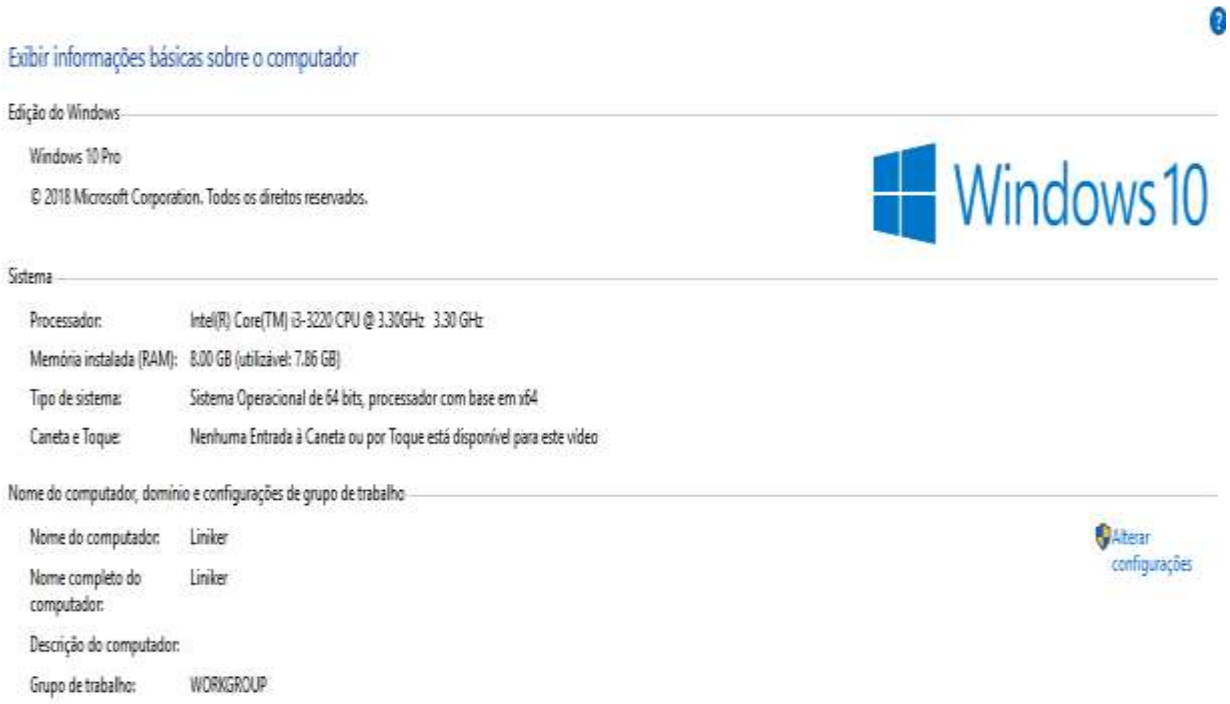


Figura 1: Configurações do sistema.
Fonte: Próprio autor.

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.

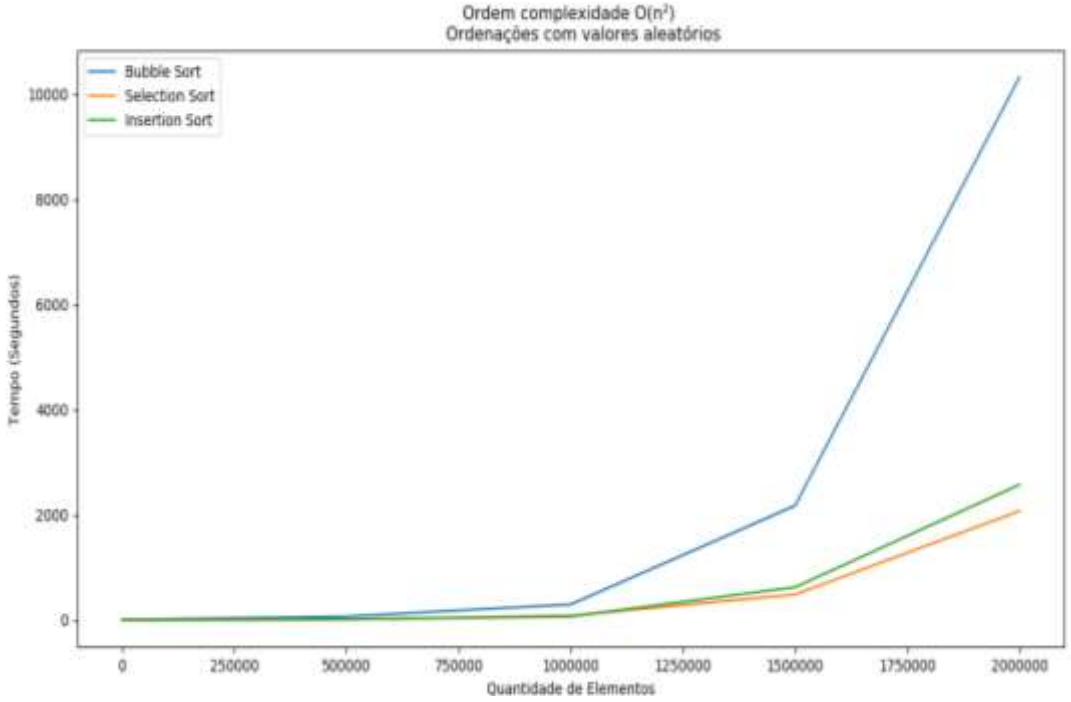


Figura 2: Ordenações Quadráticas Aleatórias
Fonte: Próprio autor.

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

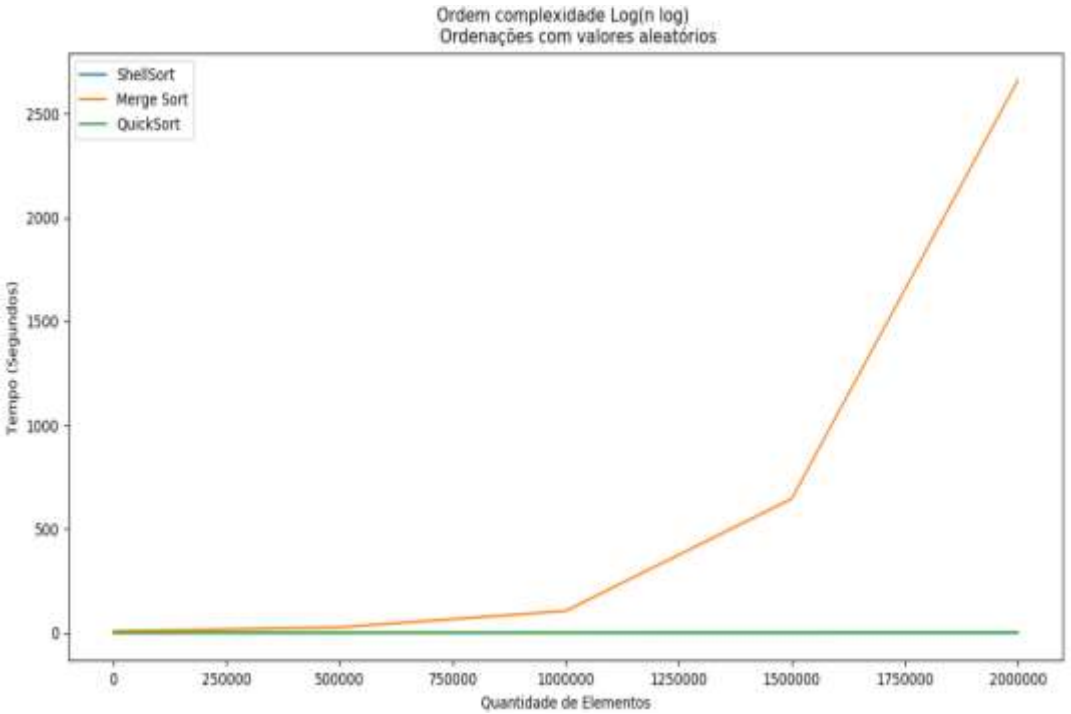


Figura 3: Ordenações Logarítmicas Aleatórias
Fonte: Próprio autor.

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

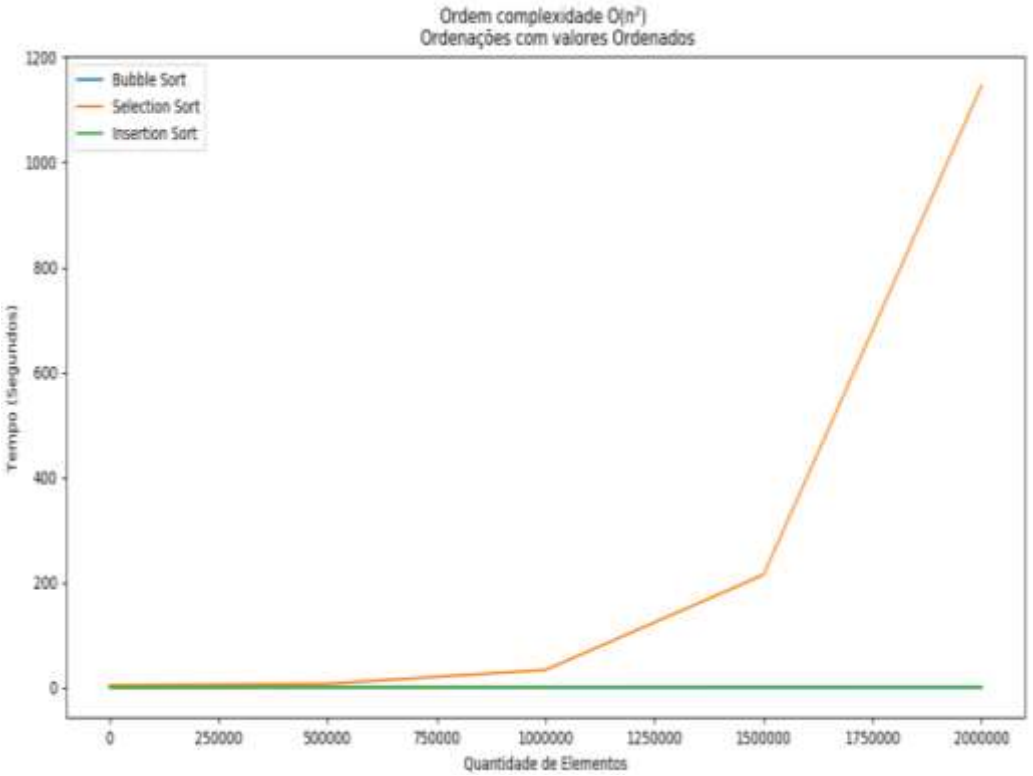


Figura 4: Complexidade Quadráticas Ordenadas
Fonte: Próprio autor.

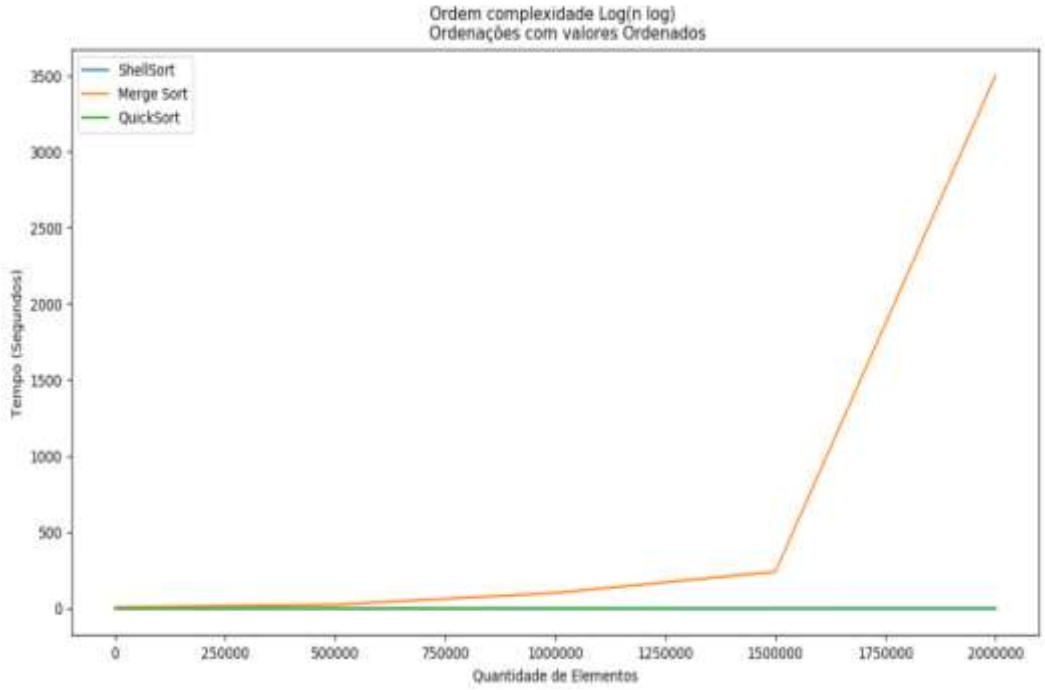


Figura 5: Complexidade Logarítmicas Ordenadas
Fonte: Próprio autor.

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

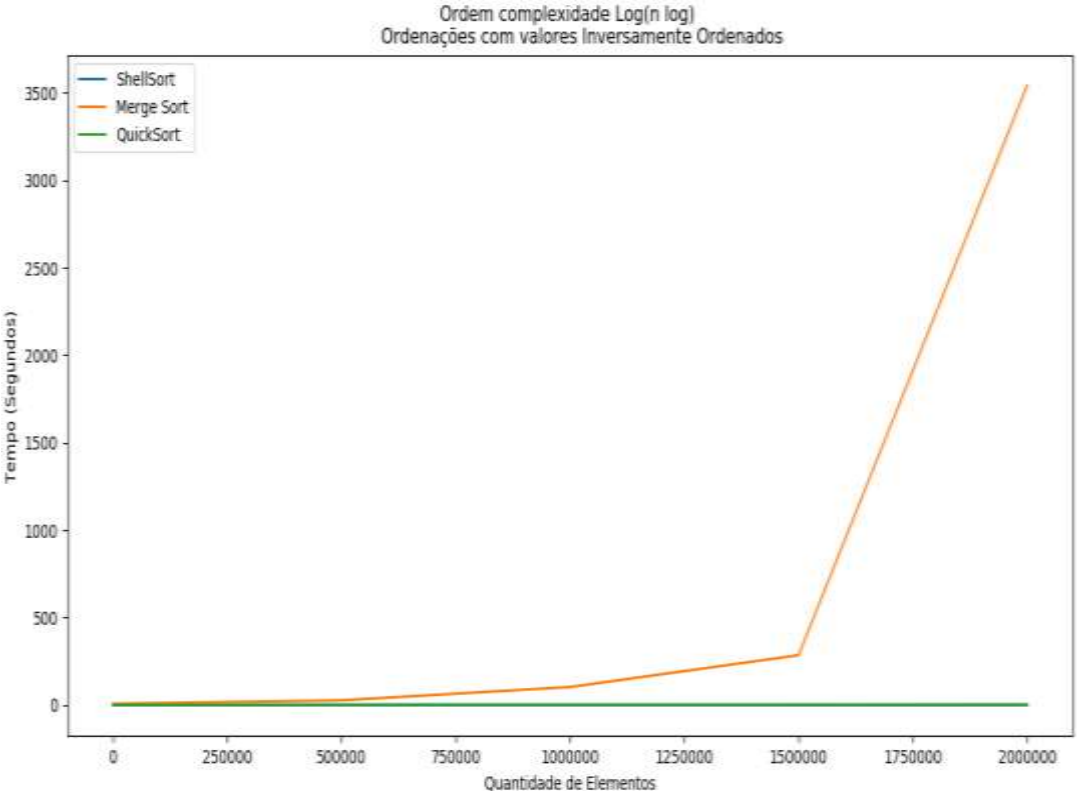


Figura 6: Complexidade Logarítmica Inversamente Ordenada
Fonte: Próprio autor.

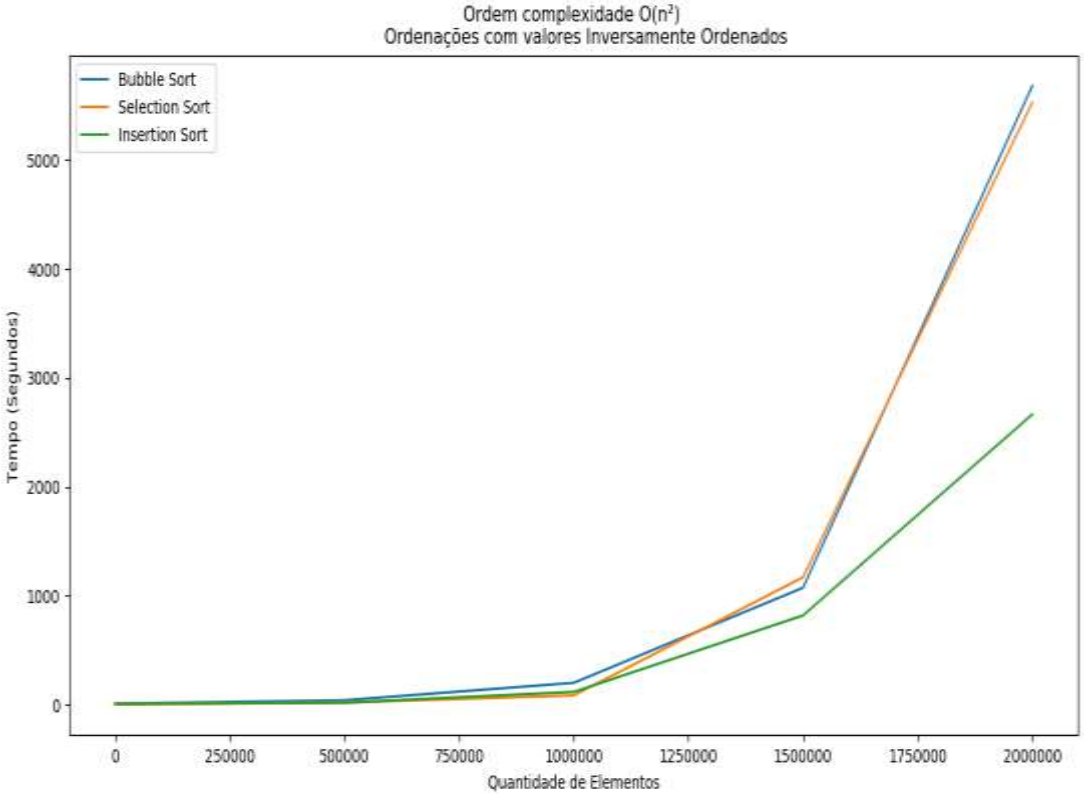
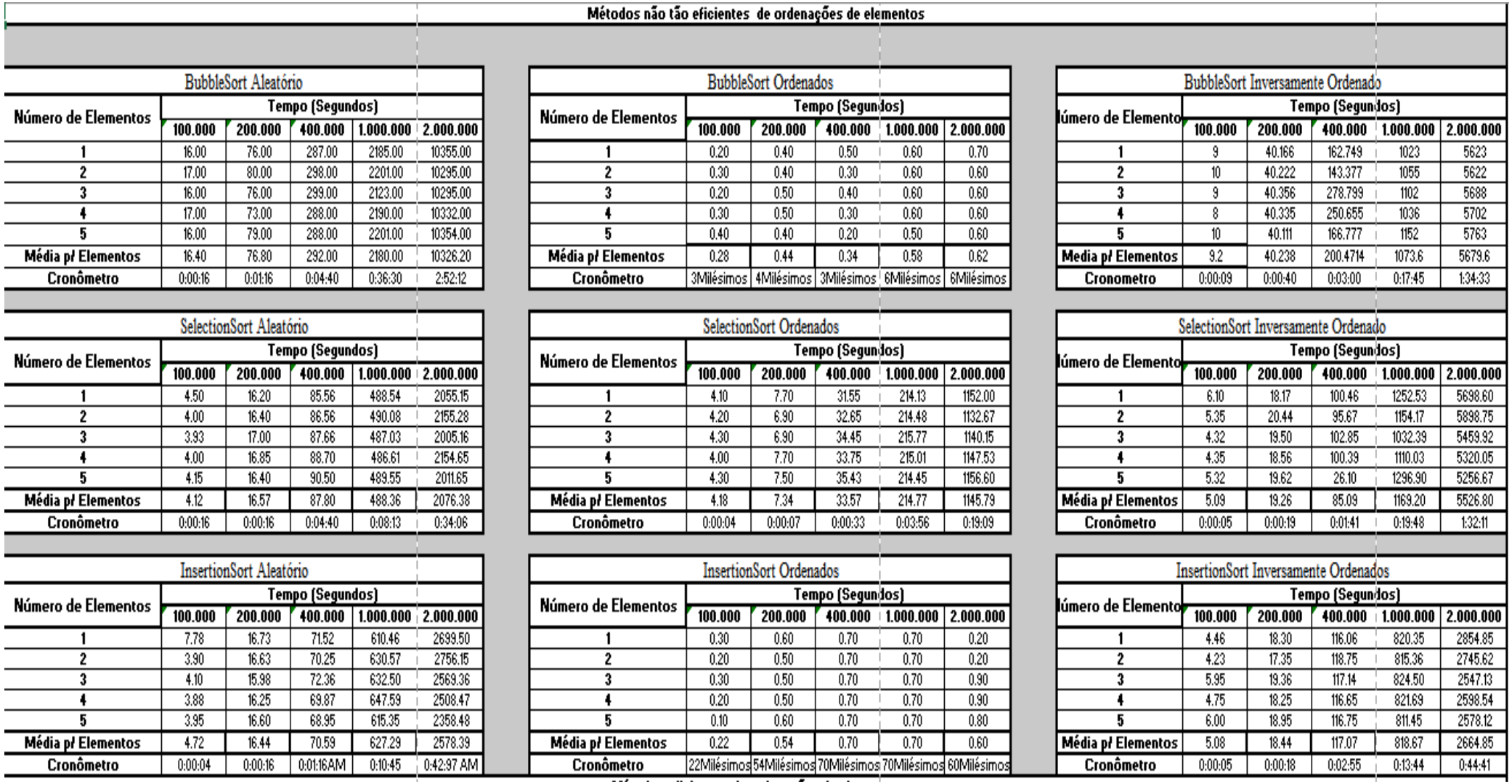


Figura 7: Complexidade Quadrática Inversamente Ordenada
Fonte: Próprio autor.

Resultados dos métodos de ordenações



Métodos não tão eficientes de ordenações de elementos

BubbleSort Aleatório

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	16.00	76.00	287.00	2185.00	10355.00
2	17.00	80.00	298.00	2201.00	10295.00
3	16.00	76.00	299.00	2123.00	10295.00
4	17.00	73.00	288.00	2190.00	10332.00
5	16.00	79.00	288.00	2201.00	10354.00
Média p/ Elementos	16.40	76.80	292.00	2180.00	10326.20
Cronômetro	0:00:16	0:01:16	0:04:40	0:36:30	2:52:12

BubbleSort Ordenados

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	0.20	0.40	0.50	0.60	0.70
2	0.30	0.40	0.30	0.60	0.60
3	0.20	0.50	0.40	0.60	0.60
4	0.30	0.50	0.30	0.60	0.60
5	0.40	0.40	0.20	0.50	0.60
Média p/ Elementos	0.28	0.44	0.34	0.58	0.62
Cronômetro	3Milésimos	4Milésimos	3Milésimos	6Milésimos	6Milésimos

BubbleSort Inversamente Ordenado

Número de Elemento	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	9	40.166	162.749	1023	5623
2	10	40.222	143.377	1055	5622
3	9	40.356	278.799	1102	5688
4	8	40.335	250.655	1036	5702
5	10	40.111	166.777	1152	5763
Media p/ Elementos	9.2	40.238	200.4714	1073.6	5679.6
Cronometro	0:00:09	0:00:40	0:03:00	0:17:45	1:34:33

SelectionSort Aleatório

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	4.50	16.20	85.56	488.54	2055.15
2	4.00	16.40	86.56	490.08	2155.28
3	3.93	17.00	87.66	487.03	2005.16
4	4.00	16.85	88.70	486.61	2154.65
5	4.15	16.40	90.50	489.55	2011.65
Média p/ Elementos	4.12	16.57	87.80	488.36	2076.38
Cronômetro	0:00:16	0:00:16	0:04:40	0:08:13	0:34:06

SelectionSort Ordenados

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	4.10	7.70	31.55	214.13	1152.00
2	4.20	6.90	32.65	214.48	1132.67
3	4.30	6.90	34.45	215.77	1140.15
4	4.00	7.70	33.75	215.01	1147.53
5	4.30	7.50	35.43	214.45	1156.60
Média p/ Elementos	4.18	7.34	33.57	214.77	1145.79
Cronômetro	0:00:04	0:00:07	0:00:33	0:03:56	0:19:09

SelectionSort Inversamente Ordenado

Número de Elemento	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	6.10	18.17	100.46	1252.53	5698.60
2	5.35	20.44	95.67	1154.17	5898.75
3	4.32	19.50	102.85	1032.39	5459.92
4	4.35	18.56	100.39	1110.03	5320.05
5	5.32	19.62	26.10	1296.90	5256.67
Média p/ Elementos	5.09	19.26	85.09	1169.20	5526.80
Cronômetro	0:00:05	0:00:19	0:01:41	0:19:48	1:32:11

InsertionSort Aleatório

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	7.78	16.73	71.52	610.46	2699.50
2	3.90	16.63	70.25	630.57	2756.15
3	4.10	15.98	72.36	632.50	2569.36
4	3.88	16.25	69.87	647.59	2508.47
5	3.95	16.60	68.95	615.35	2358.48
Média p/ Elementos	4.72	16.44	70.59	627.29	2578.39
Cronômetro	0:00:04	0:00:16	0:01:16AM	0:10:45	0:42:97 AM

InsertionSort Ordenados

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	0.30	0.60	0.70	0.70	0.20
2	0.20	0.50	0.70	0.70	0.20
3	0.30	0.50	0.70	0.70	0.90
4	0.20	0.50	0.70	0.70	0.90
5	0.10	0.60	0.70	0.70	0.80
Média p/ Elementos	0.22	0.54	0.70	0.70	0.60
Cronômetro	22Milésimos	54Milésimos	70Milésimos	70Milésimos	60Milésimos

InsertionSort Inversamente Ordenados

Número de Elemento	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	4.46	18.30	116.06	820.35	2854.85
2	4.23	17.35	118.75	815.36	2745.62
3	5.95	19.36	117.14	824.50	2547.13
4	4.75	18.25	116.65	821.69	2598.54
5	6.00	18.95	116.75	811.45	2578.12
Média p/ Elementos	5.08	18.44	117.07	818.67	2664.85
Cronômetro	0:00:05	0:00:18	0:02:55	0:13:44	0:44:41

Figura 8: Resultado dos métodos não eficientes de ordenação
Fonte: Próprio autor.

Métodos eficientes de ordenações de elementos

ShellSort Aleatório

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	0.05	0.10	0.21	0.54	1.09
2	0.04	0.08	0.17	0.50	1.15
3	0.04	0.09	0.17	0.49	1.06
4	0.03	0.08	0.19	0.49	1.11
5	0.04	0.08	0.18	0.49	1.08
Média p/ Elementos	0.04	0.09	0.18	0.50	1.10
Cronômetro	04Milésimos	09Milésimos	18Milésimos	5Milésimos	0:00:01

ShellSort Ordenados

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	0.01	0.03	0.03	0.04	0.06
2	0.01	0.03	0.03	0.04	0.06
3	0.01	0.03	0.03	0.04	0.06
4	0.01	0.03	0.03	0.04	0.06
2	0.01	0.03	0.03	0.04	0.06
Média p/ Elementos	0.01	0.03	0.03	0.04	0.06
Cronômetro	01Milésimos	03Milésimos	03Milésimos	04Milésimos	06Milésimos

ShellSort Inversamente Ordenado

Número de Elemento	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	0.02	0.04	0.06	0.12	0.25
2	0.02	0.04	0.08	0.12	0.27
3	0.02	0.04	0.07	0.13	0.25
4	0.02	0.04	0.07	0.13	0.25
5	0.02	0.04	0.07	0.13	0.25
Média p/ Elementos	0.02	0.04	0.07	0.13	0.25
Cronômetro	02Milésimos	04Milésimos	07Milésimos	13Milésimos	25Milésimos

MergeSort Aleatório

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	7.90	26.31	104.00	650.00	2662.50
2	7.69	26.27	103.00	640.00	2660.00
3	7.51	25.88	105.00	650.00	2662.00
4	7.77	25.97	105.00	650.00	2661.00
5	7.84	26.69	105.00	640.00	2660.00
Média p/ Elementos	7.74	26.22	104.40	646.00	2661.10
Cronômetro	0:00:07	0:00:26	0:01:43	0:10:46	0:44:35

MergeSort Ordenados

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	7.70	24.47	102.64	244.63	3229.95
2	7.70	26.11	102.00	233.54	3325.63
3	7.60	26.25	101.70	241.96	3102.64
4	7.50	25.65	102.35	242.36	3165.44
5	7.40	25.50	102.52	247.15	3108.56
Média p/ Elementos	7.58	25.60	102.24	241.93	3186.44
Cronômetro	0:00:07	0:00:25	0:01:41	0:04:32	0:53:10

MergeSort Inversamente Ordenado

Número de Elemento	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	7.84	26.65	102.80	285.65	3574.00
2	7.84	25.81	102.40	289.13	3657.77
3	7.66	26.32	102.60	288.65	3499.02
4	7.39	26.47	102.40	279.51	3468.08
5	7.91	26.00	102.20	278.92	3509.68
Média p/ Elementos	7.73	26.25	102.48	284.37	3541.71
Cronômetro	0:00:07	0:00:26	0:01:41	0:04:43	0:59:02

QuickSort Aleatório

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	0.05	0.09	0.13	0.32	0.70
2	0.05	0.08	0.18	0.33	0.64
3	0.05	0.08	0.13	0.31	0.68
4	0.05	0.09	0.14	0.33	0.65
5	0.05	0.08	0.15	0.32	0.71
Média p/ Elementos	0.05	0.08	0.15	0.32	0.68
Cronômetro	05Milésimos	08Milésimos	15Milésimos	32Milésimos	68Milésimos

QuickSort Ordenados

Número de Elementos	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	0.03	0.04	0.06	0.10	0.16
2	0.02	0.03	0.06	0.11	0.22
3	0.02	0.04	0.05	0.11	0.18
4	0.03	0.03	0.06	0.11	0.20
5	0.02	0.03	0.05	0.10	0.17
Média p/ Elementos	0.02	0.03	0.06	0.11	0.19
Cronômetro	02Milésimos	03Milésimos	06Milésimos	11Milésimos	19Milésimos

QuickSort Inversamente Ordenados

Número de Elemento	Tempo (Segundos)				
	100.000	200.000	400.000	1.000.000	2.000.000
1	0.03	0.05	0.06	0.12	0.24
2	0.02	0.04	0.07	0.11	0.31
3	0.02	0.05	0.06	0.10	0.23
4	0.02	0.05	0.07	0.11	0.25
5	0.03	0.04	0.06	0.11	0.30
Média p/ Elementos	0.02	0.05	0.06	0.11	0.27
Cronômetro	02Milésimos	05Milésimos	06Milésimos	11Milésimos	27Milésimos

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...")
```