**FIAP**

ANÁLISE E DESENVOLVIMENTO DE SISTEMA

Integrantes:

Amorgan Mendes Lopes; RM: 98552

Gustavo Godoi da Silva; RM: 99585

CP 5: PYTHON

Algoritmos de ordenação (sort)

SÃO PAULO

-------------------------------------------------------------------------------------------------------------------------------

Especificações das máquinas utilizadas:

\*-Gustavo (Máquina que rodou 50K, 70K, 80K)-\*

->6 RAM - 2,40 Hz - Processador Intel Core i3 2370Mhz - SSD 128

\*-Amorgan (Máquina que rodou 100K, 200K e 300K)-\*

->8 RAM - 2.10 GHz – Processador AMD Ryzen 5 3500U

-------------------------------------------------------------------------------------------------------------------------------

Bubble Sort:

Um algoritmo de ordenação simples e o menos eficiente na maioria dos casos. Nesse algoritmo o pivô terá que percorrer a lista diversas vezes comparando os números e os colocando em ordem até que não haja mais nenhum elemento fora de ordem.

Código:

#Algoritimo de ordenação (Bubble\_sort)

def *bubble\_sort*(*lista*):

    tam = len(lista)

**for** i **in** range(tam - 1):

**for** j **in** range(0, tam - i - 1):

**if** lista[j] > lista[j + 1]:

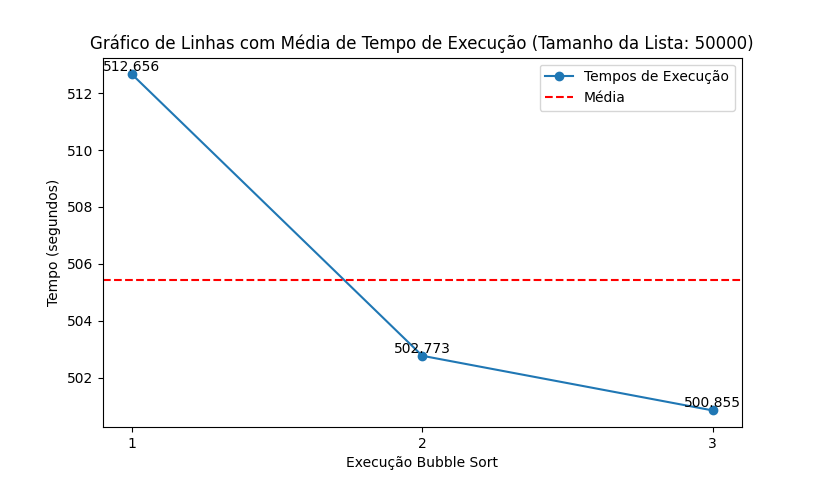
                lista[j], lista[j + 1] = lista[j + 1], lista[j]

Valores e média:

-Gustavo:

50 mil:

A média é de 00:08:25 (505.428)



70 mil:

A média é 00:15:19 (919.503)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

90 mil:

A média é 00:27:34 (1654.228)

Gráfico de linhas

Descrição gerada automaticamente

-Amorgan:

100 mil:

A média é 00:20:20 (1220.192)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

200mil:

A média é 01:04:41 (3881.357)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

300 mil:

A média é 02:44:24 (9864.878)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Considerações e experiencia:

Utilizamos pela primeira vez de fato um algoritmo de ordenação e começamos com o Bubble sort, já de início podemos relatar o tempo assustador que leva pra ordenar uma lista, até mesmo das menores.

Inicialmente iriamos fazer com o valor máximo de 2 milhões, mas depois de um longo tempo de espera, percebemos que mesmo que saíssemos daquela demora no dia seguinte, escalonar de forma não exagerada o intervalo do número máximo de elementos de uma lista pra outra seria impossível, visto que não terminaríamos o projeto até o prazo dado.

Sendo assim, mesmo com uma quantidade consideravelmente menor de elementos, conseguimos equilibrar de uma forma que não quebrasse nossa experiencia, chegando ao valor de 300 mil elementos como ponto máximo, e mesmo assim foram longas e longas horas olhando para o computador e questionando se a ordenação ainda estava sendo aplicada.

-------------------------------------------------------------------------------------------------------------------------------

Insertion Sort:

Esse algoritmo constrói a lista ordenada de forma gradual, inserindo cada elemento na posição ideal enquanto uma lista não ordenada, movendo assim os elementos maiores para o lado direito dessa lista.

Código:

#Algoritimo de ordenação (insertion\_sort)

def *insertion\_sort*(*lista*):

**for** i **in** range(1, len(lista)):

        pivo = lista[i]

        j = i - 1

**while** j >= 0 and pivo < lista[j]:

            lista[j + 1] = lista[j]

            j -= 1

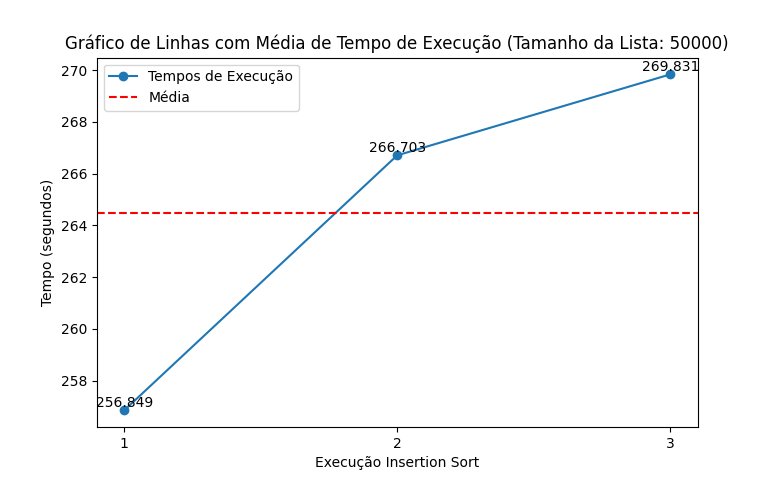
        lista[j + 1] = pivo

Valores e média:

- Gustavo:

50 mil:

A média é 00:04:24 (264.461)



70 mil:

A média é 00:06:39 (399.251)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

90 mil:

A média é 00:11:57 (717.671)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

-Amorgan:

100 mil:

A média é 00:06:26 (386.006)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

200 mil:

A média é 00:28:21 (1701.549)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

300 mil:

A média é 01:13:53 (4433.846)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Considerações e experiencia:

Nós dois achávamos que a diferença do bubble para o Insertion sort já seria algo notável, mas estávamos enganados, a diferença não chegou perto de ser algo realmente relevante em nossas máquinas e foi a nossa maior decepção entre os 4 métodos.

-------------------------------------------------------------------------------------------------------------------------------

Selection Sort:

Um algoritmo já com um grau razoável de complexidade.

O selection sort divide a lista em duas (elementos ordenados e não ordenados), assim selecionando o menor elemento dá lista desordenada e o alocando na lista já ordenada até que o processo seja concluído com todos os elementos necessários.

Código:

#Algoritimo de ordenação (selection\_sort)

def *selection\_Sort*(*lista*):

    tam = len(lista)

**for** i **in** range(tam):

        min\_index = i

**for** j **in** range(i + 1, tam):

**if** lista[j] < lista[min\_index]:

                min\_index = j

        lista[i], lista[min\_index] = lista[min\_index], lista[i]

Valores e média:

- Gustavo:

50 mil:

A média é de 00:03:59 (239.970)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

70 mil:

A média é 00:06:08 (368.122)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

90 mil:

A média é 00:11:48 (708.959)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

-Amorgan:

100 mil:

A média é 00:08:56 (536.011)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

200 mil:

A média é 00:26:07 (536.011)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

300 mil:

A média é 01:03:11 (3791.400)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Considerações e experiencia:

Aqui já estávamos percebendo uma boa diferença, ficamos até felizes que finalmente não ficaríamos quase metade do dia rodando uma lista só.

Após conversar com colegas da sala, percebemos que a discrepância entre o selection e o insertion que tivemos não foi uma experiencia de todos, já que por algum motivo alguns deles tiveram essa distancia bem menos relevante, mas em contrapartida tinham um tempo de ordenação bem melhores que o nosso desde o próprio bobble sort.

-------------------------------------------------------------------------------------------------------------------------------

Merge Sort:

O algoritmo mais eficiente e rápido que todos aqui citados.

Ele se baseia na estratégia “dividir para conquistar”, dividindo a lista em pequenas partes fracionadas e ordenando as mesmas depois de combinar seus elementos, formando assim uma lista ordenada.

Código:

#Algoritimo de ordenação (merge\_sort)

def *merge\_sort*(*lista*):

**if** len(lista) > 1:

        meio = len(lista) // 2

        L = lista[:meio]

        R = lista[meio:]

        merge\_sort(L)

        merge\_sort(R)

        i = j = k = 0

**while** i < len(L) and j < len(R):

**if** L[i] < R[j]:

                lista[k] = L[i]

                i += 1

**else**:

                lista[k] = R[j]

                j += 1

            k += 1

**while** i < len(L):

            lista[k] = L[i]

            i += 1

            k += 1

**while** j < len(R):

            lista[k] = R[j]

            j += 1

            k += 1

Valores e média:

- Gustavo:

50 mil:

A média é de 00:00:01 (1.888)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

70 mil:

A média é 00:00:00:84 (0.843)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

90 mil:

A média é 00:00:00:96 (1.011)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

-Amorgan:

100 mil:

A média é 00:00:00:88 (0.885)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

200 mil:

A média é 00:00:01 (1.640)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

300 mil:

A média é 00:00:02 (2.286)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

BONUS:

5 MILHÕES

A média é 00:00:49 (49.642)

Gráfico, Gráfico de linhas

Descrição gerada automaticamente

Considerações e experiencia:

E então chegamos na nossa maior surpresa do projeto, o Merge sort.

Sua eficiência e velocidade são simplesmente avassaladoras se comparadas a qualquer um dos últimos algoritmos, encerrando em menos de 1 segundo algumas listas que utilizamos.

Foi uma sensação tão satisfatória e surpreendente que tivemos que testar o que ela faria com uma lista de 5 milhões de elementos, mas não tínhamos a mínima confiança em rodar essa quantidade de elementos em nossas máquinas, mas mais uma vez, fomos surpreendidos.

Na máquina do Amorgan, foram incríveis e inacreditáveis 40 segundos para ordena-la.