Algoritmos de ordenação

Disciplina: Algoritmos e estruturas de dados

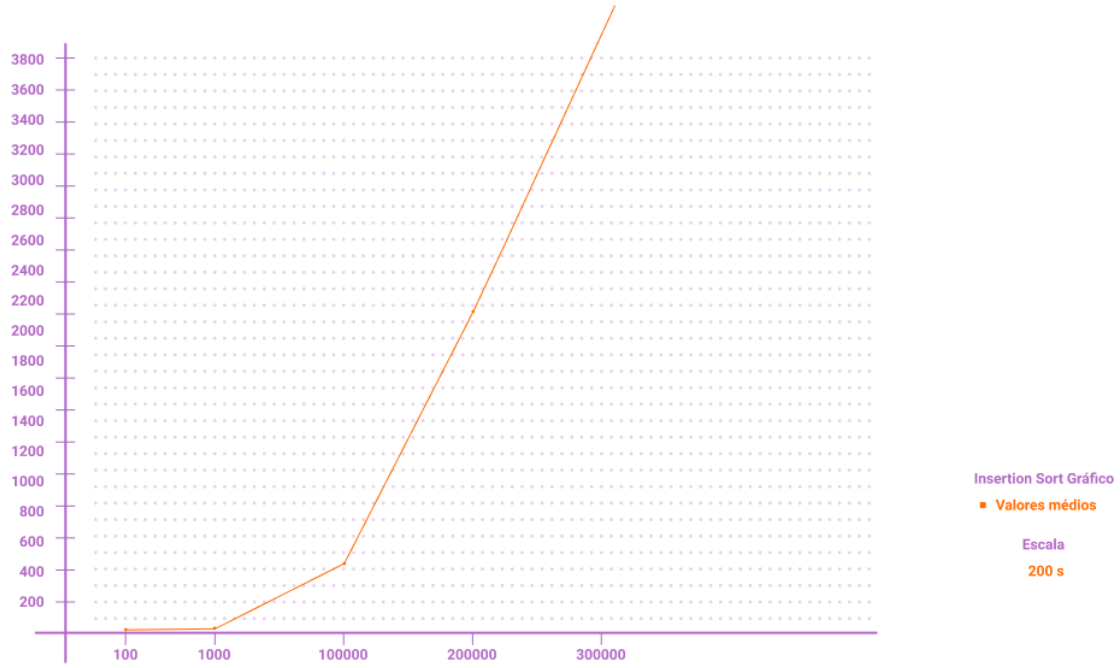
Alunos:

- Alecsander Silva da Costa
- Linda Hillary Brandão Souza
- Mateus Santos

Isertion Sort

O Insertion-Sort (ordenação por inserção) inicia com o segundo valor do vetor (índice 1) e vai alocando-o para a esquerda (início do vetor). Ele usa um laço de repetição para percorrer todo o vetor uma única vez e outro laço interno para poder alocar os valores para a esquerda.

Qtd.	Tempo 1	Tempo 2	Tempo 3	Tempo Medio
100	0.00047270s	0.00046020s	0.00040580s	0.00044623s
1000	0.04873180s	0.04264620s	0.04319980s	0.04485927s
100000	482.98712390s	468.57839360s	464.28688690s	471.95080147s
200000	2071.21434900s	-	-	2071.21434900s
300000	5011.40649160s	-	-	5011.40649160s




```
from util import *
import timeit

def insert_sort(lista):
```

```
    #percorre o vetor complemante
    for i in range(1, len(lista)):
        chave = lista[i]
        j = i - 1
        #percorre o vetor para trás
        while j >= 0 and chave < lista[j]:
            lista[j + 1] = lista[j]
            j = j - 1

        lista[j + 1] = chave

    return lista


valores = [100, 1000, 100000, 200000, 300000]
cabecalho = f'|Qtd.|Tempo 1|Tempo 2|Tempo 3|Tempo Medio|\n|--|--|--|--|--|\n'
with open('insertion300.md', 'w') as arquivo:
    arquivo.write(cabecalho)
    for valor in valores:
        media_tempo = 0
        linha = f'|{valor}|'
        for i in range(3):
            print('criando lista {} do valor {}'.format(i+1, valor))
            lista = randomList(valor, True, 0, valor)
            print('ordenando lista {} do valor {}'.format(i+1, valor))
            inicio = timeit.default_timer()
            lista_ordenada = insert_sort(lista)
            fim = timeit.default_timer()
            linha += f'{(fim - inicio):.8f}s|'
            media_tempo += float(f'{(fim - inicio):.8f}')

        media_tempo /= 3
        linha += f'{media_tempo:.8f}s|\n'

        arquivo.write(linha)
```

Quick Sort

O algoritmo quicksort é um método de ordenação muito rápido e eficiente, inventado por C.A.R.

O quicksort adota a estratégia de divisão e conquista. A estratégia consiste em rearranjar as chaves de modo que as chaves "menores" precedam as chaves "maiores". Em seguida o quicksort ordena as duas sublistas de chaves menores e maiores recursivamente até que a lista completa se encontre ordenada. Os passos são:

1. Escolha um elemento da lista, denominado ;

   pivô

2. Particiona: rearranje a lista de forma que todos os elementos anteriores ao pivô sejam menores que ele, e todos os elementos posteriores ao pivô sejam maiores que ele. Ao fim do processo o pivô estará em sua posição final e haverá duas sub listas não ordenadas. Essa operação é denominada ;

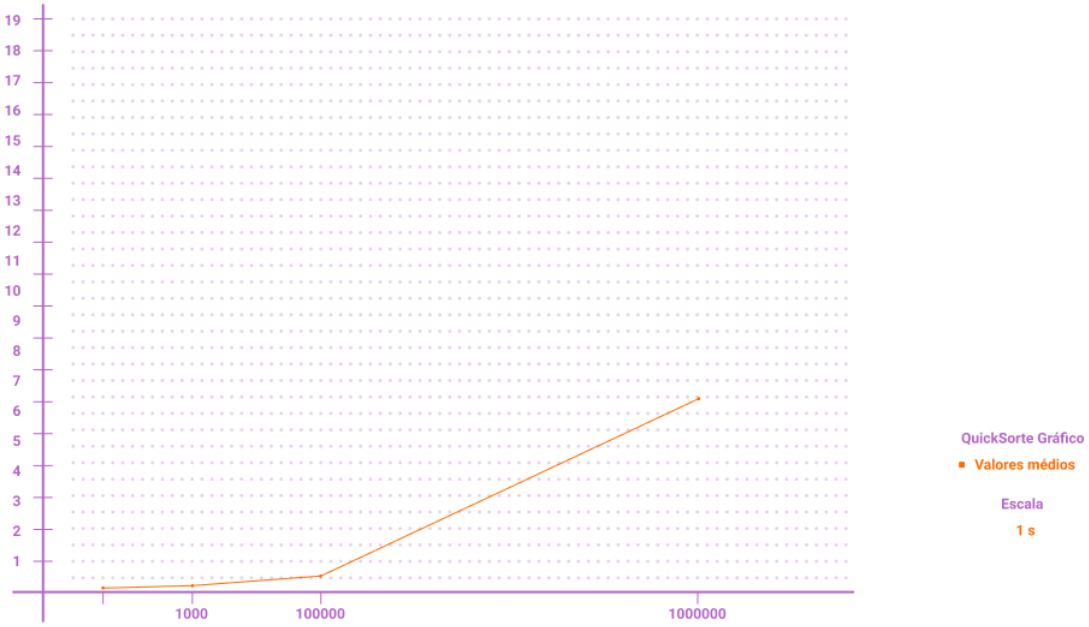
   partição

3. Recursivamente ordene a sub lista dos elementos menores e a sub lista dos elementos maiores;

O caso base da recursão são as listas de tamanho zero ou um, que estão sempre ordenadas. O processo é finito, pois a cada iteração pelo menos um elemento é posto em sua posição final e não será mais manipulado na iteração seguinte.

A escolha do pivô e os passos do Particiona podem ser feitos de diferentes formas e a escolha de uma implementação específica afeta fortemente a performance do algoritmo.

Qtd.	Tempo 1	Tempo 2	Tempo 3	Tempo medio
100	0.00018660s	0.00028750s	0.00035230s	0.00046440s
1000	0.00376230s	0.00297810s	0.00746770s	0.00456180s
100000	0.58961740s	0.59177330s	0.50064740s	0.54859820s
1000000	6.43238040s	6.54314490s	6.83572570s	6.85164800s

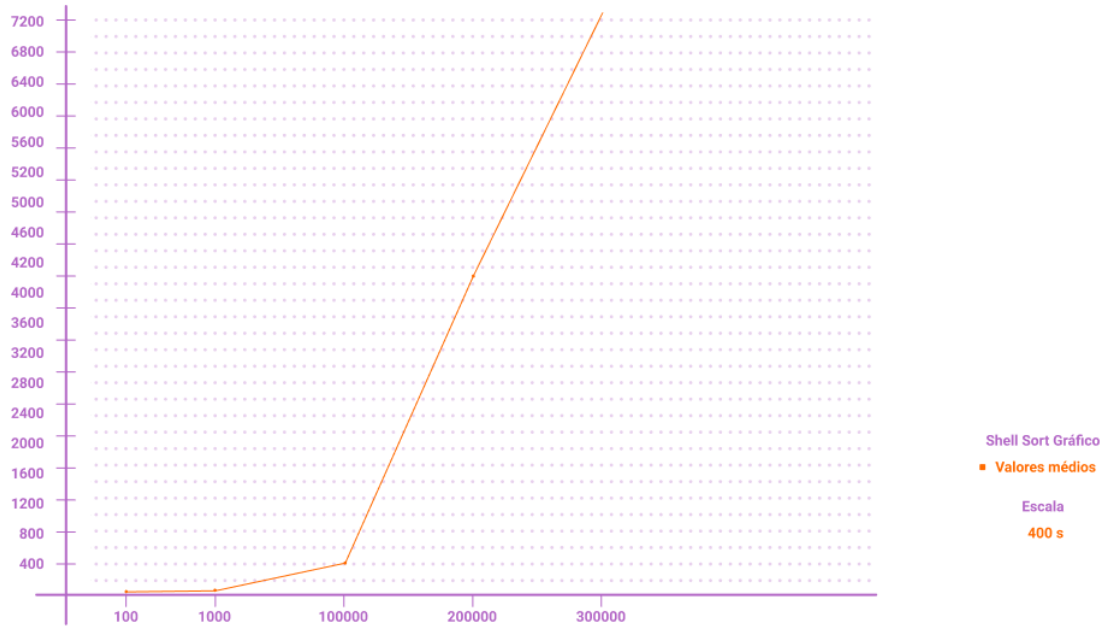



```
import timeit
from util import *

def quicksort(lista, inicio=0, fim=None):
    if fim is None:
        fim = len(lista)-1
    if inicio < fim:
        p = partition(lista, inicio, fim)
        # recursivamente na sublista à esquerda (menores)
        quicksort(lista, inicio, p-1)
        # recursivamente na sublista à direita (maiores)
        quicksort(lista, p+1, fim)

def partition(lista, inicio, fim):
    pivot = lista[fim]
    i = inicio
    for j in range(inicio, fim):
        # j sempre avança, pois representa o elementa em análise
        # e delimita os elementos maiores que o pivô
        if lista[j] <= pivot:
            lista[j], lista[i] = lista[i], lista[j]
            # incrementa-se o limite dos elementos menores que o pivô
            i = i + 1
    lista[i], lista[fim] = lista[fim], lista[i]
    return i

values = [12]
header = f'|Qtd.|Tempo 1|Tempo 2|Tempo 3|Tempo medio|\n|--|--|--|--|--|\n'
with open("test.md", "w") as f:
    f.write(header)
    for v in values:
        avg = 0
        line = f'|{v}|'
        for i in range(3):
            print(f'Criando a {i+1}ª lista de {v} valores...')
            a = randomList(v, True, 0, v)
            print(f'Ordenando a {i+1}ª lista de {v} valores...')
            print(a)
            start = timeit.default_timer()
            quicksort(a)
            end = timeit.default_timer()
            print(a)
            line += f'{(end - start):.8f}s|'
            avg += float(f'{(end - start):.8f}')
        avg /= 3
        line += f'{avg:.8f}s|\n'

        f.write(line)
```

Shell Sort

Tanto o seu funcionamento quanto o código são parecidos ao insert sort, seu diferencial é que ele começa a ordenar elementos que estão mais distantes, e vai diminuindo a distância do ordenamento, até que a lista esteja totalmente ordenada.

Qtd.	Tempo 1	Tempo 2	Tempo 3	Tempo Medio
100	0.00050450s	0.00045800s	0.00045580s	0.00047277s
1000	0.04274610s	0.03858530s	0.04070380s	0.04067840s
100000	492.39783810s	409.84731850s	396.72636860s	432.99050840s
200000	4001.08500550s	-	-	4001.08500550s
300000	8139.82452450s	-	-	8139.82452450s




```
from util import *
import timeit
def ShellSort(nums):
    #definimos os valores para h, que será o responsavel pela ordenação.
    h = 1
    n = len(nums)
    while h > 0:
        #enquanto h for maior que 0, será feita a continua reorganização da lista
        for i in range(h, n):
            c = nums[i]
            j = i
            while j >= h and c < nums[j - h]:
                nums[j] = nums[j - h]
                j = j - h
                #h será subitraido a -1 até que a lista esta organizada
                nums[j] = c
        h = int(h / 2.2)
    return nums
#função para criar uma lista aleatoria, com os parametros:quantidade de itens na lista; se pode repetir os numeros; de onde começa;de
```

```
values = [100, 1000, 100000, 200000, 300000]
header = f'|Qtd.|Tempo 1|Tempo 2|Tempo 3|Tempo Medio|\n|--|--|--|--|--|\n'
with open('shell_sort300000.md','w') as f:
    f.write(header)
    for v in values:
        avg = 0
        line = f'|{v}|'
        for i in range(1):
            print(f'Criando a {i + 1}ª lista de {v} valores')
            num = randomList(v, True, 0, v)
            print(f'Ordenando a {i + 1}ª lista de {v} valores')
            start = timeit.default_timer()
            my_list =ShellSort(num)
            end = timeit.default_timer()
            line += f'{(end - start):.8f}s|'
            avg += float(f'{(end - start):.8f}')
        avg /= 3
        line += f'{avg:.8f}s|\n'

        f.write(line)
```

Gerador de lista com números aleatórios

Para os casos de teste a equipe criou uma função que retorna uma lista com valores aleatórios desordenados. Essa função foi armazenada no arquivo util.py.

```
from random import randint

def randomList(qtt = 0, repeat = False, start = 0, end = 0):
    numbers = []
    while len(numbers) < qtt:
        number = randint(start, end)
        if not repeat:
            if number not in numbers:
                numbers.append(number)
        else:
                numbers.append(number)
    return numbers
```