Relatório Projeto 3 AED 2023/2024

Nome: Rodrigo Borges
Nº Estudante: 2022244993
PL (inscrição): 7 Email: uc2022244993@student.uc.pt

IMPORTANTE:

- As conclusões devem ser manuscritas... texto que não obedeça a este requisito não é considerado.
- Texto para além das linhas reservadas, ou que não seja legível para um leitor comum, não é considerado.
- O relatório deve ser submetido num único PDF que deve incluir os anexos. A não observância deste formato é penalizada.

1. Planeamento

	Semana 1	Semana 2	Semana 3	Semana 4	Semana 5
Insertion Sort		x			
Heap Sort			x		
Quick Sort				x	
Finalização Relatório					x

2. Recolha de Resultados *(tabelas)*

Insertion Sort:

Algorithm Type	Time	Elements	Input Type
Insertion Sort	0,001005411	20000	A
Insertion Sort	0,002965212	40000	A
Insertion Sort	0,004031181	60000	A
Insertion Sort	0,005997896	80000	A
Insertion Sort	0,007032633	100000	A
Insertion Sort	34,51708937	20000	B
Insertion Sort	140,6959507	40000	B
Insertion Sort	316,0503759	60000	B
Insertion Sort	561,9721806	80000	B
Insertion Sort	881,6545157	100000	B
Insertion Sort	17,66074562	20000	C
Insertion Sort	70,03372765	40000	C
Insertion Sort	157,9431815	60000	C
Insertion Sort	283,2553246	80000	C
Insertion Sort	439,8197994	100000	C

Heap Sort:

Algorithm Type	Time	Elements	Input Type
Heap Sort	0,16	20000	A
Heap Sort	0,33	40000	A
Heap Sort	0,52	60000	A
Heap Sort	0,71	80000	A
Heap Sort	0,9	100000	A
Heap Sort	0,14	20000	B
Heap Sort	0,31	40000	B
Heap Sort	0,47	60000	B
Heap Sort	0,65	80000	B
Heap Sort	0,84	100000	B
Heap Sort	0,15	20000	C
Heap Sort	0,33	40000	C
Heap Sort	0,53	60000	C
Heap Sort	0,69	80000	C
Heap Sort	0,89	100000	C

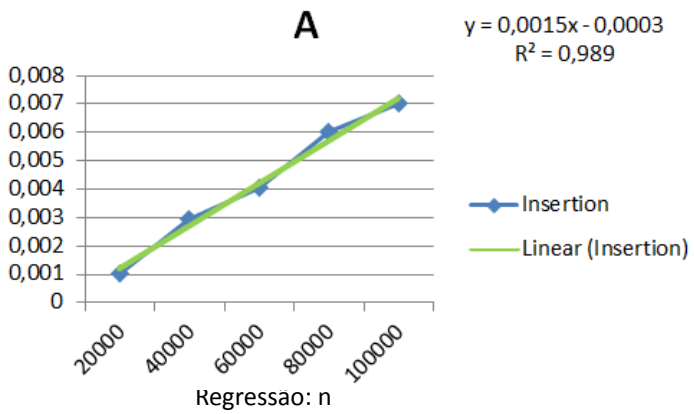
Quick Sort:

Algorithm Type	Time	Elements	Input Type
Quick Sort	0,03	20000	A
Quick Sort	0,07	40000	A
Quick Sort	0,11	60000	A
Quick Sort	0,15	80000	A
Quick Sort	0,19	100000	A
Quick Sort	0,06	20000	B
Quick Sort	0,13	40000	B
Quick Sort	0,2	60000	B
Quick Sort	0,27	80000	B
Quick Sort	0,35	100000	B
Quick Sort	0,04	20000	C
Quick Sort	0,09	40000	C
Quick Sort	0,13	60000	C
Quick Sort	0,18	80000	C
Quick Sort	0,23	100000	C

3. Visualização de Resultados *(gráficos)*

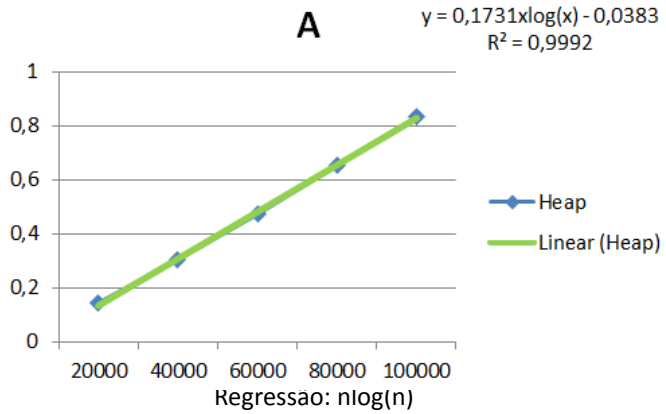
Conjunto A:

Insertion Sort:



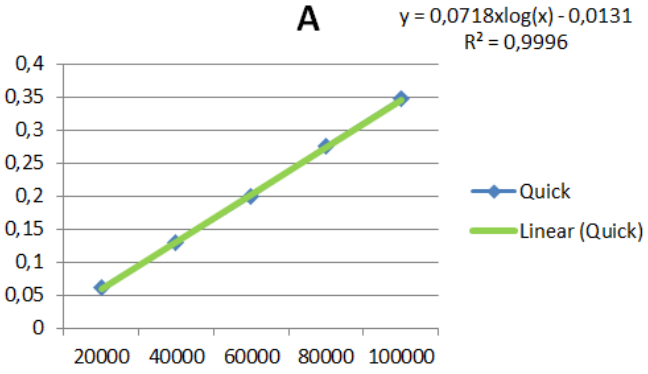
Regressão: n

Heap Sort:



Regressão: nlog(n)

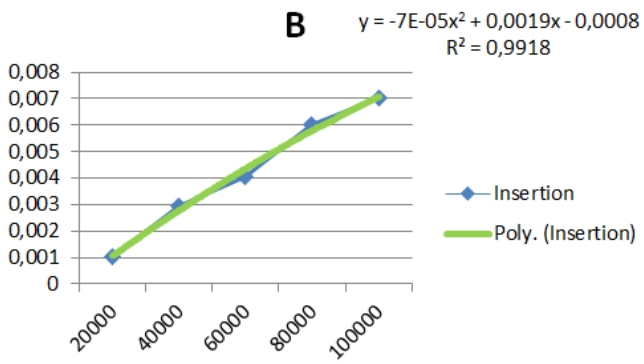
Quick Sort:



Regressão: nlog(n)

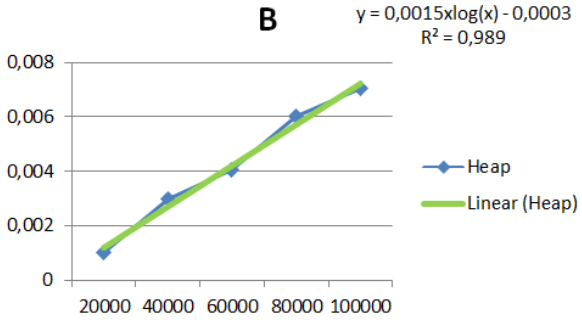
Conjunto B:

Insertion Sort:



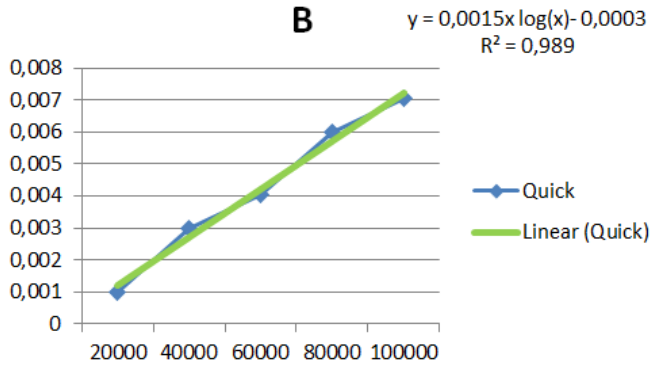
Regressão: n^2

Heap Sort:



Regressão: nlog(n)

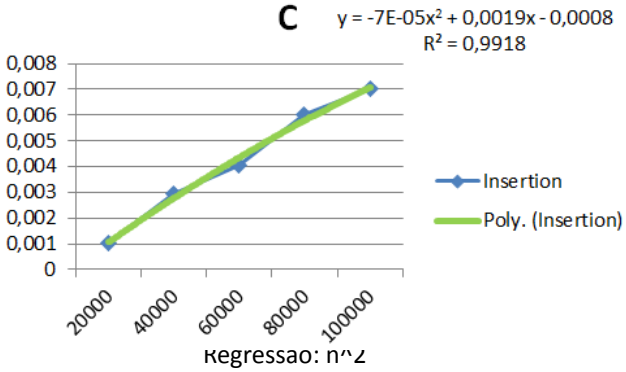
Quick Sort:



Regressão: nlog(n)

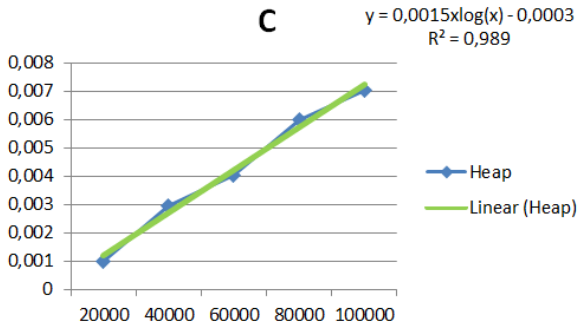
Conjunto C:

Insertion Sort:



Regressao: n^2

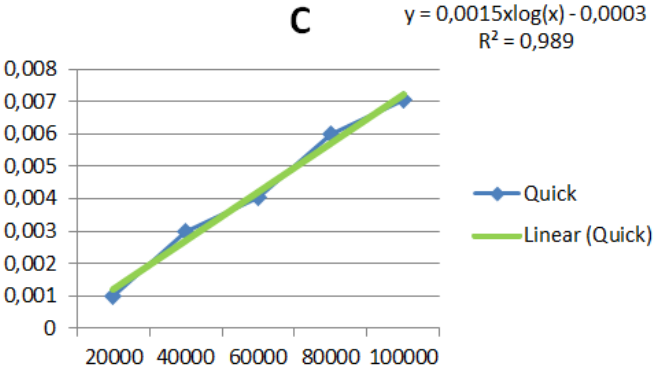
Heap Sort:



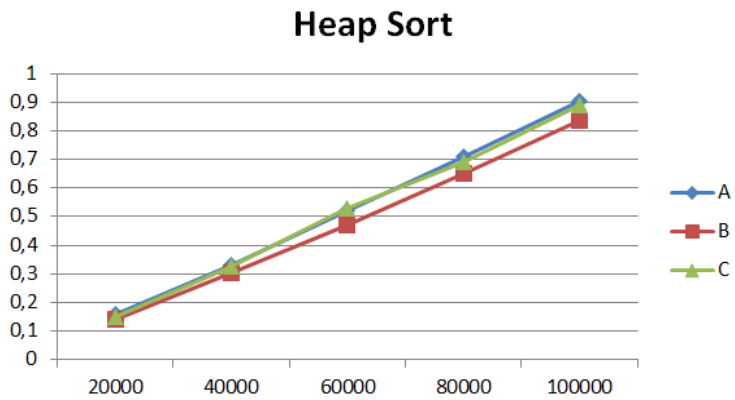
Regressão: nlog(n)

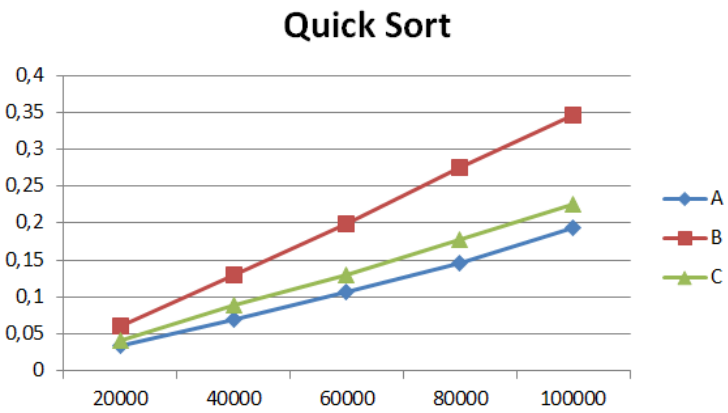
Quick Sort:

Regressão: nlog(n)



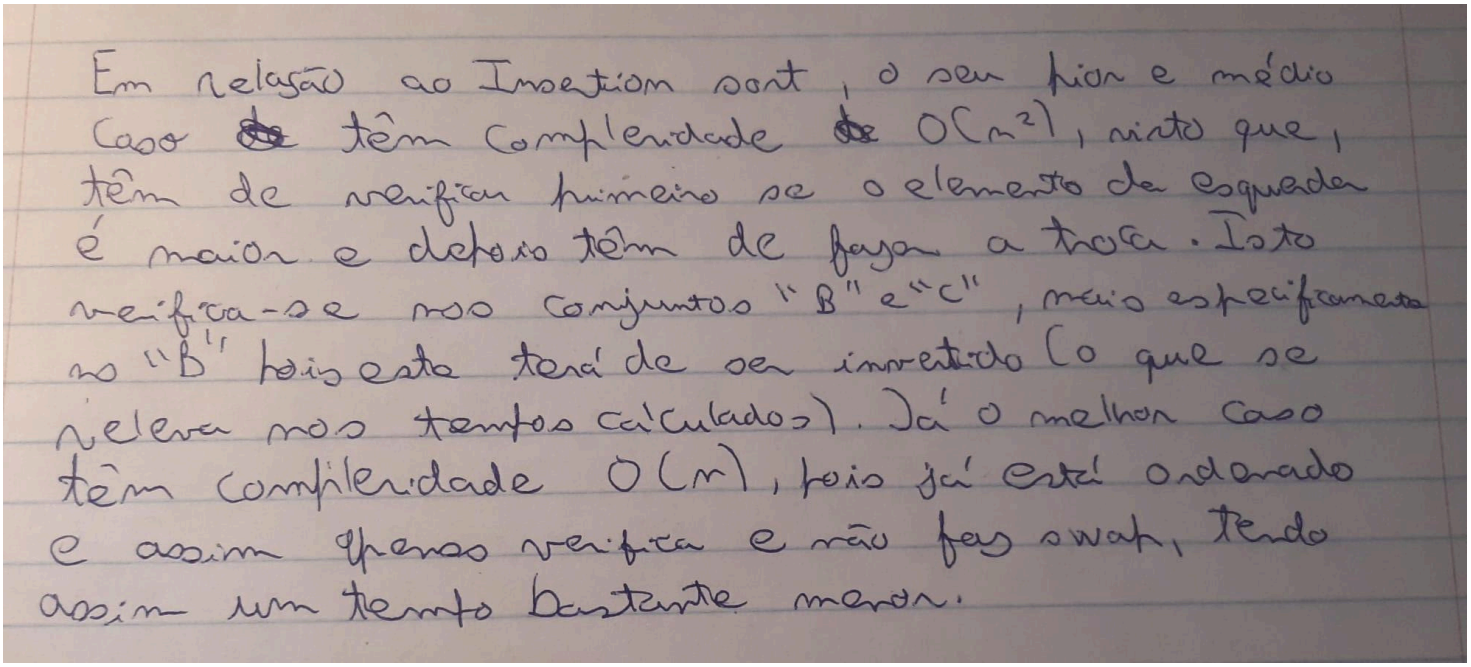
Por sorts:



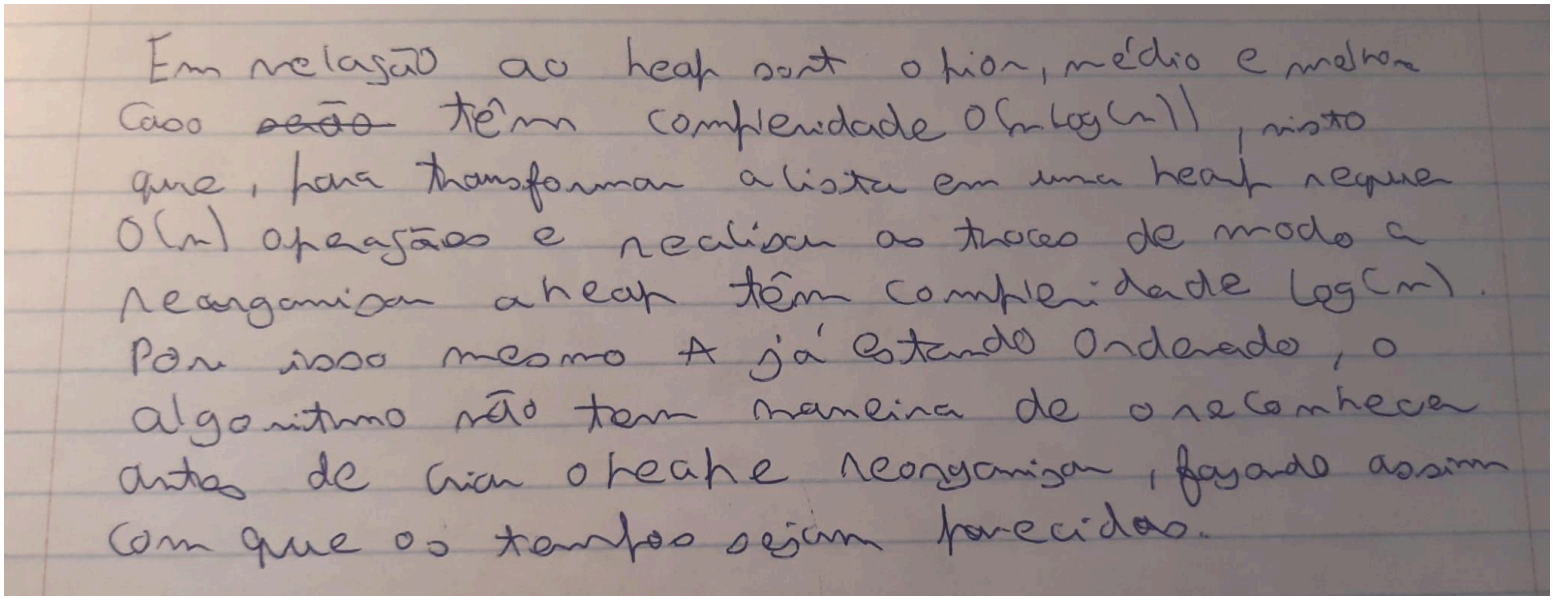


4. Conclusões *(as linhas desenhadas representam a extensão <u>máxima</u> de texto manuscrito)*

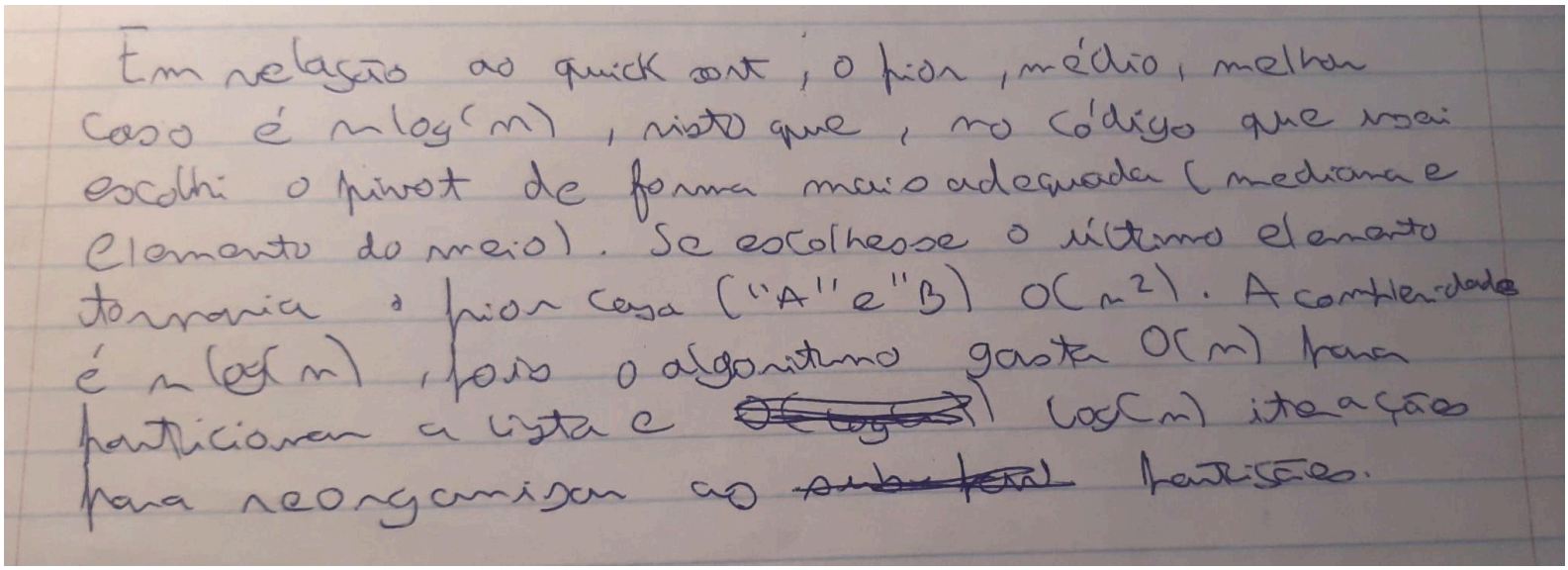
4.1 Tarefa 1


Em relação ao Insertion sort, o seu pior e médio caso têm complexidade $O(n^2)$, visto que, têm de verificar primeiro se o elemento da esquerda é maior e depois têm de fazer a troca. Isto verifica-se nos conjuntos "B" e "C", mais especificamente no "B" pois este terá de ser invertido (o que se relevou nos tempos calculados). Já o melhor caso têm complexidade $O(n)$, pois já está ordenado e assim apenas verifica e não faz swap, tendo assim um tempo bastante menor.

4.2 Tarefa 2


Em relação ao heap sort o pior, médio e melhor caso ~~são~~ têm complexidade O(n log(n)), visto que, para transformar a lista em uma heap requer O(n) operações e realizar as trocas de modo a reorganizar a heap têm complexidade log(n). Por isso mesmo A já estando ordenado, o algoritmo não tem maneira de o reconhecer antes de criar o heap e reorganizar, fazendo assim com que os tempos sejam parecidos.

4.3 Tarefa 3


Em relação ao quick sort, o pior, médio, melhor caso é n log(n), visto que, no código que usei escolhi o pivot de forma mais adequada (mediana e elemento do meio). Se escolhesse o último elemento tornaria o pior caso ("A" e "B") O(n²). A complexidade é n(log n), pois o algoritmo gasta O(n) para particionar a lista e ~~O(log(n))~~ log(n) iterações para reorganizar ao ~~subtotal~~ partições.

Anexo A - Delimitação de Código de Autor

```
import random
import time
import pandas as pd
#Criação dos inputs
def criar_input(n, c):
    numeros = list(range(1, n + 1))
    random.shuffle(numeros)
    lista = []
    if c == "A":
        lista = sorted(numeros)
    elif c == "B":
        lista = sorted(numeros, reverse=True)
    elif c == "C":
        lista = numeros
    return lista
# Insertion sort
def insertion_sort(lista):
    #itera do segundo até o ultimo elemento da lista
    for i in range(1,len(lista)):
        j=i
        #verifica se o que está à esquerda na lista é maior e então
troca~
        while lista[j-1] >lista[j] and j>0:
           temp = lista[j-1]
           lista[j-1] = lista[j]
           lista[j] = temp
           j-=1
# Heap Sort
def swap(lista,i,j):
    #troca a posição dos elementos
    temp = lista[i]
    lista[i] = lista[j]
```

```
    lista[j] = temp
def heap_sort(lista):
    #Transforma a lista em um heap
    for i in range((len(lista)-2)//2,-1,-1):
       heapify(lista,i,len(lista))
    for j in range(len(lista)-1,0,-1):
       # Troca o elemento raiz (maior elemento) com o último
elemento não ordenado
       swap(lista,0,j)
       # Reorganiza a heap, excluindo o último elemento ordenado
       heapify(lista,0,j)
def partition(lista, menor, maior):
    i = menor - 1
    pivot = lista[maior]

    for j in range(menor, maior):
        if lista[j] <= pivot:
            i += 1
            swap(lista,i,j)

    swap(lista,i+1,maior)
    return i + 1  # Índice do pivô

def escolher_pivo(lista, menor, maior):
    meio = (menor + maior) // 2
    if lista[menor]>lista[meio]:
        swap(lista,menor,meio)
    if lista[meio]>lista[maior]:
        swap(lista,meio,maior)
        if lista[menor]>lista[meio]:
            swap(lista,menor,meio)
    swap(lista,meio,maior)

def _quick_sort(lista, menor, maior):
        if menor < maior:
            escolher_pivo(lista, menor, maior)
            partition_pos = partition(lista, menor, maior)
            _quick_sort(lista, menor, partition_pos - 1)
```

```
            _quick_sort(lista, partition_pos + 1, maior)


def quick_sort(lista):
    _quick_sort(lista, 0, len(lista) - 1)
```

Anexo B - Referências

geekforgeeks:

```
def heapify(lista,i,upper):
    #Garante que a subárvore tenha na raiz a sua heap máxima
    while(True):
        #define o index do filho esquerdo e filho direito de i
        left,right=i*2+1,i*2+2
        #verifica se i tem 2 filhos
        if max(left,right) < upper:
           if lista[i]>= max(lista[left],lista[right]):
               break
           elif lista[left] > lista[right]:
            swap(lista,i,left)
            i=left
           else:
            swap(lista,i,right)
            i=right
        #verifica se i tem 1 filho (esquerda)
        elif left< upper:
           if lista[left]>lista[i]:
              swap(lista,i,left)
              i=left
           else:
              break
        #verifica se i tem 1 filho (direita)
        elif right< upper:
           if lista[right]>lista[i]:
```

```
                swap(lista,i,right)
                i=right
            else:
                break
        else:
            break
```

chatgpt:

```
algorithm  = {
    'Insertion Sort':insertion_sort,
    'Heap Sort':heap_sort,
    'Quick Sort':quick_sort

}
input_types = ["A","B","C"]
def measure_sorting_time(algorithm, input_types, num_elements_list):
    data = {
        "Algorithm Type": [],
        "Time": [],
        "Elements": [],
        "Input Type": []
    }

    for algo_name, algo_func in algorithm.items():
        for input_type in input_types:
            for num_elements in num_elements_list:
                # Create input
                input_list = criar_input(num_elements, input_type)

                # Measure time
                start_time = time.time()
                algo_func(input_list)
                end_time = time.time()
                elapsed_time = end_time - start_time
                print(elapsed_time)
                # Store data
                data["Algorithm Type"].append(algo_name)
                data["Time"].append(elapsed_time)
                data["Elements"].append(num_elements)
                data["Input Type"].append(input_type)

    return pd.DataFrame(data)
```

```
# Define parameters
num_elements_list = [20000, 40000, 60000, 80000, 100000]

# Measure sorting time
sorting_data = measure_sorting_time(algorithm, input_types,
num_elements_list)
sorting_data.to_excel("sorting_data_todos.xlsx", index=False)
```

Anexo C – Listagem Código

```
import random
import time
import pandas as pd
#Criação dos inputs
def criar_input(n, c):
    numeros = list(range(1, n + 1))
    random.shuffle(numeros)
    lista = []
    if c == "A":
        lista = sorted(numeros)
    elif c == "B":
        lista = sorted(numeros, reverse=True)
    elif c == "C":
        lista = numeros
    return lista
# Insertion sort
def insertion_sort(lista):
    #itera do segundo até o ultimo elemento da lista
    for i in range(1,len(lista)):
        j=i
        #verifica se o que está à esquerda na lista é maior e então
troca~
        while lista[j-1] >lista[j] and j>0:
```

```
            temp = lista[j-1]
            lista[j-1] = lista[j]
            lista[j] = temp
            j-=1
# Heap Sort
def swap(lista,i,j):
    #troca a posição dos elementos
    temp = lista[i]
    lista[i] = lista[j]
    lista[j] = temp

def heapify(lista,i,upper):
    #Garante que a subárvore tenha na raiz a sua heap máxima
    while(True):
        #define o index do filho esquerdo e filho direito de i
        left,right=i*2+1,i*2+2
        #verifica se i tem 2 filhos
        if max(left,right) < upper:
           if lista[i]>= max(lista[left],lista[right]):
               break
           elif lista[left] > lista[right]:
            swap(lista,i,left)
            i=left
           else:
            swap(lista,i,right)
            i=right
        #verifica se i tem 1 filho (esquerda)
        elif left< upper:
           if lista[left]>lista[i]:
              swap(lista,i,left)
              i=left
           else:
```

```
            break
        #verifica se i tem 1 filho (direita)
        elif right< upper:
            if lista[right]>lista[i]:
                swap(lista,i,right)
                i=right
            else:
                break
        else:
            break
def heap_sort(lista):
    #Transforma a lista em um heap
    for i in range((len(lista)-2)//2,-1,-1):
        heapify(lista,i,len(lista))
    for j in range(len(lista)-1,0,-1):
        # Troca o elemento raiz (maior elemento) com o último
elemento não ordenado
        swap(lista,0,j)
        # Reorganiza a heap, excluindo o último elemento ordenado
        heapify(lista,0,j)



# Quick Sort
def partition(lista, menor, maior):
    i = menor - 1
    pivot = lista[maior]


    for j in range(menor, maior):
        if lista[j] <= pivot:
            i += 1
            swap(lista,i,j)
```

```
    swap(lista,i+1,maior)
    return i + 1  # Índice do pivô

def escolher_pivo(lista, menor, maior):
    meio = (menor + maior) // 2
    if lista[menor]>lista[meio]:
        swap(lista,menor,meio)
    if lista[meio]>lista[maior]:
        swap(lista,meio,maior)
        if lista[menor]>lista[meio]:
            swap(lista,menor,meio)
    swap(lista,meio,maior)

def _quick_sort(lista, menor, maior):
        if menor < maior:
            escolher_pivo(lista, menor, maior)
            partition_pos = partition(lista, menor, maior)
            _quick_sort(lista, menor, partition_pos - 1)
            _quick_sort(lista, partition_pos + 1, maior)

def quick_sort(lista):
    _quick_sort(lista, 0, len(lista) - 1)

algorithm  = {
    'Insertion Sort':insertion_sort,
    'Heap Sort':heap_sort,
    'Quick Sort':quick_sort

}
input_types = ["A","B","C"]
def measure_sorting_time(algorithm, input_types, num_elements_list):
    data = {
```

```
        "Algorithm Type": [],
        "Time": [],
        "Elements": [],
        "Input Type": []
    }

    for algo_name, algo_func in algorithm.items():
        for input_type in input_types:
            for num_elements in num_elements_list:
                # Create input
                input_list = criar_input(num_elements, input_type)

                # Measure time
                start_time = time.time()
                algo_func(input_list)
                end_time = time.time()
                elapsed_time = end_time - start_time
                print(elapsed_time)
                # Store data
                data["Algorithm Type"].append(algo_name)
                data["Time"].append(elapsed_time)
                data["Elements"].append(num_elements)
                data["Input Type"].append(input_type)

    return pd.DataFrame(data)

# Define parameters
num_elements_list = [20000, 40000, 60000, 80000, 100000]

# Measure sorting time
sorting_data = measure_sorting_time(algorithm, input_types,
num_elements_list)
```

```
sorting_data.to_excel("sorting_data_todos.xlsx", index=False)
```