Relatório Projeto 3 AED 2023/2024

Nome: Alexandre José Martins Rodrigues Nº Estudante:2022249408
PL (inscrição):1 Email: uc2022249408@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

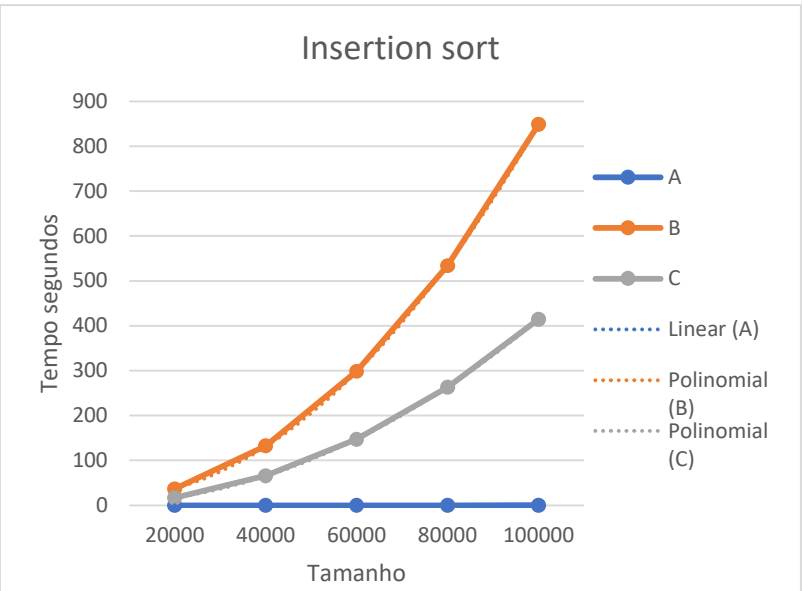
2. Recolha de Resultados *(tabelas)*

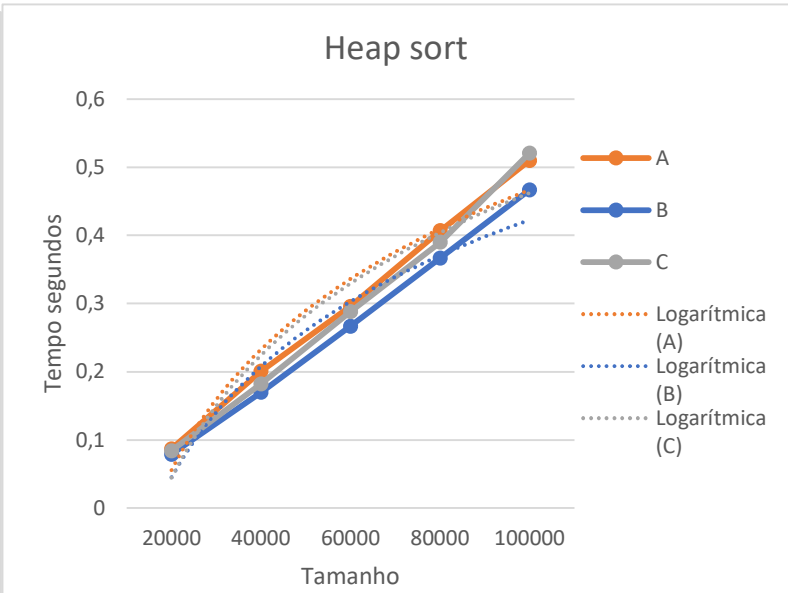
Tipo	Tamanho	Insertion Sort	Heap Sort	Quick Sort
A	20000	0,00100565	0,086993217	0,030970573
A	40000	0,004985571	0,200983286	0,06802845
A	60000	0,004997492	0,296034813	0,101968765
A	80000	0,005965471	0,407147408	0,149579763
A	100000	0,008040428	0,510010242	0,183989048

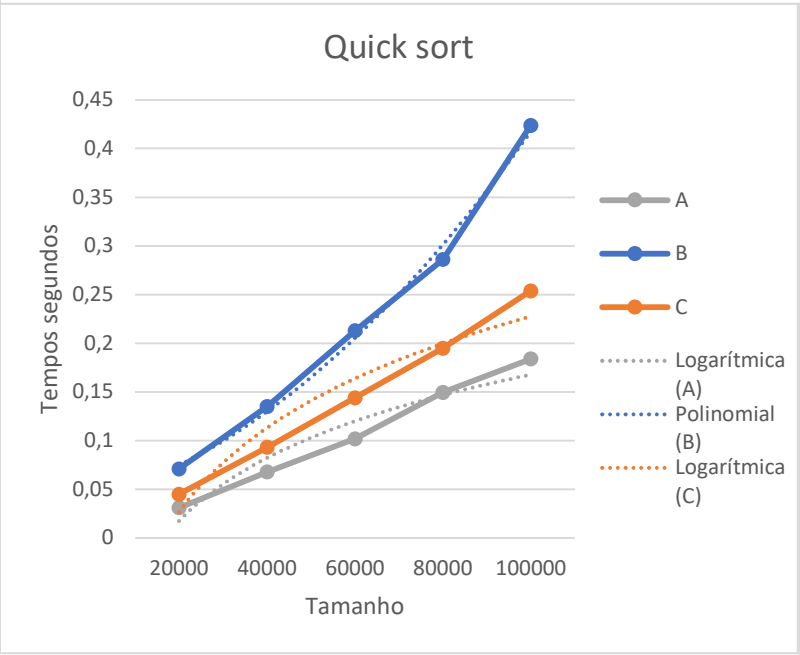
Tipo	Tamanho	Insertion Sort	Heap Sort	Quick Sort
B	20000	36,63428044	0,078998804	0,071000099
B	40000	132,4366863	0,170002699	0,134997368
B	60000	298,6911023	0,2668221	0,21299839
B	80000	534,1254988	0,366564512	0,286045551
B	100000	849,658716	0,467036486	0,423965693

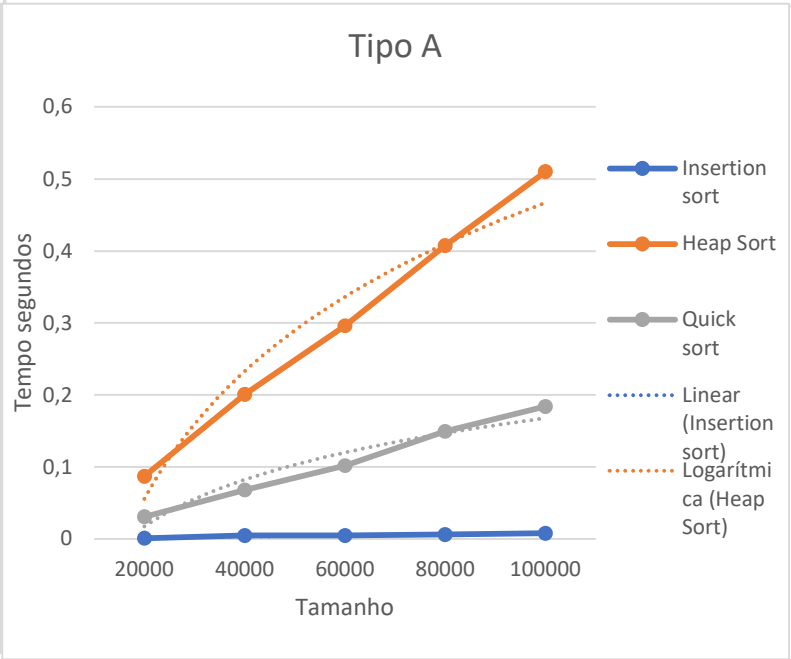
Tipo	Tamanho	Insertion Sort	Heap Sort	Quick Sort
C	20000	16,61526513	0,083995819	0,045000076
C	40000	65,94008517	0,181990623	0,093099117
C	60000	147,4109817	0,288005829	0,143998861
C	80000	263,4040155	0,390001059	0,194831848
C	100000	414,5831461	0,521003962	0,253997564

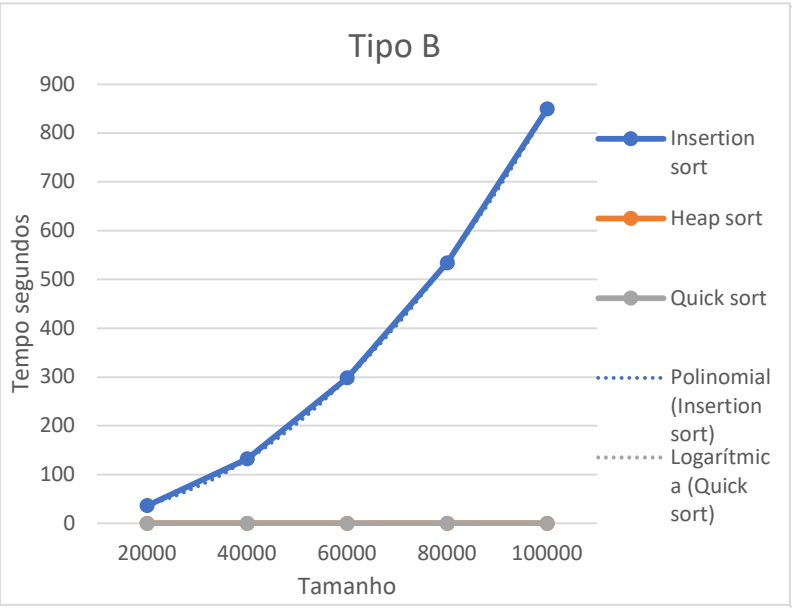
3. Visualização de Resultados *(gráficos)*

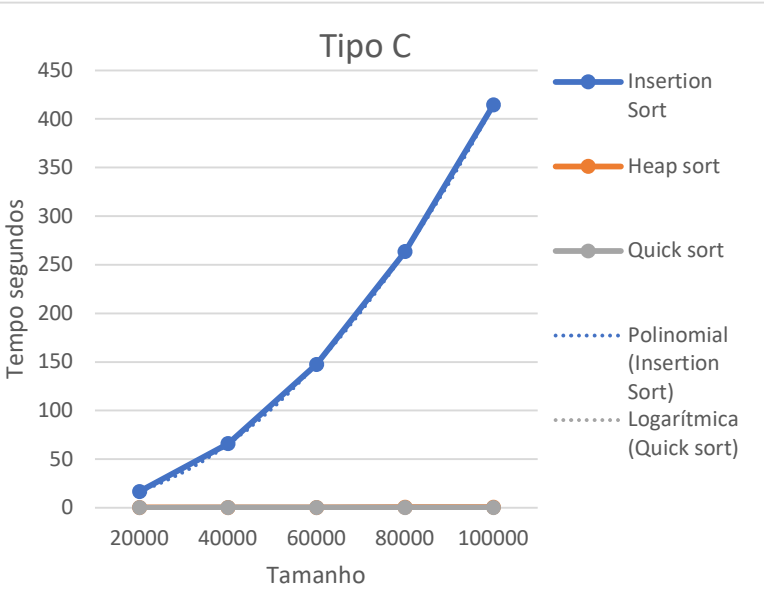












4. Conclusões *(as linhas desenhadas representam a extensão máxima de texto manuscrito)*

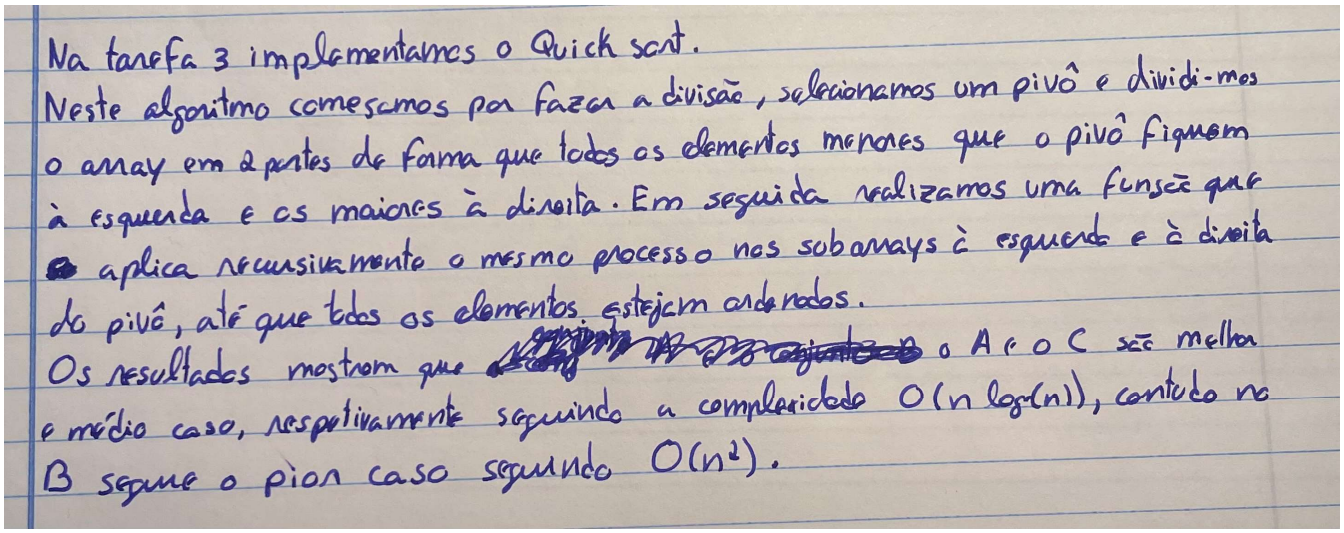
4.1 Tarefa 1

Na tarefa 1 implementamos o Insertion Sort.
Este funciona percorrendo a lista introduzida e compara cada elemento com os que já estão ordenados antes dele. No caso, quando o elemento é maior passa para a direita, quando é menor passa para a esquerda. Este processo é repetido até organizar todos os elementos.
Utilizando os 3 conjuntos obtivemos resultados bastantes distintos, no A a complexidade foi $O(n)$, melhor caso, no B e C, pior e médio caso, respetivamente, foi $O(n^2)$. Isto acontece pois no B tem de ordenar todos os elementos e no C é aleatório.

4.2 Tarefa 2

Na tarefa 2 implementamos o Heap sort.
Este começa por construir a heap a partir do conjunto de entrada, é transformado numa heap máxima, garantindo que o elemento em cada nó pai seja maior ou igual aos seus filhos. Em seguida realizamos a extração do valor máximo e a reorganização, onde o maior elemento é removido e colocado no final do array. Depois, a heap é arranjada para garantir que a propriedade da heap máxima seja mantida. Por fim os passos são repetidos até que todo o array esteja ordenado.
Cada iteração reduz o tamanho da heap em 1.
Ao nível de complexidade verificamos que é sempre $O(n \log(n))$, o n vem do número de elementos no conjunto e o $\log(n)$ vem da altura do heap. Nos resultados foram todos semelhantes.

4.3 Tarefa 3



Anexo A - Delimitação de Código de Autor

```
def criar_input(n, l):
    chaves_ord = []
    i = 0
    while i < n:
        a = random.randint(1, n)
        chaves_ord.append(a)
        i += 1
    if l == "A":
        return sorted(chaves_ord)
    if l == "B":
        return sorted(chaves_ord, reverse=True)
    if l == "C":
        return chaves_ord

lista[j] = lista[j-1]
            lista[j-1] = temp
            j -= 1

tipos = ["A", "B", "C"]
tamanhos = [20000, 40000, 60000, 80000, 100000]

data_dict = {'Tipo': [], 'Tamanho': [], 'Insertion Sort': [], 'Heap Sort':
[], 'Quick Sort': []}

for tipo in tipos:
    for tam in tamanhos:
        data_ord = criar_input(tam, tipo)
        leng =  len(data_ord)-1
        start_time_insertion = time.time()
        insertion_sort(data_ord[:],0,leng)
```

```
        end_time_insertion = time.time()
        elapsed_time_insertion = end_time_insertion - start_time_insertion

        start_time_heap = time.time()
        heapSort(data_ord[:])
        end_time_heap = time.time()
        elapsed_time_heap = end_time_heap - start_time_heap

        start_time_quick = time.time()
        quickSort(data_ord[:],0,leng)
        end_time_quick = time.time()
        elapsed_time_quick = end_time_quick - start_time_quick

        data_dict['Tipo'].append(tipo)
        data_dict['Tamanho'].append(tam)
        data_dict['Insertion Sort'].append(elapsed_time_insertion)
        data_dict['Heap Sort'].append(elapsed_time_heap)
        data_dict['Quick Sort'].append(elapsed_time_quick)
        print("feito",tipo,tam)
df = pd.DataFrame(data_dict)
df.to_excel('F2.xlsx', index=False)
```

Anexo B - Referências

```
def heapify(chaves_ord, tam, i):
    maior = i
    esquerda = 2 * i + 1
    direita = 2 * i + 2
    if esquerda < tam and chaves_ord[maior] < chaves_ord[esquerda]:
        maior = esquerda
    if direita < tam and chaves_ord[maior] < chaves_ord[direita]:
        maior = direita
    if maior != i:
        chaves_ord[i], chaves_ord[maior] = chaves_ord[maior],
chaves_ord[i]
        return heapify(chaves_ord, tam, maior)
    else:
        return chaves_ord

def heapSort(chaves_ord):
    tam = len(chaves_ord)

    for i in range(tam // 2 - 1, -1, -1):
        heapify(chaves_ord, tam, i)

    for i in range(tam - 1, 0, -1):
        chaves_ord[i], chaves_ord[0] = chaves_ord[0], chaves_ord[i]
        heapify(chaves_ord, i, 0)
```

```
    return chaves_ord

def troca(lista,a,b):
    temp = lista[a]
    lista[a]= lista[b]
    lista[b]= temp

def partition(lista, low, high):
    mid = (low+high)//2
    if lista[low]>lista[mid]:
        troca(lista,low,mid)
    if lista[mid]>lista[high]:
        troca(lista,mid,high)
        if lista[low]>lista[mid]:

troca(lista,mid,high-1)

i = low - 1
    for j in range(low, high-1):
        if lista[j] <= pivot:
            i = i + 1
            (lista[i], lista[j]) = (lista[j], lista[i])

    (lista[i + 1], lista[high]) = (lista[high], lista[i + 1])

    return i + 1
def quickSort(lista, low, high):
    if high-low>50:
        pi = partition(lista, low, high)
        quickSort(lista, low, pi - 1)
        quickSort(lista, pi, high)
    else:
        insertion_sort(lista,low,high)
```

chatgpt - ChatGPT (openai.com)

geeksforgeeks - GeeksforGeeks | A computer science portal for geeks

slides

Anexo C – Listagem Código

```
import random
import time
import pandas as pd


def criar_input(n, l):
```

```
    chaves_ord = []
    i = 0
    while i < n:
        a = random.randint(1, n)
        chaves_ord.append(a)
        i += 1
    if l == "A":
        return sorted(chaves_ord)
    if l == "B":
        return sorted(chaves_ord, reverse=True)
    if l == "C":
        return chaves_ord

def insertion_sort(lista,inicio,fim):
    for i in range(inicio,fim+1):
        j = i
        while lista[j]<lista[j-1] and j!=0:
            temp = lista[j]
            lista[j] = lista[j-1]
            lista[j-1] = temp
            j -= 1

maior = i
    esquerda = 2 * i + 1
    direita = 2 * i + 2
    if esquerda < tam and chaves_ord[maior] < chaves_ord[esquerda]:
        maior = esquerda
    if direita < tam and chaves_ord[maior] < chaves_ord[direita]:
        maior = direita
    if maior != i:
        chaves_ord[i], chaves_ord[maior] = chaves_ord[maior],
chaves_ord[i]
        return heapify(chaves_ord, tam, maior)
    else:
        return chaves_ord

def heapSort(chaves_ord):
    tam = len(chaves_ord)

    for i in range(tam // 2 - 1, -1, -1):
        heapify(chaves_ord, tam, i)

    for i in range(tam - 1, 0, -1):
        chaves_ord[i], chaves_ord[0] = chaves_ord[0], chaves_ord[i]
        heapify(chaves_ord, i, 0)

    return chaves_ord
```

```
def troca(lista,a,b):
    temp = lista[a]
    lista[a]= lista[b]
    lista[b]= temp

def partition(lista, low, high):
    mid = (low+high)//2
    if lista[low]>lista[mid]:
        troca(lista,low,mid)
    if lista[mid]>lista[high]:
        troca(lista,mid,high)
        if lista[low]>lista[mid]:
            troca(lista,low,mid)
    troca(lista,mid,high-1)
    pivot = lista[high-1]
    i = low - 1
    for j in range(low, high-1):
        if lista[j] <= pivot:
            i = i + 1
            (lista[i], lista[j]) = (lista[j], lista[i])

    (lista[i + 1], lista[high]) = (lista[high], lista[i + 1])

    return i + 1

def quickSort(lista, low, high):
    if high-low>50:
        pi = partition(lista, low, high)
        quickSort(lista, low, pi - 1)
        quickSort(lista, pi, high)
    else:
        insertion_sort(lista,low,high)

tipos = ["A", "B", "C"]
tamanhos = [20000, 40000, 60000, 80000, 100000]

data_dict = {'Tipo': [], 'Tamanho': [], 'Insertion Sort': [], 'Heap Sort':
[], 'Quick Sort': []}

for tipo in tipos:
    for tam in tamanhos:
        data_ord = criar_input(tam, tipo)
        leng =  len(data_ord)-1
        start_time_insertion = time.time()
        insertion_sort(data_ord[:],0,leng)
        end_time_insertion = time.time()
        elapsed_time_insertion = end_time_insertion - start_time_insertion
```

```
        start_time_heap = time.time()
        heapSort(data_ord[:])
        end_time_heap = time.time()
        elapsed_time_heap = end_time_heap - start_time_heap

        start_time_quick = time.time()
        quickSort(data_ord[:],0,leng)
        end_time_quick = time.time()
        elapsed_time_quick = end_time_quick - start_time_quick

        data_dict['Tipo'].append(tipo)
        data_dict['Tamanho'].append(tam)
        data_dict['Insertion Sort'].append(elapsed_time_insertion)
        data_dict['Heap Sort'].append(elapsed_time_heap)
        data_dict['Quick Sort'].append(elapsed_time_quick)
        print("feito",tipo,tam)
df = pd.DataFrame(data_dict)
df.to_excel('F2.xlsx', index=False)
```