Relatório Projeto 3 AED 2023/2024

Nome: João António Faustino Vaz Nº Estudante:2022231087
PL (inscrição): 8 Email: joao.vaz1810@gmail.com

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		**feito**			
Heap Sort		**feito**			
Quick Sort			**feito**		
Finalização Relatório				**incompleto**	**feito**

2. Recolha de Resultados *(tabelas)*

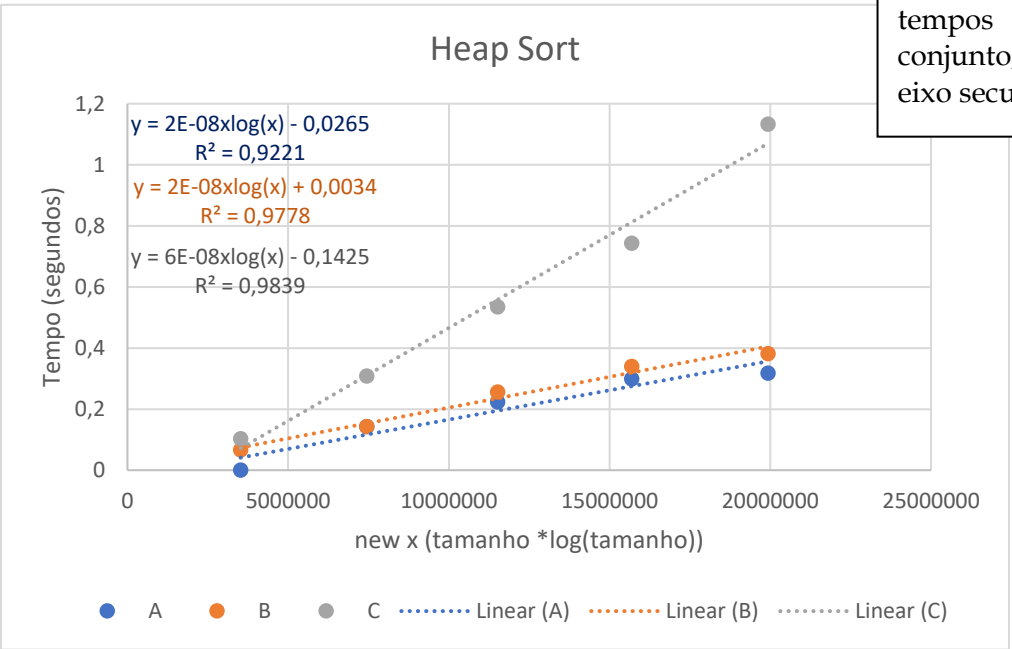
				heap	
tam	log tamanh	new x (z)	A	B	C
200000	17,60964	3521928	0,079981	0,066455	0,103106
400000	18,60964	7443856	0,143342	0,14281	0,308472
600000	19,1946	11516762	0,223274	0,256145	0,534804
800000	19,60964	15687712	0,299168	0,340274	0,743031
1000000	19,93157	19931569	0,318408	0,382181	1,1331

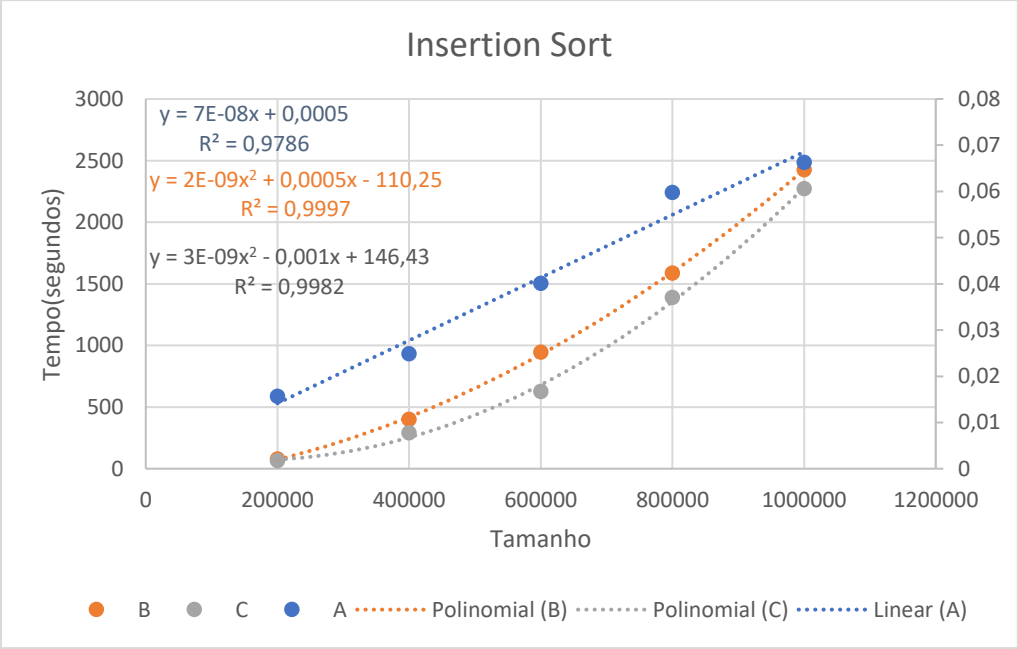
		insertion	
tam	A	B	C
200000	0,015696	79,8351	65,3864
400000	0,024866	399,93	290,85
600000	0,040149	946,66	625,63
800000	0,059764	1586,87	1390,39
1000000	0,066262	2425,82	2272,45

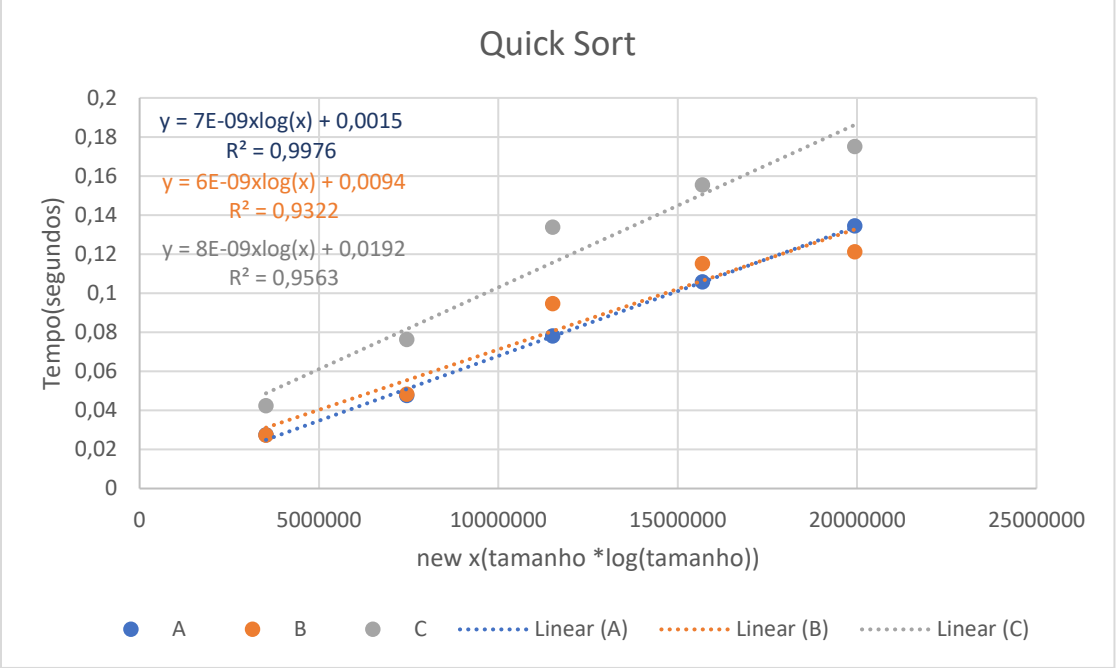
				quick	
tam	log tamanh	new x (z)	A	B	C
200000	17,60964	3521928	0,027369	0,027265	0,042342
400000	18,60964	7443856	0,047588	0,048145	0,076188
600000	19,1946	11516762	0,078083	0,094573	0,133719
800000	19,60964	15687712	0,105679	0,115047	0,155404
1000000	19,93157	19931569	0,13446	0,121169	0,175103

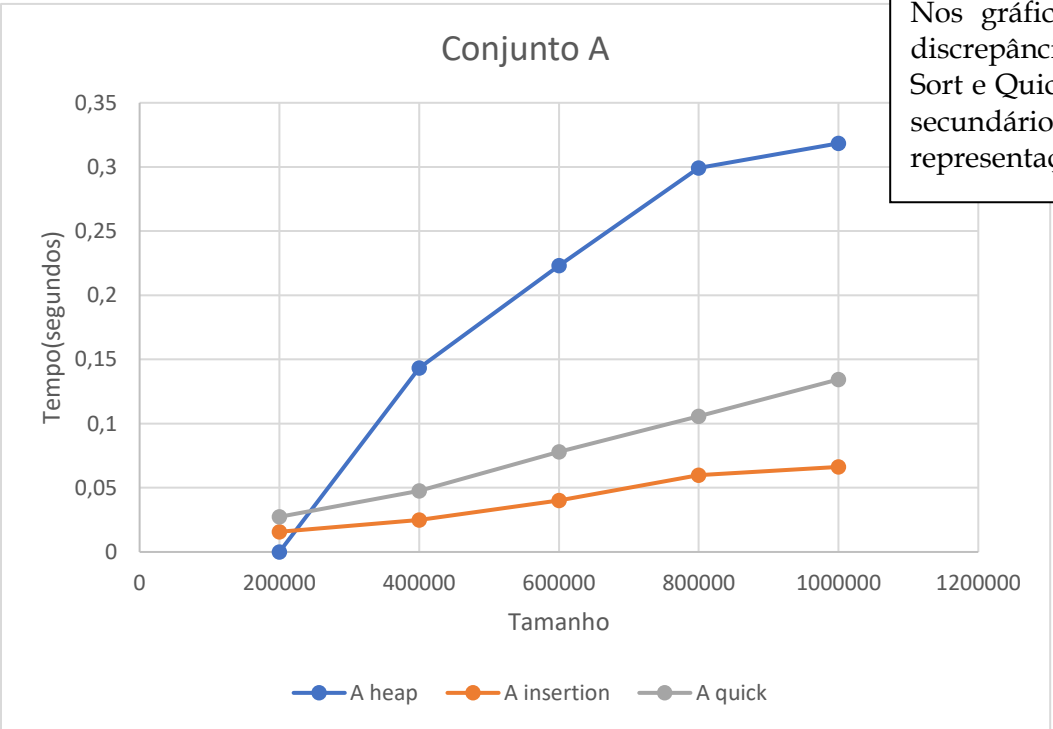
3. Visualização de Resultados *(gráficos)*

No gráfico do Insertion sort devido à discrepância dos tempos de conjunto para conjunto, o conjunto A usa um eixo secundário



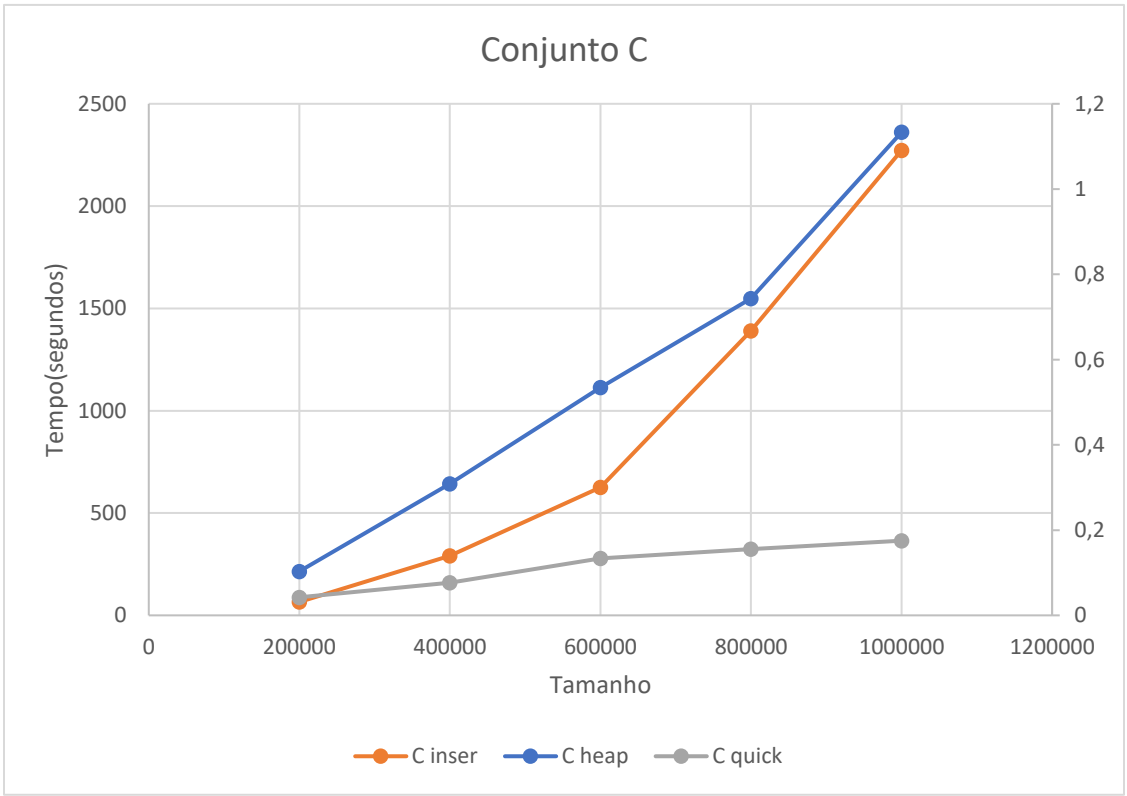






Nos gráficos B e C devido à discrepância dos tempos o Heap Sort e Quick sort usam um eixo secundário para uma melhor representação


Conjunto B
Tempo(segundos)
3000
2500
2000
1500
1000
500
0
0,45
0,4
0,35
0,3
0,25
0,2
0,15
0,1
0,05
0
0
200000
400000
600000
800000
1000000
1200000
Tamanho
B insert
B heap
B quick


Conjunto C
Tempo(segundos)
2500
2000
1500
1000
500
0
1,2
1
0,8
0,6
0,4
0,2
0
0
200000
400000
600000
800000
1000000
1200000
Tamanho
C inser
C heap
C quick

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

4.1 Tarefa 1 RESPONDIDO EM BAIXO

4.2 Tarefa 2

4.3 Tarefa 3

Anexo A - Delimitação de Código de Autor

Eu – Insertion sort e criação das chaves

Anexo B - Referências

Chatgpt – Heap sort

Proveniente do vídeo abaixo- Quick sort

https://www.youtube.com/watch?v=h8eyY7dIiN4&t=3s&pp=ygUScXVpY2sgc29ydCBpbiBqYXZh

Anexo C – Listagem Código

```java
public class Heap_sort {
    1 usage
    public static void heapSort(ArrayList<Integer> array) {
        int n = array.size();
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(array, n, i);
        for (int i = n - 1; i > 0; i--) {
            Collections.swap(array, i: 0, i);
            heapify(array, i, i: 0);
        }
    }
    3 usages
    public static void heapify(ArrayList<Integer> array, int n, int i) {
        int largest = i;
        int left = 2 * i + 1;
        int right = 2 * i + 2;

        if (left < n && array.get(left) > array.get(largest))
            largest = left;

        if (right < n && array.get(right) > array.get(largest))
            largest = right;

        if (largest != i) {
            Collections.swap(array, i, largest);
            heapify(array, n, largest);
        }
    }
```

```java
    public static void main(String[] args) {
        int size = 200000;
        ArrayList<Integer> teste=new ArrayList<>();
        for(int i=0;i<8;i++){
            teste.add(i);
        }
        ArrayList<Integer> input = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < size; i++) {
            input.add(i);
        }
        Collections.sort(input);
        ArrayList<Integer> inputb = new ArrayList<>(input);
        inputb.sort(Collections.reverseOrder());
        ArrayList<Integer> inputc = new ArrayList<>(inputb);
        Collections.shuffle(inputc);
        long ti=System.nanoTime();
        heapSort(inputb);
        long tf=System.nanoTime();
        System.out.println((tf-ti));
    }
}
```

```java
public class Insertion_sort {
    1 usage
    public static void insertionSort(ArrayList<Integer> array, int size) {
        for (int i = 1; i < size; ++i) {
            int value = array.get(i);
            int j = i - 1;
            while (j >= 0 && array.get(j) > value) {
                array.set(j + 1, array.get(j));
                j = j - 1;
            }
            array.set(j + 1, value);
        }
    }

    public static void main(String[] args) {
        int size = 200000;
        ArrayList<Integer> teste=new ArrayList<>();
        for(int i=0;i<8;i++){
            teste.add(i);
        }
        ArrayList<Integer> input = new ArrayList<>();
        for (int i = 0; i < size; i++) {
            input.add(i);
        }
        Collections.sort(input);
        ArrayList<Integer> inputb = new ArrayList<>(input);
        inputb.sort(Collections.reverseOrder());
        ArrayList<Integer> inputc = new ArrayList<>(inputb);
        Collections.shuffle(inputc);
        long ti=System.nanoTime();
        insertionSort(inputb, size);
        long tf=System.nanoTime();
        System.out.println((tf-ti));
    }
```

```java
public class Quick_sort {
    1 usage
    private static void quicksort(int[] array) {
        quicksort(array, lowIndex: 0, highIndex: array.length - 1);
    }
    3 usages
    private static void quicksort(int[] array, int lowIndex, int highIndex) {
        if (lowIndex >= highIndex) {
            return;
        }
        int pivotIndex = new Random().nextInt( bound: highIndex - lowIndex) + lowIndex;
        int pivot = array[pivotIndex];
        swap(array, pivotIndex, highIndex);
        int leftPointer = partition(array, lowIndex, highIndex, pivot);
        quicksort(array, lowIndex, highIndex: leftPointer - 1);
        quicksort(array, lowIndex: leftPointer + 1, highIndex);
    }
```

```java
    private static int partition(int[] array, int lowIndex, int highIndex, int pivot) {
        int leftPointer = lowIndex;
        int rightPointer = highIndex - 1;

        while (leftPointer < rightPointer) {
            while (array[leftPointer] <= pivot && leftPointer < rightPointer) {
                leftPointer++;
            }
            while (array[rightPointer] >= pivot && leftPointer < rightPointer) {
                rightPointer--;
            }

            swap(array, leftPointer, rightPointer);
        }
        if(array[leftPointer] > array[highIndex]) {
            swap(array, leftPointer, highIndex);
        }
        else {
            leftPointer = highIndex;
        }

        return leftPointer;
    }

    3 usages
    private static void swap(int[] array, int index1, int index2) {
        int temp = array[index1];
        array[index1] = array[index2];
        array[index2] = temp;
    }
```

```java
    public static void main(String[] args) {
        int size = 800000;
        ArrayList<Integer> teste=new ArrayList<>();
        for(int i=0;i<8;i++){
            teste.add(i);
        }
        ArrayList<Integer> input = new ArrayList<>();
        for (int i = 0; i < size; i++) {
            input.add(i);
        }
        ArrayList<Integer> inputb = new ArrayList<>(input);
        inputb.sort(Collections.reverseOrder());
        ArrayList<Integer> inputc = new ArrayList<>(inputb);
        Collections.shuffle(inputc);
        int[] arr = new int[size];
        for (int i = 0; i < size; i++) {
            arr[i] = inputc.get(i);
        }
        long ti = System.nanoTime();
        quicksort(arr);
        long tf = System.nanoTime();
        System.out.println(tf - ti);
    }
```

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

4.1 Tarefa 1

A complexidade para o conjunto A no insertion sort é O(n) porque percorremos o array todo (n) através do for e nunca se verifica a condição do while porque os elementos estão por ordem crescente. Nos conjuntos B e C a complexidade é $O(n^2)$. No B devido aos elementos estarem por ordem decrescente a condição do while é sempre verdadeira e o array é percorrido novamente (n^2). No C, em que os elementos estão distribuídos aleatoriamente pode ser necessário variar comparações e verificar-se a condição do while (n^2)

4.2 Tarefa 2

A complexidade do Heap Sort é O(nlogn) porque o algoritmo converte ~~e~~ o array a ordenar numa arvore heap binária e em que a altura corresponde a logn e portanto para n elementos, a complexidade é O(nlogn). Este algoritmo, no geral, podemos ver que é mais rapido que o insertion Sort e mais lento que o Quick Sort.

4.3 Tarefa 3

A complexidade do Quick Sort é O(nlogn). A escolha do pivot é algo importante neste algoritmo. A escolha de um pivot fixo, como o primeiro, ultimo ou do meio pode levar a um mau desempenho se o

array já estiver ordenado, então eu escolhi de forma aleatória o pivot pois apresenta um melhor desempenho médio. A complexidade deste algoritmo é O(n log n) sendo n o número de elementos e log n corresponde ao tamanho da árvore que resulta das partições.

Anexo A - Delimitação de Código de Autor

Eu – Insertion sort e criação das chaves

Anexo B - Referências

Cha[redacted] sort

Proven[redacted] abaixo- Quick sort

https://www[redacted]/watch?v=h8eyY7dIiN4&t=3s&pp=ygUScXVp[redacted]pbiBqYXZh

Anexo C – Listagem Código

```
public class Heap_sort {
    1 usage
    public static void heapSort(ArrayList<Integer> ar
        int n = array.size();
        for (int i = n / 2 - 1; i >= 0; i--)
            heapify(array, n, i);
        for (int i = n - 1; i > 0; i--) {
            Collections.swap(array,  0,
            heapify(array, i,  0);
        }
    }
    3 usages
    public static void heap                Integer> array, int n, int i)
        int largest = i;
        int left = 2
        int right

        if (left <  ray.get(left) > array.get(largest))
            largest = left;

        if (right < n && array.get(right) > array.get(largest))
            largest = right;

        if (largest != i) {
            Collections.swap(array, i, largest);
            heapify(array, n, largest);
        }
    }
```

```
    public static void main(String[] args) {
        int size = 200000;
        ArrayList<Integer> teste=new ArrayList<>();
        for(int i=0;i<8;i++){
            teste.add(i);

            ist<Integer> input = new ArrayList<>();
            ndom = new Random();
             0; i < size; i++) {
            );
        }
        Collec                t);
        ArrayList             b = new ArrayList<>(input);
        inputb.sort(          erseOrder());
        ArrayList<Integ                new ArrayList<>(inputb);
        Collections.shuffl            ;
        long ti=System.nanoTime();
        heapSort(inputb);
        long tf=System.nanoTime();
        System.out.println((tf-ti));
    }
}
```