HACETTEPE UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BBM204 SOFTWARE PRACTICUM II -2024 SPRING

PROGRAMMING ASSIGNMENT 1

MARCH 20, 2024

Student Name:	Student Number:
İbrahim Emek	b2210356032

1 Problem Definition

We try to sort the data, and search for an element in data using algorithms, and compare time complexity, and space complexity when the size of data increases.

2 Solution Implementation

First, we need to implement algorithms.

1- Insertion sort:

```
public static void insertionSort(int[] array)
{
    for (int elementAt = 1; elementAt < array.length; elementAt++)
    {
        int currentElement = array[elementAt];
        int comparisonAt = elementAt - 1;
        while (true)
        {
            if (comparisonAt < 0 || array[comparisonAt] <= currentElement)
                break;
            array[comparisonAt + 1] = array[comparisonAt];
            comparisonAt--;
        }
        array[comparisonAt + 1] = currentElement;
    }
}
```

2- Merge sort:

```
public static int[] mergeSort(int[] array)
{
    int arrayLength = array.length;
    if (arrayLength <= 1) return array;
    int[] leftArr = Arrays.copyOf(array, arrayLength / 2);
    int[] rightArr = Arrays.copyOfRange(array, arrayLength / 2, arrayLength);
    leftArr = mergeSort(leftArr);
    rightArr = mergeSort(rightArr);
    return merge(leftArr, rightArr);
}
public static int[] merge(int[] array1, int[] array2)
{
    int firstAt = 0, secondAt = 0, mergedArrAt = 0;
    int[] mergedArr = new int[array1.length + array2.length];

    while (firstAt < array1.length && secondAt < array2.length)
    {
        if (array1[firstAt] > array2[secondAt])
        {
            mergedArr[mergedArrAt] = array2[secondAt];
```

```
            secondAt++;
        }
        else
        {
            mergedArr[mergedArrAt] = array1[firstAt];
            firstAt++;
        }
        mergedArrAt++;
    }
    while (firstAt < array1.length)
    {
        mergedArr[mergedArrAt] = array1[firstAt];
        firstAt++;
        mergedArrAt++;
    }
    while (secondAt < array2.length)
    {
        mergedArr[mergedArrAt] = array2[secondAt];
        secondAt++;
        mergedArrAt++;
    }
    return mergedArr;

}
```

3- Counting sort

```
public static int[] countingSort(int[]arr, int maxEl)
{
    int[] countArray = new int[maxEl + 1];
    int arrSize = arr.length;
    int[] outputArray = new int[arrSize];
    for (int i = 0; i < arrSize; i++)
    {
        int key = arr[i];
        countArray[key]++;
    }
    for (int i = 1; i < maxEl + 1; i++)
    {
        countArray[i] += countArray[i - 1];
    }
    for (int i = arrSize; i > 0; i--)
    {
        int key = arr[i - 1];
        countArray[key]--;
        outputArray[countArray[key]] = arr[i - 1];
    }
    return outputArray;
}
```

4- Linear search

```java
public static int linearSearch(int[] arr, int value)
{
    for (int i = 0; i < arr.length; i++)
    {
        if (arr[i] == value)
            return i;
    }
    return -1;
}
```

5- Binary search

```java
public static int binarySearch(int[] arr, int value)
{
    int low = 0, high = arr.length - 1;
    while (high - low > 1)
    {
        int mid = (high + low) / 2;
        if (arr[mid] < value)
        {
            low = mid + 1;
        }
        else {
            high = mid;
        }
    }
    if (arr[low] == value)
        return low;
    else if (arr[high] == value) {
        return high;
    }
    return -1;

}
```

Now, we should read csv file, and convert the data to array. I used a function:

```java
public static ArrayList<Integer> getFlowDuration(int columnCount, String
fileName) throws IOException {
    BufferedReader reader = new BufferedReader(new FileReader(fileName));
    reader.readLine();
    String row;
    ArrayList<Integer> allFlowDuration = new ArrayList<>();
    int count = 0;
    while ((row = reader.readLine()) != null && count < columnCount)
    {
        String[] allElements = row.split(",");
        allFlowDuration.add(Integer.parseInt(allElements[6]));
        count++;
    }
    return allFlowDuration;
}
```

We read the file according to input column count (which will be how many data should we store), and file name to read. Then we store the data.

Now we should use algorithms, and calculate runtimes, and save them.

```java
public static void main(String args[]) throws IOException {
    ArrayList<Integer> allFlows = new ArrayList<>();

    // X axis data
    int[] inputAxis = {500, 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000, 250000};

    double[][] yAxis = new double[3][10];
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int[] flowArray = convertToArray(allFlows);
        long time1 = System.currentTimeMillis();
        insertionSort(flowArray);
        long time2 = System.currentTimeMillis();
        yAxis[0][i] = time2 - time1;
        System.out.println("Insertion random " + inputAxis[i] + " element. Total time: " + (time2 - time1));
    }
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int[] flowArray = convertToArray(allFlows);
        long time1 = System.currentTimeMillis();
        mergeSort(flowArray);
        long time2 = System.currentTimeMillis();
        yAxis[1][i] = time2 - time1;
        System.out.println("Merge random " + inputAxis[i] + " element. Total time: " + (time2 - time1));

    }
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int max = allFlows.get(0);
        for (int flow : allFlows)
        {
            if (flow > max)
                max = flow;
        }
        int[] flowArray = convertToArray(allFlows);

        long time1 = System.currentTimeMillis();
        countingSort(flowArray, max);
        long time2 = System.currentTimeMillis();
        yAxis[2][i] = time2 - time1;
        System.out.println("Counting random  " + inputAxis[i] + " element. Total time: " + (time2 - time1));

    }
    showAndSaveChart("Random Sort", inputAxis, yAxis);

    double[][] yAxisSorted = new double[3][10];
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int[] flowArray = convertToArray(allFlows);
```

```java
        insertionSort(flowArray);
        long time1 = System.currentTimeMillis();
        insertionSort(flowArray);
        long time2 = System.currentTimeMillis();
        yAxisSorted[0][i] = time2 - time1;
        System.out.println("Insertion sorted " + inputAxis[i] + " element. 
Total time: " + (time2 - time1));

    }
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int[] flowArray = convertToArray(allFlows);
        mergeSort(flowArray);
        long time1 = System.currentTimeMillis();
        mergeSort(flowArray);
        long time2 = System.currentTimeMillis();
        yAxisSorted[1][i] = time2 - time1;
        System.out.println("Merge sorted " + inputAxis[i] + " element. Total 
time: " + (time2 - time1));
    }
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int max = allFlows.get(0);
        for (int flow : allFlows)
        {
            if (flow > max)
                max = flow;
        }
        int[] flowArray = convertToArray(allFlows);
        countingSort(flowArray, max);
        long time1 = System.currentTimeMillis();
        countingSort(flowArray, max);
        long time2 = System.currentTimeMillis();
        yAxisSorted[2][i] = time2 - time1;
        System.out.println("Counting sorted " + inputAxis[i] + " element. 
Total time: " + (time2 - time1));

    }
    showAndSaveChart("Sorted Sort", inputAxis, yAxisSorted);


    double[][] yAxisReversed = new double[3][10];
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int[] flowArray = convertToArray(allFlows);
        insertionSort(flowArray);
        int[] newArray = reverseArray(flowArray);
        long time1 = System.currentTimeMillis();
        insertionSort(newArray);
        long time2 = System.currentTimeMillis();
        yAxisReversed[0][i] = time2 - time1;
        System.out.println("Insertion reversed " + inputAxis[i] + " element. 
Total time: " + (time2 - time1));
```

```java
    }
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int[] flowArray = convertToArray(allFlows);
        mergeSort(flowArray);
        int[] newArray = reverseArray(flowArray);
        long time1 = System.currentTimeMillis();
        mergeSort(newArray);
        long time2 = System.currentTimeMillis();
        yAxisReversed[1][i] = time2 - time1;
        System.out.println("Merge reversed " + inputAxis[i] + " element.
Total time: " + (time2 - time1));

    }
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        int max = allFlows.get(0);
        for (int flow : allFlows)
        {
            if (flow > max)
                max = flow;
        }
        int[] flowArray = convertToArray(allFlows);
        countingSort(flowArray, max);
        int[] newArray = reverseArray(flowArray);
        long time1 = System.currentTimeMillis();
        countingSort(newArray, max);
        long time2 = System.currentTimeMillis();
        yAxisReversed[2][i] = time2 - time1;
        System.out.println("Counting reversed " + inputAxis[i] + " element.
Total time: " + (time2 - time1));

    }
    showAndSaveChart("Reversed Sort", inputAxis, yAxisReversed);

    double[][] yAxisSearch = new double[3][10];
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);
        Random random = new Random();
        int randomNumber = random.nextInt(inputAxis[i]);
        System.out.println(randomNumber);
        int[] flowArray = convertToArray(allFlows);
        long time1 = System.nanoTime();
        linearSearch(flowArray, flowArray[randomNumber]);
        long time2 = System.nanoTime();
        yAxisSearch[0][i] = time2 - time1;
        System.out.println("Linear random " + inputAxis[i] + " element. Total
time: " + (time2 - time1));
    }
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);

        Random random = new Random();
```

```
        int randomNumber = random.nextInt(inputAxis[i]);
        System.out.println(randomNumber);

        int[] flowArray = convertToArray(allFlows);
        insertionSort(flowArray);
        long time1 = System.nanoTime();
        linearSearch(flowArray, flowArray[randomNumber]);
        long time2 = System.nanoTime();
        yAxisSearch[1][i] = time2 - time1;
        System.out.println("Linear sorted " + inputAxis[i] + " element. Total
time: " + (time2 - time1));
    }
    for (int i = 0; i < 10; i++)
    {
        allFlows = getFlowDuration(inputAxis[i], args[0]);

        Random random = new Random();
        int randomNumber = random.nextInt(inputAxis[i]);
        System.out.println(randomNumber);
        int[] flowArray = convertToArray(allFlows);
        insertionSort(flowArray);
        long time1 = System.nanoTime();
        binarySearch(flowArray, flowArray[randomNumber]);
        long time2 = System.nanoTime();
        yAxisSearch[2][i] = time2 - time1;
        System.out.println("Binary sorted " + inputAxis[i] + " element. Total
time: " + (time2 - time1));
    }
    showAndSaveChartSearch("Search Algorithms", inputAxis, yAxisSearch);


}
```

Between time1 and time2 we use algorithms. Then we subtract time1 from time2, and we reach runtimes.

For per 3 algorithm, we create a table using showAndSaveChart function.

These functions are the other functions:

```
public static int[] reverseArray(int[] arrayToReverse)
{
    int[] reversedArray = new int[arrayToReverse.length];
    for (int i = 0; i < arrayToReverse.length; i++)
    {
        reversedArray[i] = arrayToReverse[arrayToReverse.length - i - 1];
    }
    return reversedArray;
}
public static int[] convertToArray(ArrayList<Integer> toConvert)
{
    int[] y = new int[toConvert.size()];
    for (int a = 0; a < toConvert.size(); a++)
    {
        y[a] = toConvert.get(a);
```

```java
    }
    return y;
}



public static void showAndSaveChart(String title, int[] xAxis, double[][]
yAxis) throws IOException {
    // Create Chart
    XYChart chart = new XYChartBuilder().width(800).height(600).title(title)
            .yAxisTitle("Time in Milliseconds").xAxisTitle("Input
Size").build();

    // Convert x axis to double[]
    double[] doubleX = Arrays.stream(xAxis).asDoubleStream().toArray();

    // Customize Chart
    chart.getStyler().setLegendPosition(Styler.LegendPosition.InsideNE);

chart.getStyler().setDefaultSeriesRenderStyle(XYSeries.XYSeriesRenderStyle.Li
ne);

    // Add a plot for a sorting algorithm
    chart.addSeries("insertion sort", doubleX, yAxis[0]);
    chart.addSeries("merge sort", doubleX, yAxis[1]);
    chart.addSeries("counting sort", doubleX, yAxis[2]);


    // Save the chart as PNG
    BitmapEncoder.saveBitmap(chart, title + ".png",
BitmapEncoder.BitmapFormat.PNG);

    // Show the chart
    new SwingWrapper(chart).displayChart();
}
public static void showAndSaveChartSearch(String title, int[] xAxis,
double[][] yAxis) throws IOException {
    // Create Chart
    XYChart chart = new XYChartBuilder().width(800).height(600).title(title)
            .yAxisTitle("Time in Milliseconds").xAxisTitle("Input
Size").build();

    // Convert x axis to double[]
    double[] doubleX = Arrays.stream(xAxis).asDoubleStream().toArray();

    // Customize Chart
    chart.getStyler().setLegendPosition(Styler.LegendPosition.InsideNE);

chart.getStyler().setDefaultSeriesRenderStyle(XYSeries.XYSeriesRenderStyle.Li
ne);

    // Add a plot for a sorting algorithm
    chart.addSeries("Linear random ", doubleX, yAxis[0]);
    chart.addSeries("Linear sorted ", doubleX, yAxis[1]);
    chart.addSeries("Binary sorted ", doubleX, yAxis[2]);
```

```
    // Save the chart as PNG
    BitmapEncoder.saveBitmap(chart, title + ".png",
BitmapEncoder.BitmapFormat.PNG);

    // Show the chart
    new SwingWrapper(chart).displayChart();
}
```

A function to reverse an array, a function to convert arraylist to array, and functions to create tables.

3 Results, Analysis, Discussion

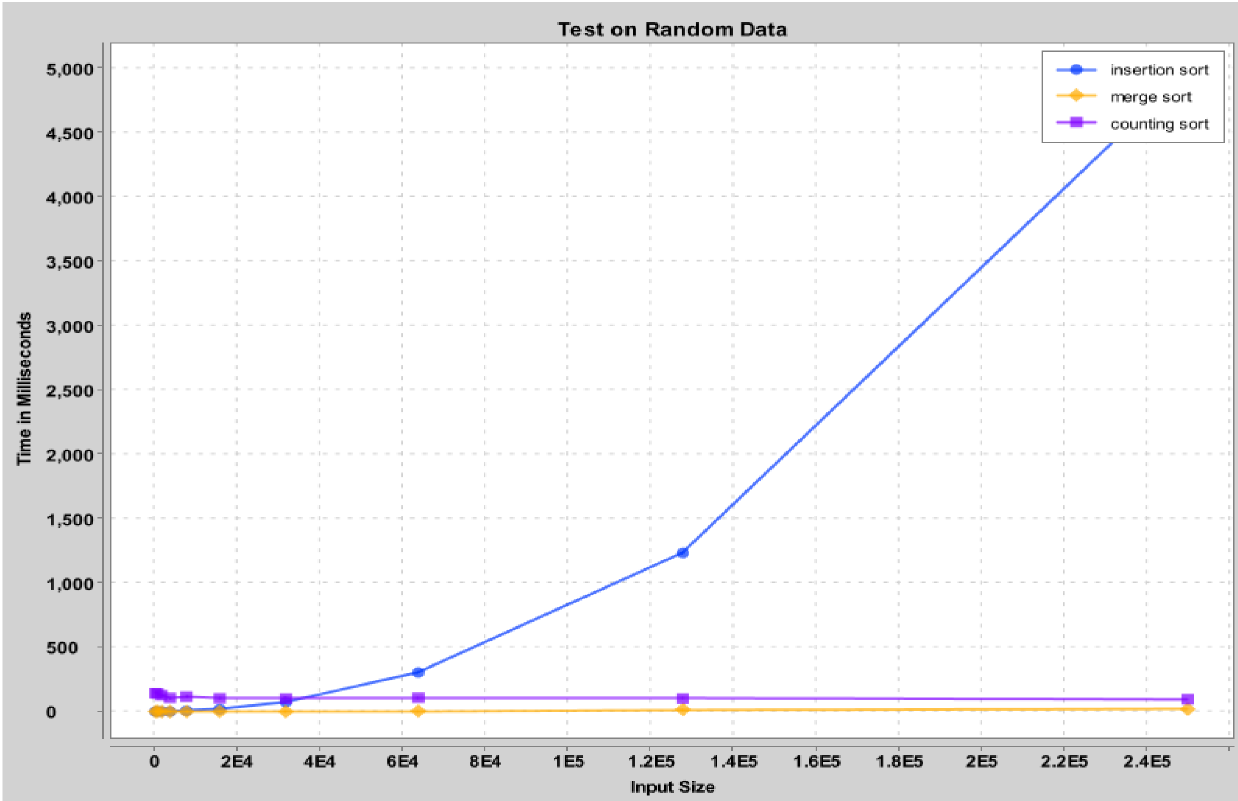
Algorithm	500	1000	2000	4000	8000	16000	32000	64000	128000	256000
Random Input Data Timing Results in ms										
Insertion sort	0	0	1	3	4	21	71	300	1190	4822
Merge sort	0	1	0	0	0	3	2	3	14	23
Counting sort	101	99	102	102	100	92	91	89	98	92
Sorted Input Data Timing Results in ms										
Insertion sort	0	0	0	0	0	0	0	0	0	0
Merge sort	0	0	1	3	0	0	4	6	12	20
Counting sort	81	84	88	80	86	85	85	90	84	88
Reversely Sorted Input Data Timing Results in ms										
Insertion sort	0	0	0	4	12	36	142	592	2290	8694
Merge sort	0	0	0	0	0	0	5	2	12	21
Counting sort	102	100	102	101	98	91	95	90	99	98

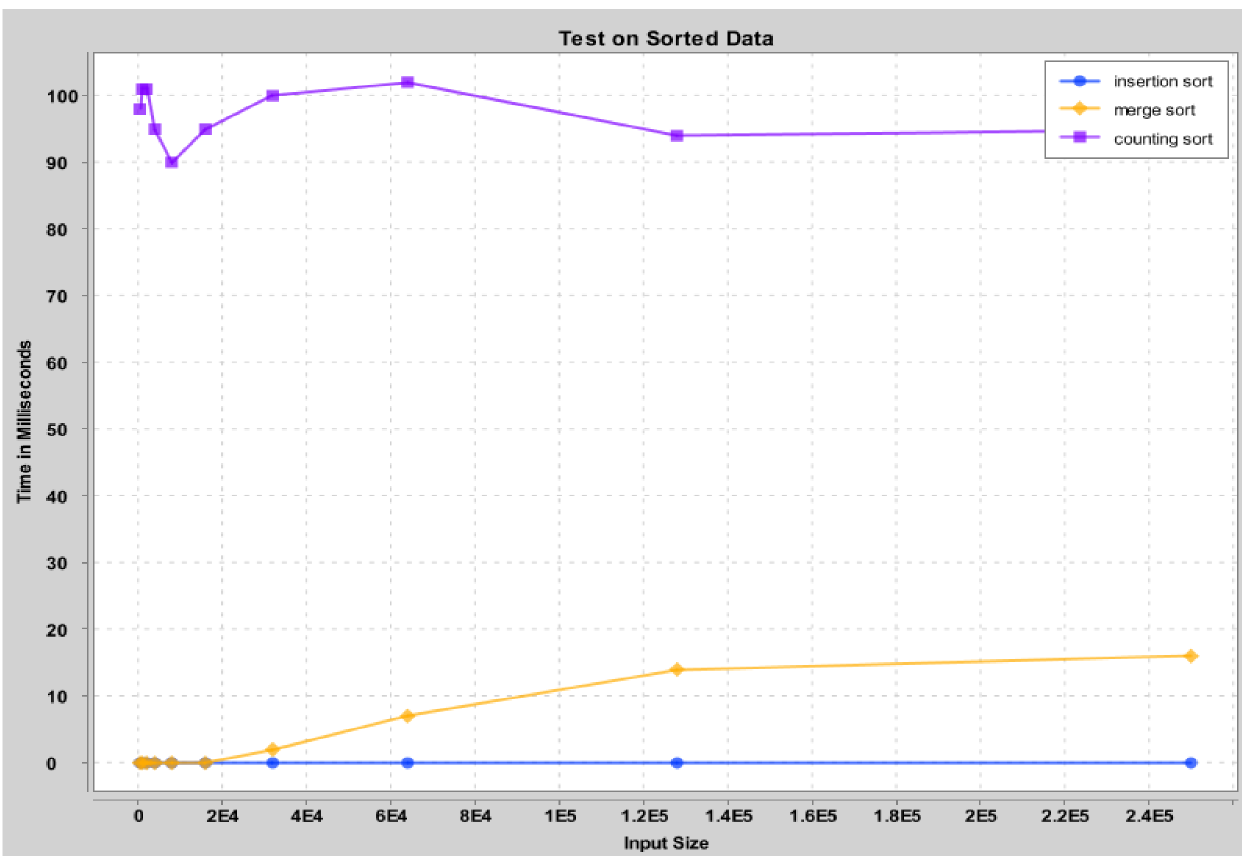
Algorithm	Best Case	Average Case	Worst Case
Insertion Sort	O(n)	O(n2)	O(n2)
Merge Sort	O(n log(n))	O(n log(n))	O(n log(n))
Counting Sort	O(n +k)	O(n +k)	O(n +k)

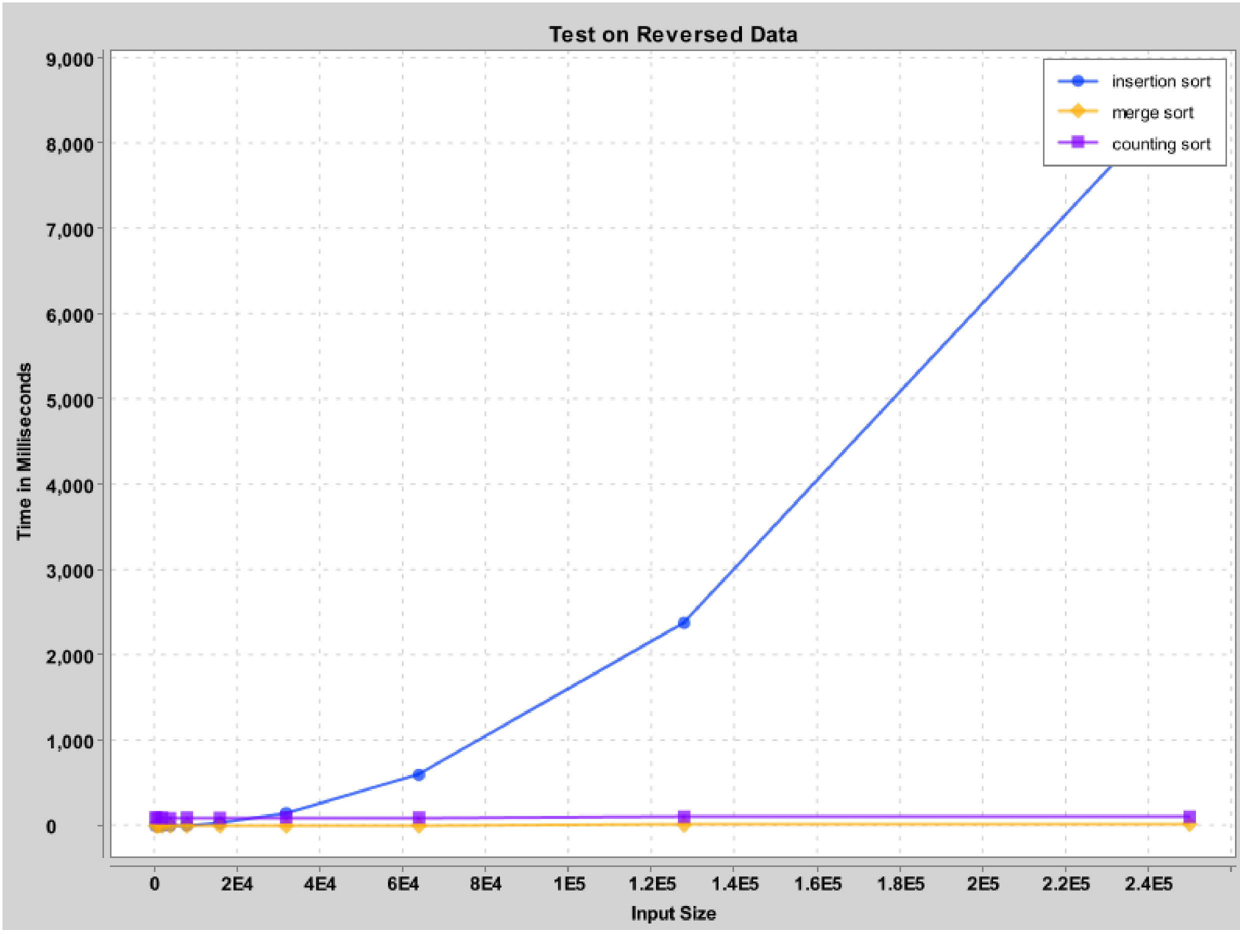
First, let's look at insertion sort. In best case (In case of sorted data) time complexity is O(n), and we see all of them happened in 0 ms, because it is fast in this data. If we look at average, and worst case, we can see the difference. In small data, it is very small number, but if we look at bigger than 32000, we can see when we increase data size 2 times, runtime increases 4 time. As a result we can say that it is O(n*n) in worst case, and average case.

If we look to merge sort, it is same for all cases, and if we look at the table, they almost same in all cases. Because the data is small to see the difference clearly, I can't say if it's time complexity is O(nlog(n)), but we can say it is same for all cases.

It is same case for counting sort as merge sort. We can only say it is same for all cases.


Test on Random Data
insertion sort
merge sort
counting sort
Time in Milliseconds
0
500
1,000
1,500
2,000
2,500
3,000
3,500
4,000
4,500
5,000
0
2E4
4E4
6E4
8E4
1E5
1.2E5
1.4E5
1.6E5
1.8E5
2E5
2.2E5
2.4E5
Input Size


Test on Sorted Data
insertion sort
merge sort
counting sort
Time in Milliseconds
0
10
20
30
40
50
60
70
80
90
100
0
2E4
4E4
6E4
8E4
1E5
1.2E5
1.4E5
1.6E5
1.8E5
2E5
2.2E5
2.4E5
Input Size

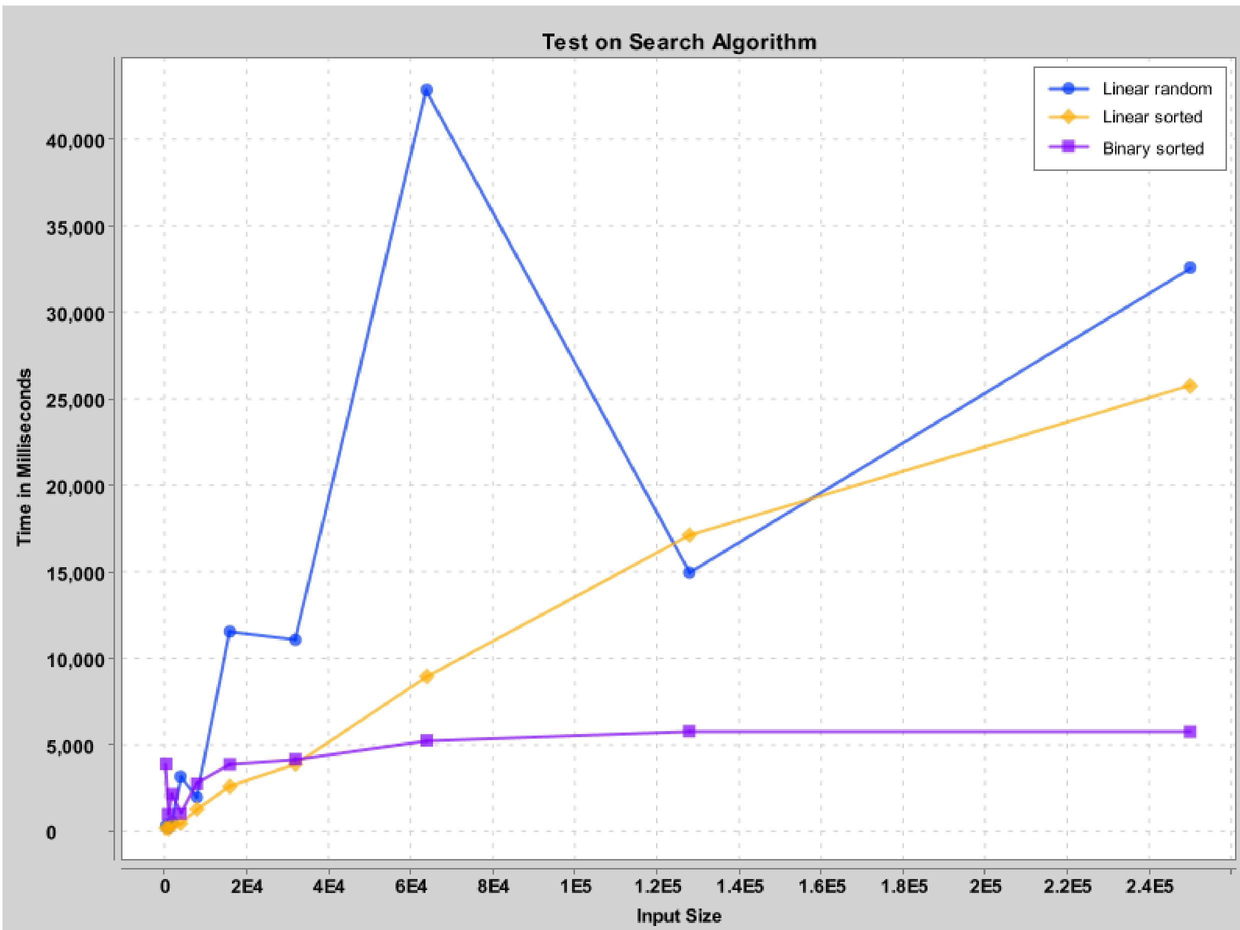


Search type	Time Complexity
Linear Search	O(n)
Binary Search	O(log(n))

Algorithm	500	1000	2000	4000	8000	16000	32000	64000	128000	256000
Search Algorithm Results in nanosec										
Linear Search (random data)	370	240	570	3200	2000	11590	11100	42890	14990	32620
Linear Search (sorted data)	230	190	420	500	1320	2630	3930	8990	17160	25780
Binary Search (sorted data)	3940	1030	2190	1080	2790	3910	4200	5290	5820	5790

In linear search, runtime only depends on data size. Generally, if the size increases, the runtime increases too. It is not the case always, because we pick a random number and search it. It can be first element, and it can be last one, so we can’t say it is certain that if data increases, the runtime increases too.

In binary search, it is same as linear search. Only difference is it increases less as a result of O(log(n)) time complexity.



Algorithm type	Space Complexity
Insertion Sort	O(1)
Merge Sort	O(n)
Counting Sort	O(k)
Linear Search	O(1)
Binary Search	O(1)

Since there is no data stored in insertion sort, linear search, and binary search, their space complexities are O(1). In merge sort we store new data in the new array, space complexity is O(n). Because counting creates k size array, it’s space complexity is O(k).

4 Conclusion

In this assignment, we explored various sorting, and searching algorithms, and analyzed their performance in terms of time, and space complexity while data size increases. While we doing this, we learned their efficiencies in different scenarios.

In conclusion, we should choose an algorithm to use according to data size, available memory, and runtime with our data size. Understanding the time and space complexity will provide us efficient apps.

References

1- Time Complexities of all Sorting Algorithms - GeeksforGeeks
2- All Sorting Algorithm, Its Time Complexities & Space Complexities. | by Salveketan | Medium
3- BBM204_Programming_Assignment_Report_Template.pdf (hacettepe.edu.tr)