Practical 1:

To implement Time Analysis of sorting algorithms.

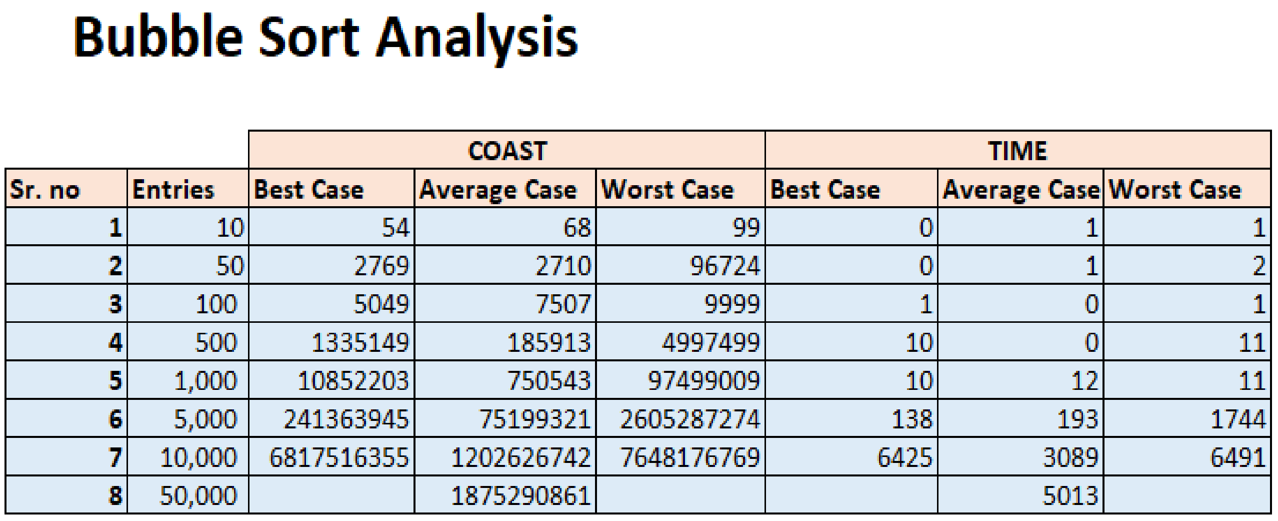
1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Merge Sort
5. Quick Sort

Code Implementation::

```
package Practical_Package;

import java.io.File;
import java.io.PrintWriter;
import java.util.*;

public class P1_TimeAnalysisSortingAlgorithm {

    public static void main(String[] args) {

        ArrayList<Integer> arrayList = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        String answer;
        long cost = 0;

        do {
            System.out.print("\nGive the file Name of Random Numbers\nEg.
[1_Digit_100_entries.txt] :: ");
            String fName = sc.next();

            File sourceFile = new File(fName);

            //if file doesn't exists exit
            if (!sourceFile.exists()) {
                System.out.println("\nCan't Locate " + fName);
                System.exit(1);
            }

            System.out.print("\nEnter the File name you want to store the Result\nEx.
[1_Digit_100_sorted_entries.txt] :: ");
            fName = sc.next();

            File destFile = new File(fName);

            //if file exists exit
            if (destFile.exists()) {
                System.out.println("\nDuplicate File names :(");
                System.exit(2);
            }

            try (
                    Scanner input = new Scanner(sourceFile);
                    PrintWriter output = new PrintWriter(destFile)
            ) {
                //fetch data from source file to arrayList
                while (input.hasNext())
                    arrayList.add(input.nextInt());

//                int[] arr = new int[arrayList.size()];
//                for (int i = 0; i < arrayList.size(); i++)
//                    arr[i] = arrayList.get(i);
```

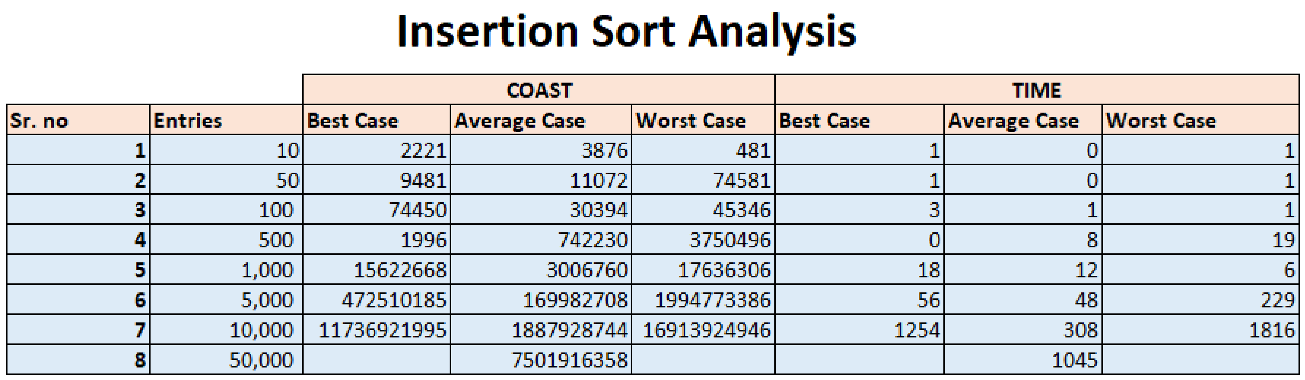
```
                //reversing sorting for worst case analysis
                int[] arr = new int[arrayList.size()];
                for (int i = arrayList.size() - 1, j = 0; i >= 0; i--, j++)
                    arr[j] = arrayList.get(i);

                System.out.println("\n1. Bubble Sort\n2. Insertion Sort\n3. Selection
Sort\n4. Merge Sort \n5. Quick Sort\nElse. Exit");
                System.out.print("Enter your Choice for sorting technique :: ");
                int sortChoice = sc.nextInt();

                long startTime = System.currentTimeMillis();

                switch (sortChoice) {
                    case 1 -> cost = bubbleSort(arr);
                    case 2 -> cost = insertionSort(arr);
                    case 3 -> cost = selectionSort(arr);
                    case 4 -> cost = mergeSort(arr, 0, arr.length - 1);
                    case 5 -> cost = quickSort(arr, 0, arr.length - 1);
                    default -> {
                        System.out.println("\nExiting...");
                        System.exit(3);
                    }
                }

                System.out.println("\nSorting Data...");
                System.out.println("Calculating the Cost...");

                for (int j : arr) output.println(j);

                long endTime = System.currentTimeMillis();

                System.out.println("\nCost for sorting :: " + cost);
                System.out.println("Time Taken :: " + (endTime - startTime) + "
milliseconds");

            } catch (Exception e) {
                e.printStackTrace();
            }

            //prompt user if he wants to continue or not.
            System.out.print("\nDo you want to continue ? [y/n] ");
            answer = sc.next().toLowerCase();

        } while (answer.charAt(0) != 'n');
    }

    public static long bubbleSort(int[] arr) {
        long cost = 0;
        for (int i = 0; i < arr.length - 1; i++) {
            cost++;
            for (int j = 0; j < arr.length - i - 1; j++) {
                cost++;
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                    cost++;
                }
            }
        }
        return cost;
    }

    public static long insertionSort(int[] arr) {
        int key, j;
        long cost = 0;
        for (int i = 1; i < arr.length; i++, cost++) {
            key = arr[i];
            cost++; // for assuming key

            j = i - 1;
            cost++; // for assigning j

            while (j >= 0 && key < arr[j]) {
                arr[j + 1] = arr[j];
                cost++; // for shifting data in the array

                j--;
                cost++; // for decrementing j

                cost++; //for while loop
            }
            arr[j + 1] = key;
            cost++; // for writing data to the correct position
        }
        return cost;
    }

    public static long selectionSort(int[] arr) {
        int cost = 0;
        int min, loc;
        for (int i = 0; i < arr.length - 1; i++) {
            min = arr[i];
            cost++;
            loc = i;
            for (int j = i + 1; j < arr.length; j++) {
                if (arr[j] < min) {
                    min = arr[j];
                    loc = j;
                    cost++;
                }
            }
            if (loc != i) {
                int temp = arr[i];
                arr[i] = arr[loc];
                arr[loc] = temp;
                cost++;
            }
        }
        return cost;
    }

    public static long mergeSort(int[] arr, int low, int high) {

        //find mid everytime
        int mid = (low + (high - 1)) / 2;

        //note the cost
        long costLeft = 0, costRight = 0, costMerge = 0;

        if (low < high) {
            costLeft = mergeSort(arr, low, mid);
            costRight = mergeSort(arr, mid + 1, high);

            costMerge = merge(arr, low, mid, high);
        }
        return costLeft + costRight + costMerge;
    }

    public static long merge(int[] arr, int low, int mid, int high) {

        //note the cost
        long cost = 0;
        //calculate the size of arrays
        int sizeLeft = mid - low + 1;
        int sizeRight = high - mid;

        //create 2 arrays for both size
        int[] left = new int[sizeLeft];
        int[] right = new int[sizeRight];

        //copying data
        for (int index = 0; index < sizeLeft; index++) {
            left[index] = arr[low + index];
            cost++;
        }
        for (int index = 0; index < sizeRight; index++) {
            right[index] = arr[mid + 1 + index];
            cost++;
        }

        int i = 0, j = 0, k = low;
        while (i < sizeLeft && j < sizeRight) {
            if (left[i] <= right[j]) {
                arr[k] = left[i];
                i++;
            } else {
                arr[k] = right[j];
                j++;
            }
            cost++;
            k++;
            cost++;
        }

        //copy remaining element of left
        while (i < sizeLeft) {
            arr[k] = left[i];
            k++;
            i++;
            cost++;
        }

        //copy remaining element of right
        while (j < sizeRight) {
            arr[k] = right[j];
            k++;
            j++;
            cost++;
        }
        return cost;
    }

    public static long quickSort(int[] arr, int low, int high) {

        long costLeft = 0, costRight = 0, partitionCost = 0;

        int mid;
        if (low < high) {
            long[] costMap = partition(arr, low, high);
            mid = (int) costMap[0];
            partitionCost = costMap[1];

            costLeft = quickSort(arr, low, mid - 1);
            costRight = quickSort(arr, mid + 1, high);
        }
        return costLeft + costRight + partitionCost;
    }

    public static long[] partition(int[] arr, int start, int end) {
            long cost = 0;
            long[] costMap = new long[2];

            int pivot = arr[end];
            int i = (start - 1);

            for (int j = start; j <= end - 1; j++) {
                cost++;
                // If current element is smaller than the pivot
                if (arr[j] < pivot) {
                    i++; // increment index of smaller element
                    int temp = arr[i];
                    arr[i] = arr[j];
                    arr[j] = temp;
                    cost += 3;
                }
            }

            int temp = arr[i + 1];
            arr[i + 1] = arr[end];
            arr[end] = temp;
            cost += 3;

            costMap[0] = (i + 1);
            costMap[1] = cost;

            return costMap;
        }
    }

/*
OUTPUT

Give the file Name of Random Numbers
Eg. [1_Digit_100_entries.txt] :: 50K_entries.txt

Enter the File name you want to store the Result
Ex. [1_Digit_100_sorted_entries.txt] :: 50k_sorted_entries.txt

1. Bubble Sort
2. Insertion Sort
3. Selection Sort
4. Merge Sort
5. Quick Sort
Else. Exit
Enter your Choice for sorting technique :: 4

Sorting Data...
Calculating the Cost...

Cost for sorting :: 2287086
Time Taken :: 32 milliseconds

Do you want to continue ? [y/n] n
 */
```

Readings of Time Analysis ::



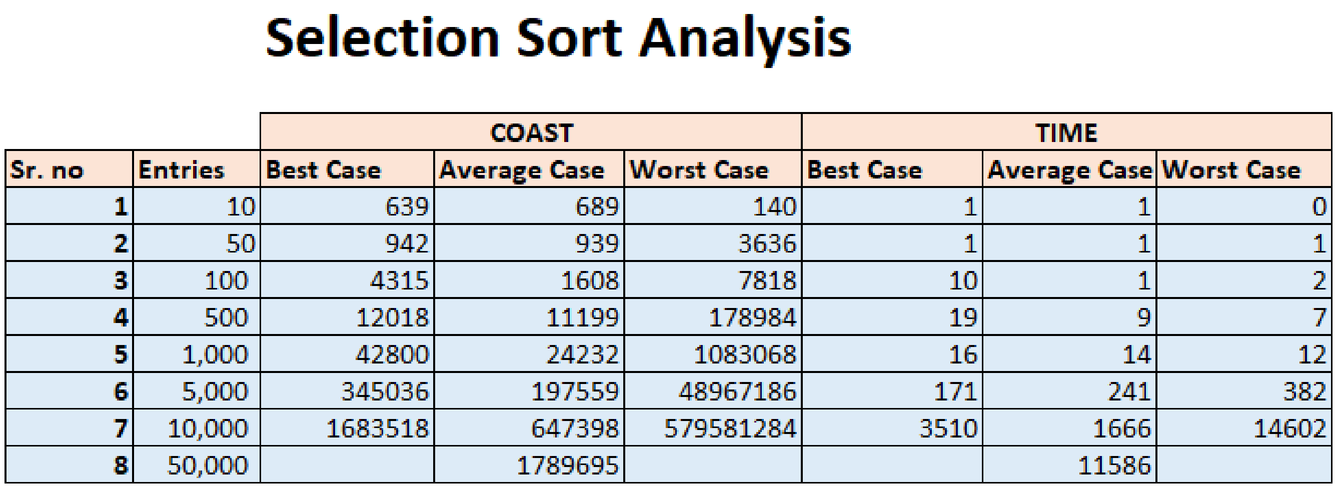
Bubble Sort Analysis

		COAST			TIME		
Sr. no	Entries	Best Case	Average Case	Worst Case	Best Case	Average Case	Worst Case
1	10	54	68	99	0	1	1
2	50	2769	2710	96724	0	1	2
3	100	5049	7507	9999	1	0	1
4	500	1335149	185913	4997499	10	0	11
5	1,000	10852203	750543	97499009	10	12	11
6	5,000	241363945	75199321	2605287274	138	193	1744
7	10,000	6817516355	1202626742	7648176769	6425	3089	6491
8	50,000		1875290861			5013	



Insertion Sort Analysis

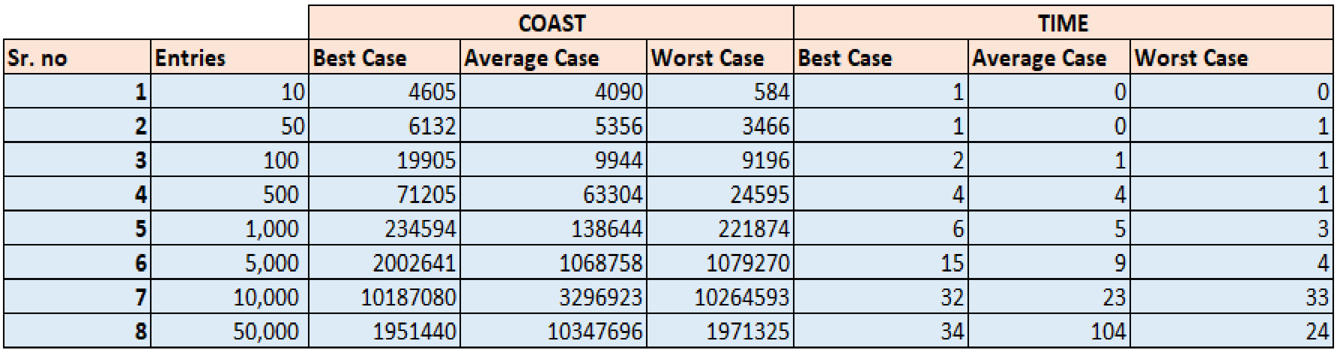
		COAST			TIME		
Sr. no	Entries	Best Case	Average Case	Worst Case	Best Case	Average Case	Worst Case
1	10	2221	3876	481	1	0	1
2	50	9481	11072	74581	1	0	1
3	100	74450	30394	45346	3	1	1
4	500	1996	742230	3750496	0	8	19
5	1,000	15622668	3006760	17636306	18	12	6
6	5,000	472510185	169982708	1994773386	56	48	229
7	10,000	11736921995	1887928744	16913924946	1254	308	1816
8	50,000		7501916358			1045	



Selection Sort Analysis

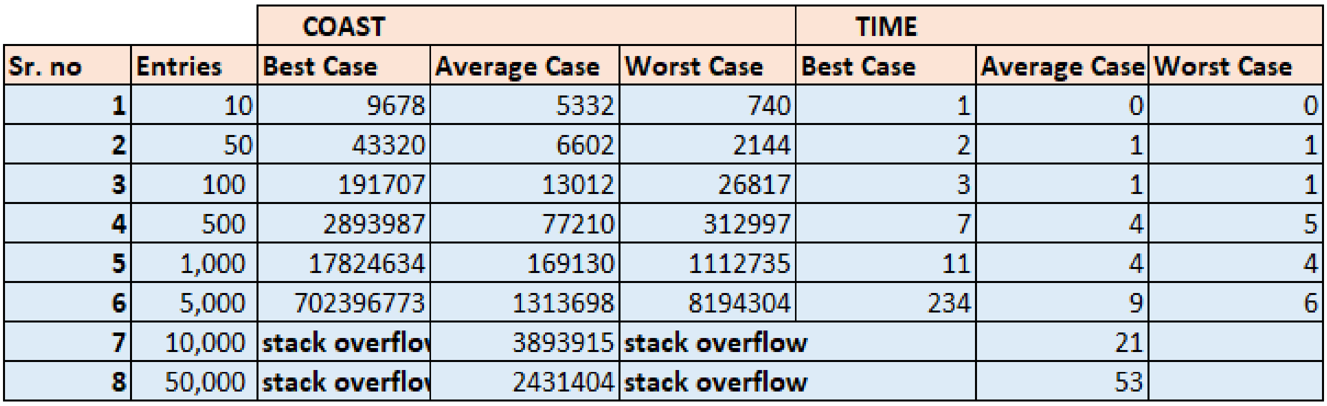
		COAST			TIME		
Sr. no	Entries	Best Case	Average Case	Worst Case	Best Case	Average Case	Worst Case
1	10	639	689	140	1	1	0
2	50	942	939	3636	1	1	1
3	100	4315	1608	7818	10	1	2
4	500	12018	11199	178984	19	9	7
5	1,000	42800	24232	1083068	16	14	12
6	5,000	345036	197559	48967186	171	241	382
7	10,000	1683518	647398	579581284	3510	1666	14602
8	50,000		1789695			11586	

Merge Sort Analysis



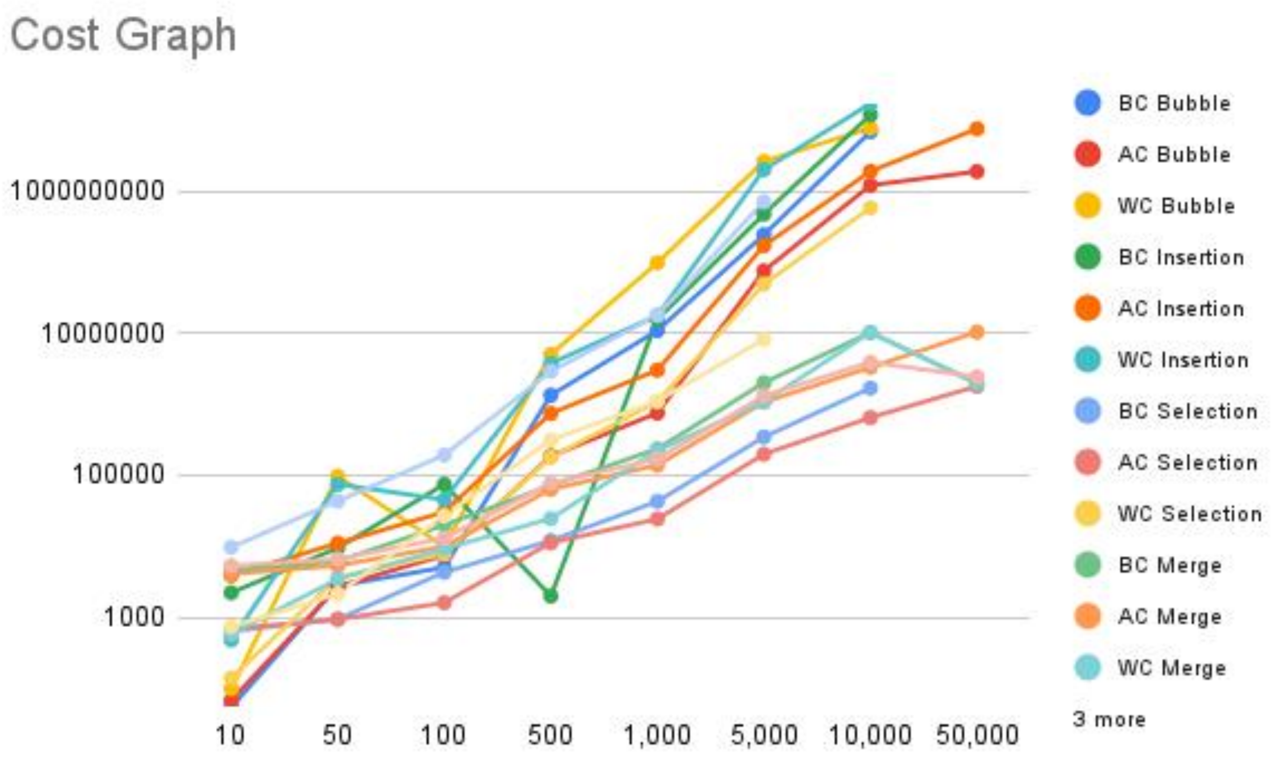
		COAST			TIME		
Sr. no	Entries	Best Case	Average Case	Worst Case	Best Case	Average Case	Worst Case
1	10	4605	4090	584	1	0	0
2	50	6132	5356	3466	1	0	1
3	100	19905	9944	9196	2	1	1
4	500	71205	63304	24595	4	4	1
5	1,000	234594	138644	221874	6	5	3
6	5,000	2002641	1068758	1079270	15	9	4
7	10,000	10187080	3296923	10264593	32	23	33
8	50,000	1951440	10347696	1971325	34	104	24

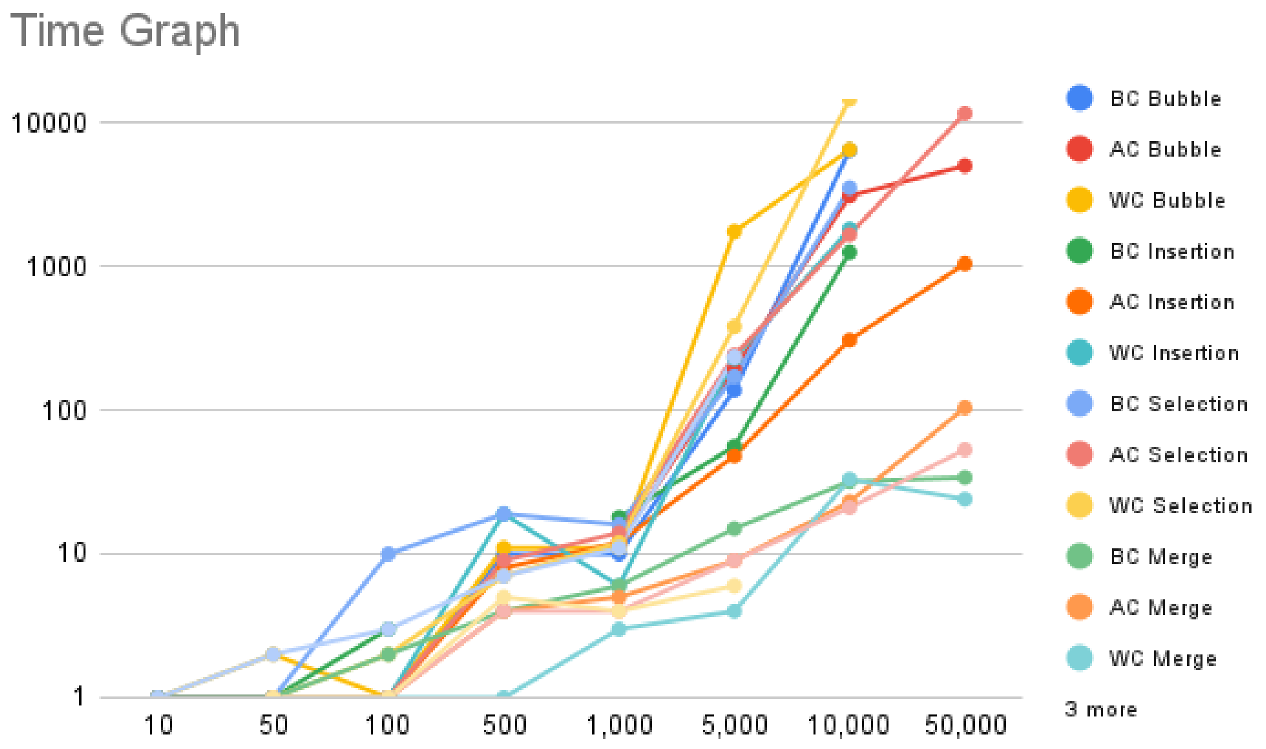
Quick Sort Analysis



		COAST			TIME		
Sr. no	Entries	Best Case	Average Case	Worst Case	Best Case	Average Case	Worst Case
1	10	9678	5332	740	1	0	0
2	50	43320	6602	2144	2	1	1
3	100	191707	13012	26817	3	1	1
4	500	2893987	77210	312997	7	4	5
5	1,000	17824634	169130	1112735	11	4	4
6	5,000	702396773	1313698	8194304	234	9	6
7	10,000	stack overflo	3893915	stack overflow		21	
8	50,000	stack overflo	2431404	stack overflow		53	

Graph Comparison::





Practical 2:

To implement Time Analysis of linear sorting algorithms.

1. Counting Sort
2. Radix Sort
3. Bucket Sort

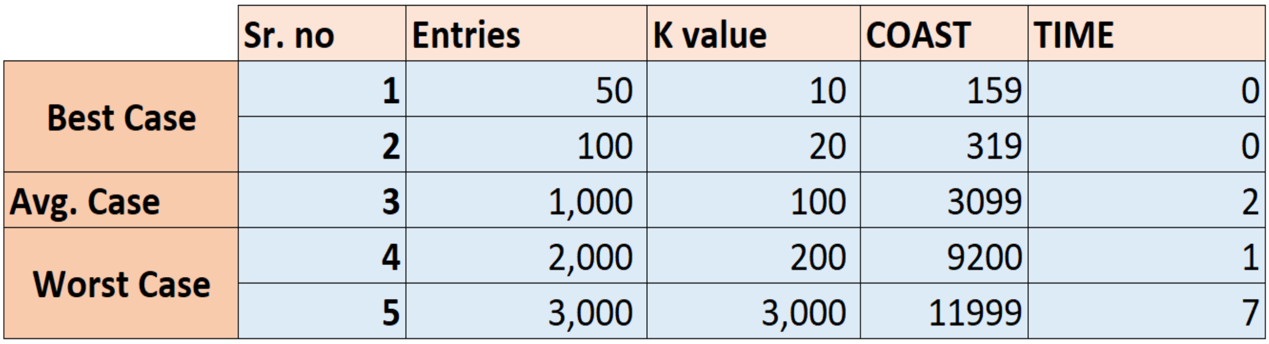
Code Implementation::

```
package Practical_Package;

import java.io.File;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Scanner;

public class P2_TimeAnalysisLinearSortingAlgorithm {

    public static void main(String[] args) {

        ArrayList<Integer> arrayList = new ArrayList<>();
        ArrayList<Double> arrayListDouble = new ArrayList<>();
        Scanner sc = new Scanner(System.in);
        String answer;
        long cost = 0;

        do {
            System.out.print("\nGive the file Name of Random Numbers\nEg.
[1_Digit_100_entries.txt] :: ");
            String fName = sc.next();

            File sourceFile = new File(fName);

            //if file doesn't exists exit
            if (!sourceFile.exists()) {
                System.out.println("\nCan't Locate " + fName);
                System.exit(1);
            }

            System.out.print("\nEnter the File name you want to store the
Result\nEx. [1_Digit_100_sorted_entries.txt] :: ");
            fName = sc.next();

            File destFile = new File(fName);

            //if file exists exit
            if (destFile.exists()) {
                System.out.println("\nDuplicate File names :(");
                System.exit(2);
            }

            try (
                    Scanner input = new Scanner(sourceFile);
                    PrintWriter output = new PrintWriter(destFile)
            ) {
                System.out.println("\n" + sourceFile + " Contains Integers or
Doubles?");

                System.out.println("1. Integer\n2.Double\nElse. Exit");
                System.out.print("Enter your Choice :: ");
                int ch = sc.nextInt();

                int[] arr = {0};
                double[] arrDouble = {0.0};
                switch (ch) {
                    case 1: {
                        //fetch data from source file to arrayList
                        while (input.hasNext())
                            arrayList.add(input.nextInt());

                        arr = new int[arrayList.size()];
                        for (int i = 0; i < arrayList.size(); i++)
                            arr[i] = arrayList.get(i);
                    }
                    break;

                    case 2: {
                        while (input.hasNext())
                            arrayListDouble.add(input.nextDouble());

                        arrDouble = new double[arrayListDouble.size()];
                        for (int i = 0; i < arrayList.size(); i++)
                            arrDouble[i] = arrayListDouble.get(i);
                    }
                    break;

                    default:
                        System.exit(1);
                }
                System.out.println("\n1. Bucket Sort\n2. Counting Sort\n3. Radix
Sort\nElse. Exit");
                System.out.print("Enter your Choice for sorting technique :: ");
                int sortChoice = sc.nextInt();

                long startTime = System.currentTimeMillis();

                switch (sortChoice) {
                    case 1 -> cost = bucketSort(arrDouble, arrDouble.length - 1);
                    case 2 -> cost = countingSort(arr);
                    case 3 -> cost = radixSort(arr);
                    default -> {
                        System.out.println("\nExiting...");
                        System.exit(3);
                    }
                }

                System.out.println("\nSorting Data...");
                System.out.println("Calculating the Cost...");

                for (int j : arr) output.println(j);

                long endTime = System.currentTimeMillis();

                System.out.println("\nCost for sorting :: " + cost);
                System.out.println("Time Taken :: " + (endTime - startTime) + "
milliseconds");

            } catch (Exception e) {
                e.printStackTrace();
            }

            //prompt user if he wants to continue or not.
            System.out.print("\nDo you want to continue ? [y/n] ");
            answer = sc.next().toLowerCase();

        } while (answer.charAt(0) != 'n');
    }

    public static long bucketSort(double[] arr, int n) {
        long cost = 0;
        if (n <= 0)
```

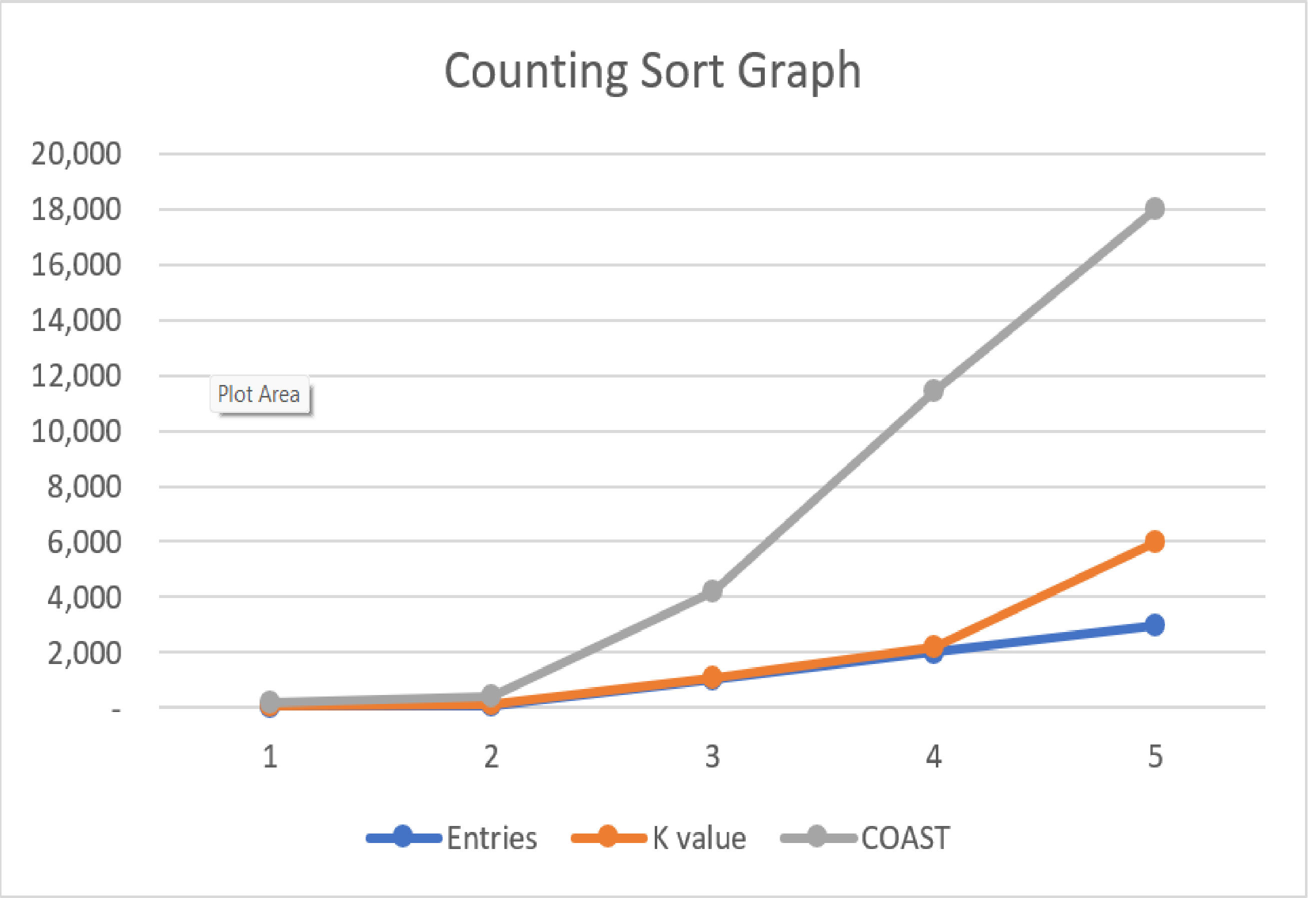
```
            return cost;

        @SuppressWarnings("unchecked")
        ArrayList<Double>[] bucket = new ArrayList[n];

        // Create empty buckets
        for (int i = 0; i < n; i++, cost++)
            bucket[i] = new ArrayList<Double>();

        // Add elements into the buckets
        for (int i = 0; i < n; i++, cost++) {
            int bucketIndex = (int) arr[i] * n;
            bucket[bucketIndex].add(arr[i]);
        }

        // Sort the elements of each bucket
        for (int i = 0; i < n; i++, cost++) {
            Collections.sort((bucket[i]));
        }

        // Get the sorted array
        int index = 0;
        for (int i = 0; i < n; i++, cost++) {
            for (int j = 0, size = bucket[i].size(); j < size; j++) {
                arr[index++] = bucket[i].get(j);
            }
        }
        return cost;
    }

    public static long countingSort(int[] arr) {
        long cost = 0;

        int maxElement = maxInArray(arr);

        int[] count = new int[maxElement + 1];
        int[] brr = new int[arr.length];

        for (int i = 0; i < arr.length; i++, cost++)
            ++count[arr[i]]; // increment the index(element) from 0 to 1 and if the
number repeat then 1 to 2 and so on...

        for (int i = 1; i <= maxElement; i++, cost++)
            count[i] += count[i - 1]; // arranging the count array in order

        for (int i = arr.length - 1; i >= 0; i--, cost++)
            brr[--count[arr[i]]] = arr[i]; // inserting the element from main
array(arr) into brr by decrementing by one first

        for (int i = 0; i < arr.length; i++, cost++)
            arr[i] = brr[i]; // copying array brr to array arr

        return cost;
    }

    public static long radixSort(int[] arr) {
        long cost = 0;
        int max = maxInArray(arr); // finding the max element from the main array

        // pos variable shows the position of the digit in the number
        // (pos = 1 means unit , pos = 10 means tens , pos = 100 = hundred)
        for (int pos = 1; max / pos > 0; pos *= 10, cost++)
            cost += countSortRadix(arr, pos); // calling count function for pos = 1
, pos = 10 , pos = 100

        return cost;
    }
    public static long countSortRadix(int[] arr, int pos) {
        long cost = 0;
        int[] count = new int[10];
        int[] brr = new int[arr.length];

        for (int i = 0; i < arr.length; i++, cost++)
            ++count[(arr[i] / pos) % 10];

        for (int i = 1; i < 10; i++, cost++)
            count[i] += count[i - 1];

        for (int i = arr.length - 1; i >= 0; i--, cost++)
            brr[--count[(arr[i] / pos) % 10]] = arr[i];

        for (int i = 0; i < arr.length; i++, cost++) // copying all element from brr to original array
            arr[i] = brr[i];

        return cost;
    }

    public static int maxInArray(int[] arr) {
        int max_value = 0;
        for (int j : arr)
            if (j > max_value)
                max_value = j;
        return max_value;
    }
}

/*
Give the file Name of Random Numbers
Eg. [1_Digit_100_entries.txt] :: sr5.txt

Enter the File name you want to store the Result
Ex. [1_Digit_100_sorted_entries.txt] :: bucket_sr5_sorted.txt
sr5.txt Contains Integers or Doubles?
1. Integer
2.Double
Else. Exit
Enter your Choice :: 2

1. Bucket Sort
2. Counting Sort
3. Radix Sort
Else. Exit
Enter your Choice for sorting technique :: 1

Sorting Data...
Calculating the Cost...

Cost for sorting :: 1996
Time Taken :: 0 milliseconds

Do you want to continue ? [y/n] n
 */
```

Readings of Time Analysis ::

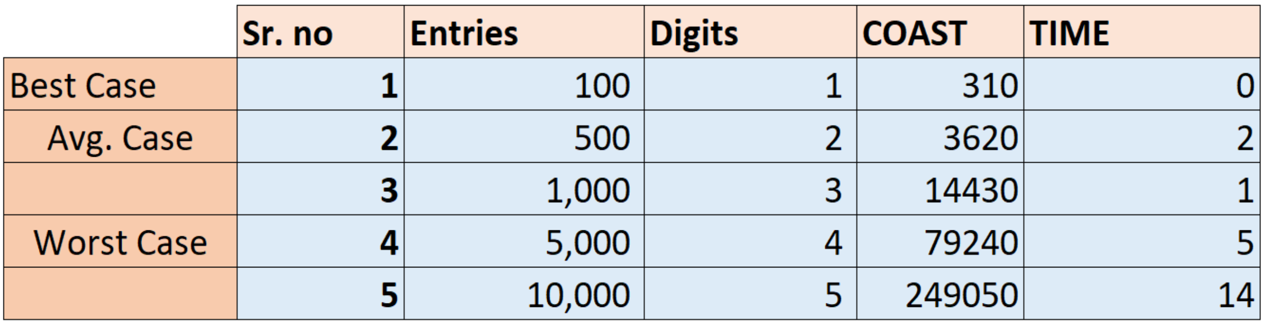
Counting Sort Analysis



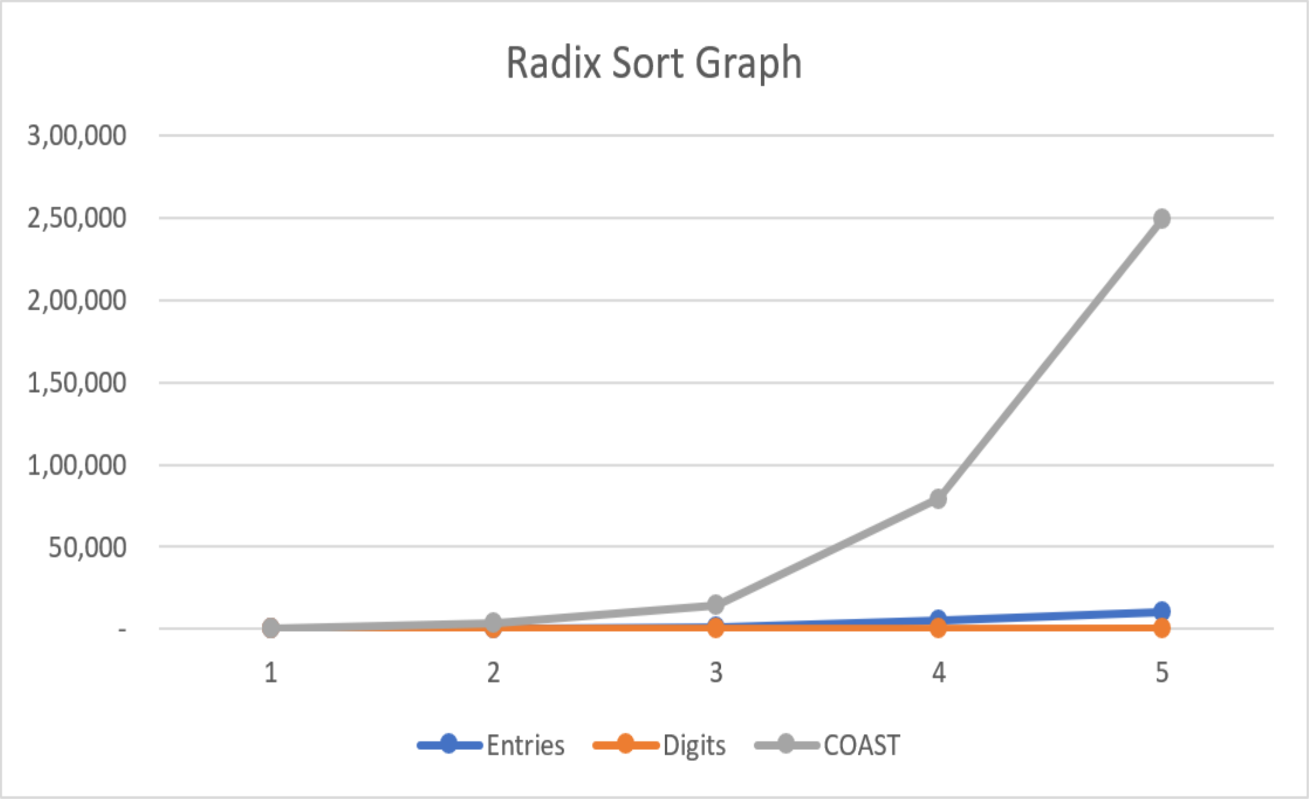
	Sr. no	Entries	K value	COAST	TIME
Best Case	1	50	10	159	0
	2	100	20	319	0
Avg. Case	3	1,000	100	3099	2
Worst Case	4	2,000	200	9200	1
	5	3,000	3,000	11999	7



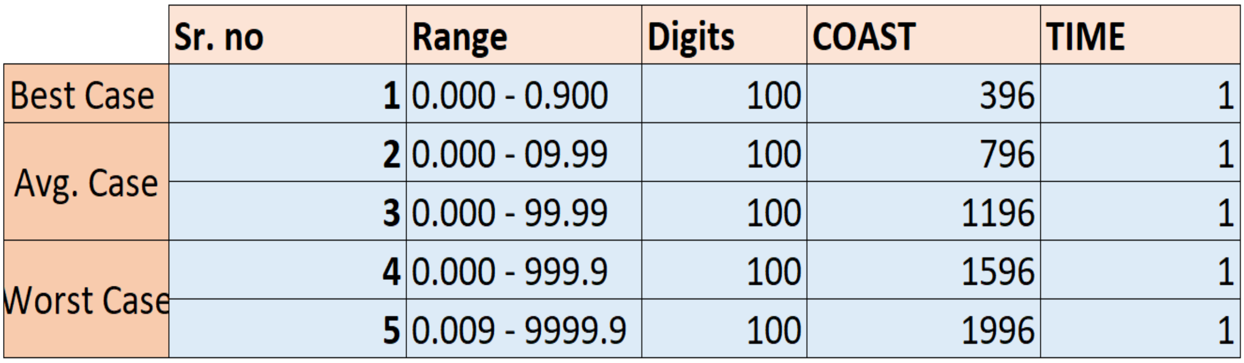
Radix Sort Analysis



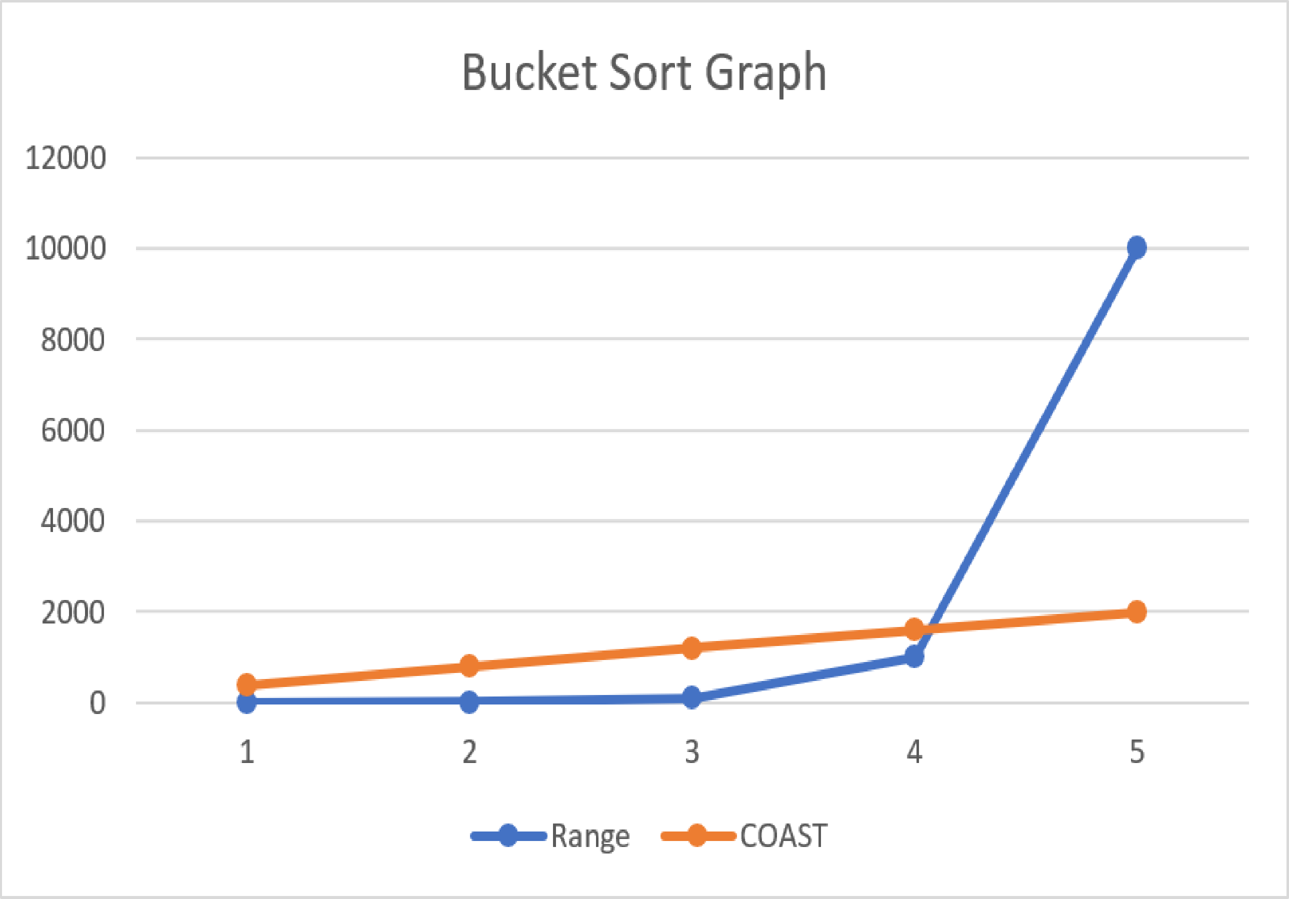
	Sr. no	Entries	Digits	COAST	TIME
Best Case	**1**	100	1	310	0
Avg. Case	**2**	500	2	3620	2
	3	1,000	3	14430	1
Worst Case	**4**	5,000	4	79240	5
	5	10,000	5	249050	14



Bucket Sort Analysis



	Sr. no	Range	Digits	COAST	TIME
Best Case	1	0.000 - 0.900	100	396	1
Avg. Case	2	0.000 - 09.99	100	796	1
	3	0.000 - 99.99	100	1196	1
Worst Case	4	0.000 - 999.9	100	1596	1
	5	0.009 - 9999.9	100	1996	1



Practical 3:

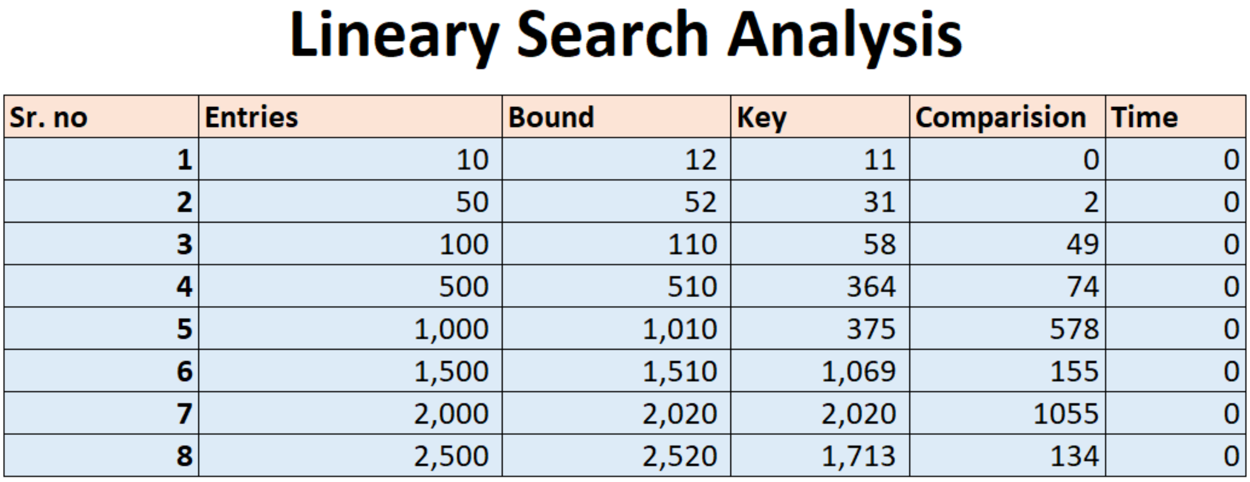
To implement Time Analysis of Binary Search Algorithm compared to Linear Search Algorithm.

Code Implementation::

```
package Practical_Package;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;
import java.util.Scanner;

public class P3_TimeAnalysisLinearAndBinarySearch {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        Random rd = new Random();
        String answer;
        int cost = 0;

        do {
            System.out.print("Enter the size of array you want to generate Randomly 
:: ");
            int arraySize = sc.nextInt();
            System.out.print("Give upper bound for " + arraySize + " entries :: ");
            int bound = sc.nextInt();

            System.out.println("\nGenerating " + arraySize + " Random entries with 
upper bound " + bound + "...");
            ArrayList<Integer> arr = new ArrayList<>(arraySize);
            for (int i = 0; i < arraySize; i++) {
                arr.add(rd.nextInt(bound));
            }

            int key = rd.nextInt(bound) + 1; //preventing zero to be generated
            System.out.println("Randomly generated Key :: " + key);

            int ans = 0;
            int[] map;

            long startTime = System.currentTimeMillis();
            map = linearSearch(arr, key);
            ans = map[0];
            cost = map[1];
            long endTime = System.currentTimeMillis();

            if (ans >= 0) {
                System.out.println("\nKey Found in Array !!");
                System.out.println("Linear Search Cost :: " + cost);
                System.out.println("Time Taken :: " + (endTime - startTime));
            } else
                System.out.println("\nKey not found in array :(");

            startTime = System.currentTimeMillis();
            map = binarySearch(arr, key);
            ans = map[0];
            cost = map[1];
            endTime = System.currentTimeMillis();

            //print here cost and time
            if (ans >= 0) {
                System.out.println("\n\nKey Found in Array !!");
                System.out.println("Binary Search Cost :: " + cost);
                System.out.println("Time Taken :: " + (endTime - startTime));
            } else
                System.out.println("\nKey not found in array :(");

            //prompt user if he wants to continue or not.
            System.out.print("\nDo you want to continue ? [y/n] ");
            answer = sc.next().toLowerCase();

        } while (answer.charAt(0) != 'n');
    }

    public static int[] linearSearch(ArrayList<Integer> arr, int key) {

        int i;
        int[] map = new int[2];
        int cost = 0;
        boolean found = false;

        for (i = 0; i < arr.size(); i++, cost++)
            if (arr.get(i) == key) {
                found = true;
                break;
            }

        if (found)
            map[0] = i;
        else
            map[0] = -1;
        map[1] = cost;

        return map;
    }

    public static int[] binarySearch(ArrayList<Integer> arr, int key) {
        //first sort the array to run binary search algo
        Collections.sort(arr);

        int low = 0, finalMid = 0, cost = 0;
        boolean found = false;
        int high = arr.size() - 1;
        int[] map = new int[2];

        while (high >= low) {

            int mid = low + (high - low) / 2;
            cost++;

            if (arr.get(mid) == key) {
                cost++;
                finalMid = mid;
                found = true;
                break;
            } else if (arr.get(mid) > key)
                high = mid - 1;
            else
                low = mid + 1;
            cost++;
        }
        if (found)
            map[0] = finalMid;
        else
            map[0] = -1;
        map[1] = cost;
        return map;
    }
}
```

```
/*
OUTPUT

Enter the size of array you want to generate Randomly :: 10
Give upper bound for 10 entries :: 12

Generating 10 Random entries with upper bound 12...
Randomly generated Key :: 11

Key Found in Array !!
Linear Search Cost :: 0
Time Taken :: 0


Key Found in Array !!
Binary Search Cost :: 8
Time Taken :: 0
 */
```

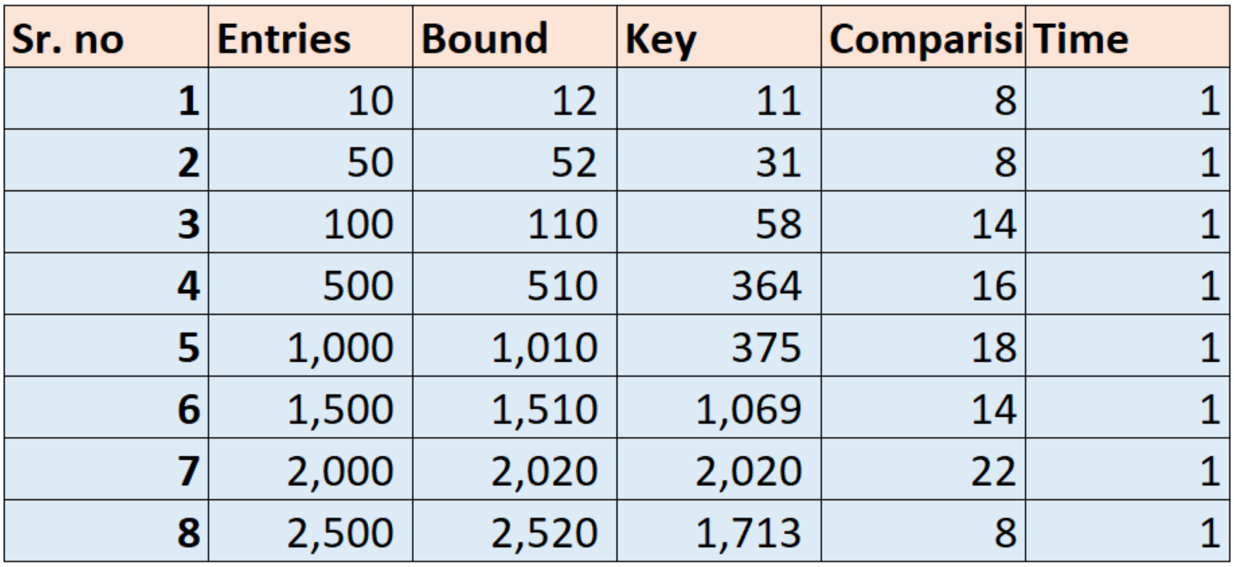
Readings of Time Analysis ::



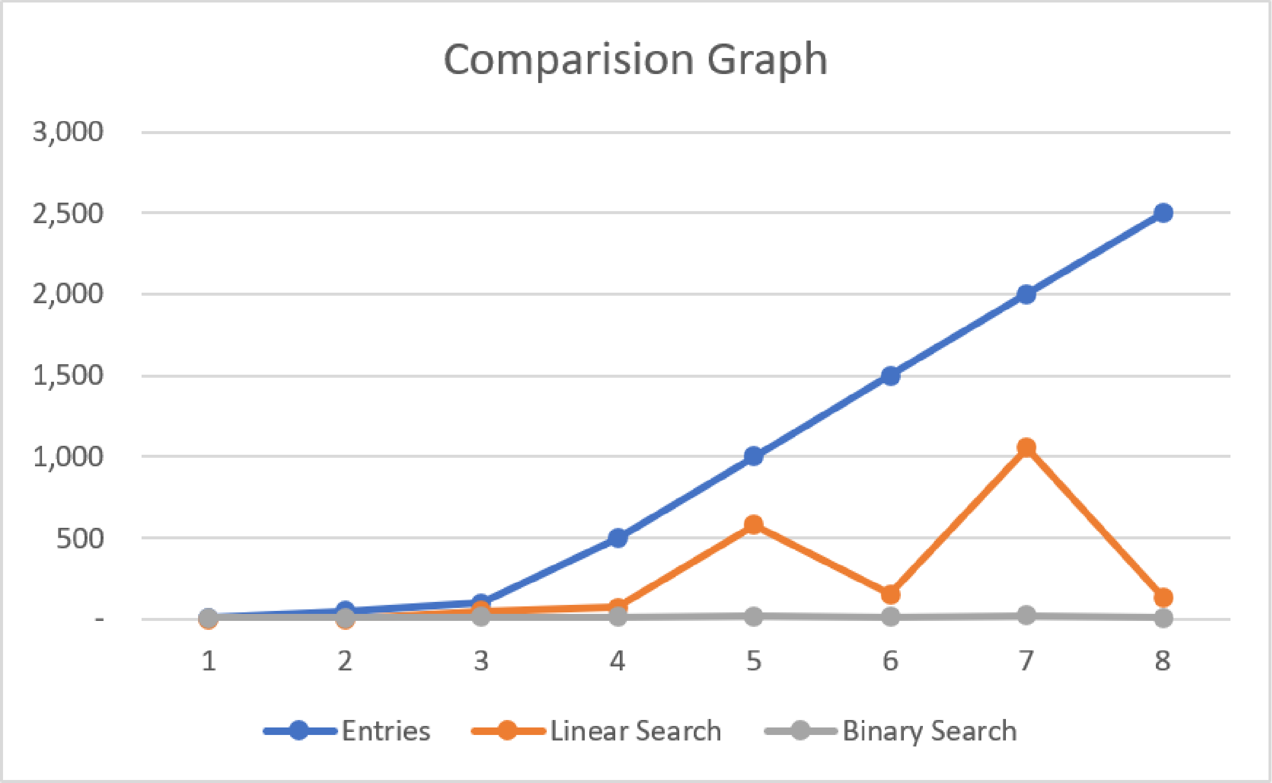
Lineary Search Analysis

Sr. no	Entries	Bound	Key	Comparision	Time
1	10	12	11	0	0
2	50	52	31	2	0
3	100	110	58	49	0
4	500	510	364	74	0
5	1,000	1,010	375	578	0
6	1,500	1,510	1,069	155	0
7	2,000	2,020	2,020	1055	0
8	2,500	2,520	1,713	134	0

Binary Search Analysis



Sr. no	Entries	Bound	Key	Comparisi	Time
1	10	12	11	8	1
2	50	52	31	8	1
3	100	110	58	14	1
4	500	510	364	16	1
5	1,000	1,010	375	18	1
6	1,500	1,510	1,069	14	1
7	2,000	2,020	2,020	22	1
8	2,500	2,520	1,713	8	1



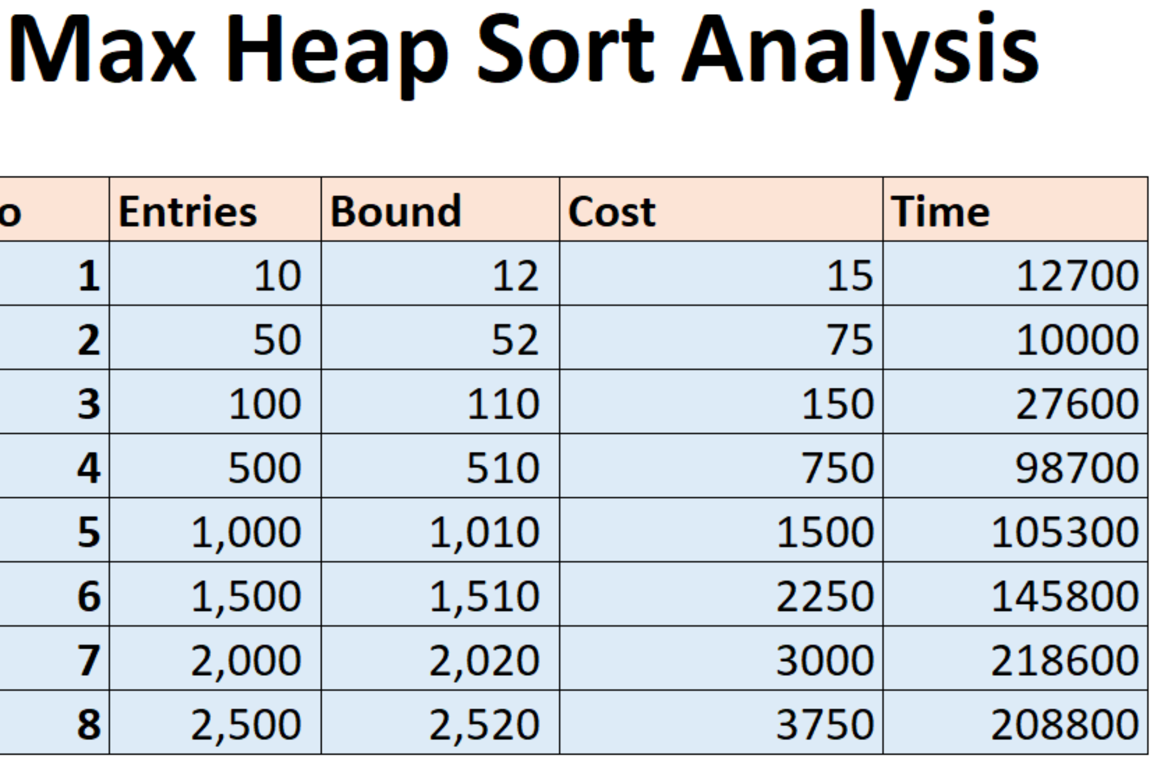
Practical 4:

To implement Time Analysis of Max Heap Sort Algorithm.

Code Implementation::

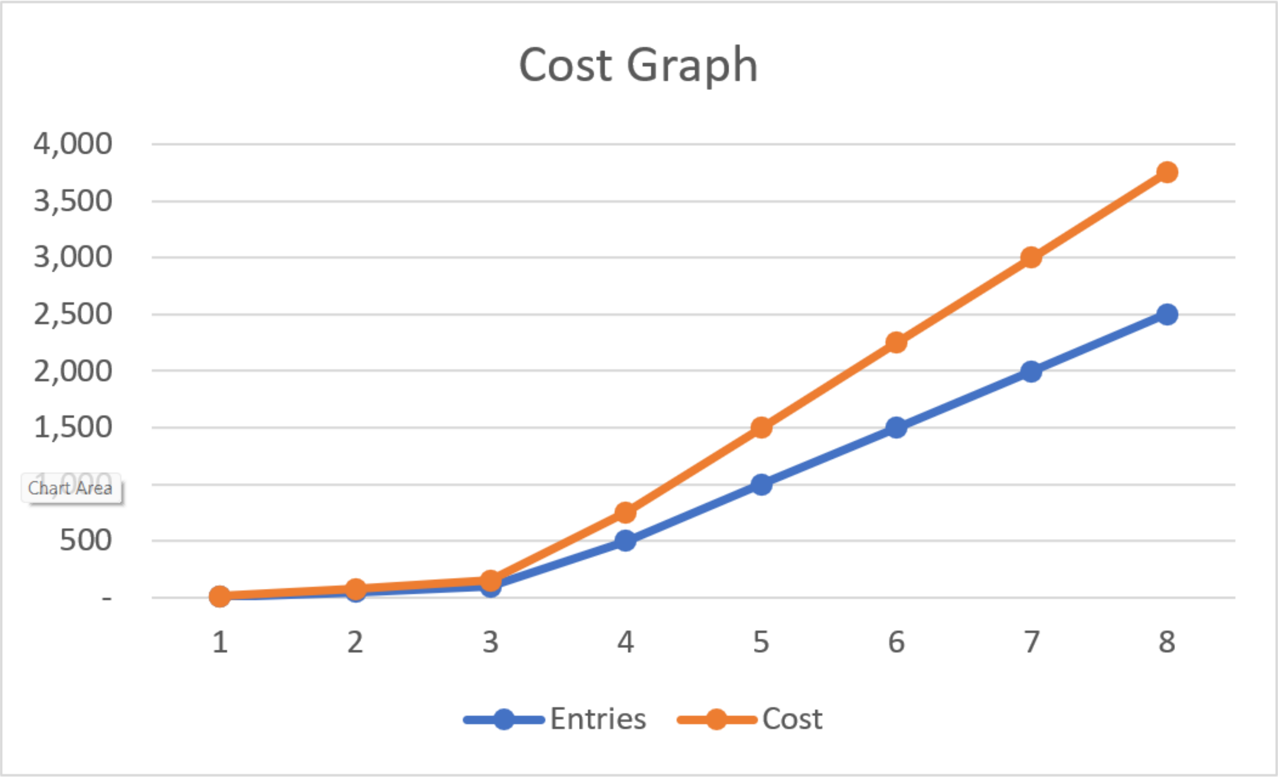
```
package Practical_Package;

import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;

public class P4_TimeAnalysisMaxHeapSort {

    public static void main(String[] args) {

        Random rd = new Random();
        Scanner sc = new Scanner(System.in);
        String answer;
        long cost;

        do {
            System.out.print("Enter the size of array you want to generate Randomly
:: ");
            int arraySize = sc.nextInt();
            System.out.print("Give upper bound for " + arraySize + " entries :: ");
            int bound = sc.nextInt();

            System.out.println("\nGenerating " + arraySize + " Random entries with
upper bound " + bound + "...");
            ArrayList<Integer> arrayList = new ArrayList<>(arraySize);
            for (int i = 0; i < arraySize; i++)
                arrayList.add(rd.nextInt(bound));

            int[] arr = new int[arrayList.size()];
            for (int i : arr)
                arr[i] = arrayList.get(i);

            System.out.println("\nSorting Data...");
            System.out.println("Calculating the Cost...");

            long startTime = System.nanoTime();
            cost = maxHeapSort(arr);
            long endTime = System.nanoTime();

            System.out.println("\nCost for sorting :: " + cost);
            System.out.println("Time Taken :: " + (endTime - startTime) + "
nanoseconds");

            //prompt user if he wants to continue or not.
            System.out.print("\nDo you want to continue ? [y/n] ");
            answer = sc.next().toLowerCase();

        } while (answer.charAt(0) != 'n');
    }

    public static long maxHeapSort(int[] arr) {
        long cost = 0;
        int size = arr.length;

        //building heap
        for (int i = size / 2 - 1; i >= 0; i--, cost++)
            heapify(arr, size, i, cost);

        for (int i = size - 1; i >= 0; i--) {
            // Move current root to end
            int temp = arr[0];
            arr[0] = arr[i];
            arr[i] = temp;

            heapify(arr, i, 0, cost);
            cost++;
        }
        return cost;
    }

    public static long heapify(int[] arr, int n, int i, long cost) {
        int largest = i;  // Initialize largest as root
        int left = 2 * i + 1;  // left = 2*i + 1
        int right = 2 * i + 2;  // right = 2*i + 2

        // If left child is larger than root
        if (left < n && arr[left] > arr[largest]) {
            largest = left;
        }

        // If right child is larger than largest so far
        if (right < n && arr[right] > arr[largest]) {
            largest = right;
        }

        // If largest is not root
        if (largest != i) {
            int temp = arr[i];
            arr[i] = arr[largest];
            arr[largest] = temp;

            // Recursively heapify the affected sub-tree
            heapify(arr, n, largest, cost);
        }
        return cost;
    }
}

/*
OUTPUT

Enter the size of array you want to generate Randomly :: 2500
Give upper bound for 2500 entries :: 2510

Generating 2500 Random entries with upper bound 2510...

Sorting Data...
Calculating the Cost...

Cost for sorting :: 3750
Time Taken :: 208800 nanoseconds

Do you want to continue ? [y/n] N
 */
```

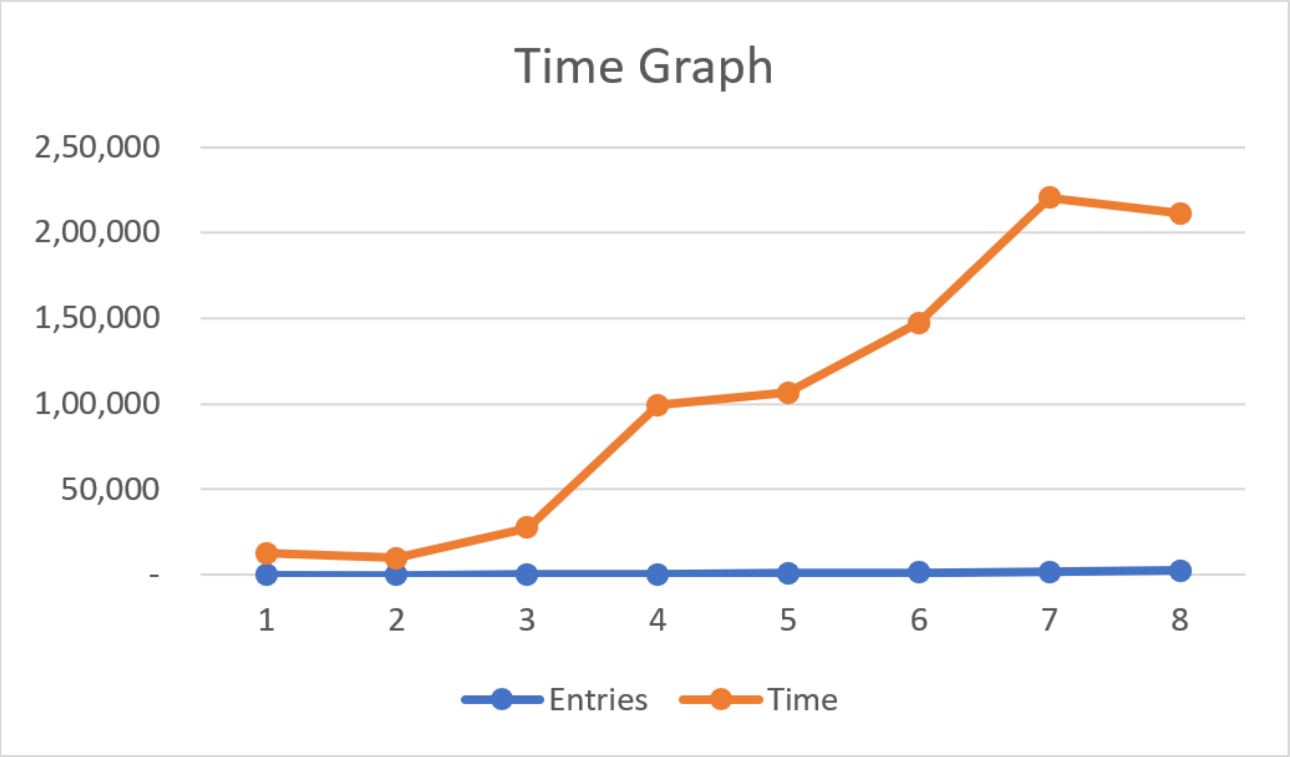
Readings of Time Analysis ::



Max Heap Sort Analysis

Sr. no	Entries	Bound	Cost	Time
1	10	12	15	12700
2	50	52	75	10000
3	100	110	150	27600
4	500	510	750	98700
5	1,000	1,010	1500	105300
6	1,500	1,510	2250	145800
7	2,000	2,020	3000	218600
8	2,500	2,520	3750	208800




Time Graph
2,50,000
2,00,000
1,50,000
1,00,000
50,000
-
1
2
3
4
5
6
7
8
Entries
Time

Practical 5:

To implement Time Analysis of factorial programs using iterative and recursive methods.

Code implementation::

```
package Practical_Package;

import java.util.Random;
import java.util.Scanner;

class Cost {
    public int bound = 20;
    public int times = 1;

    public int[] generateArray() {
        Scanner sc = new Scanner(System.in);
        Random rd = new Random();

        System.out.print("How many Times you want to do analysis :: ");
        this.times = sc.nextInt();

        //preparing array to keep random number same for recursive and iterative
        int[] arr = new int[times];
        for (int i = 0; i < arr.length; i++)
            arr[i] = rd.nextInt(bound) + 1; //preventing 0 to be calculated for factorial

        return arr;
    }
}

class RecursiveCost extends Cost {
    public long rCost = 0;

    public long factRecursive(int num) {
        this.rCost++;
        return (num == 1) ? 1 : num * factRecursive(num - 1);
    }
}

class IterativeCost extends Cost {
    public long iCost = 0;

    public long factIterative(int num) {
        long fact = 1;
        for (int i = 2; i <= num; i++, this.iCost++)
            fact = fact * i;

        return fact;
    }
}

public class P5_TimeAnalysisRecursiveAndIterative {

    public static void main(String[] args) {
        RecursiveCost recursiveCost = new RecursiveCost();
        IterativeCost iterativeCost = new IterativeCost();

        int[] arr = recursiveCost.generateArray();
        String answer;

        do {
            System.out.println("\n1. Iterative Way\n2.Recursive Way\nElse. Exit");
            System.out.print("Enter your choice :: ");
            Scanner sc = new Scanner(System.in);
            int choice = sc.nextInt();
            switch (choice) {
                case 1:
                    //loop for iterative
                    for (int i = 0; i < arr.length; i++) {
                        iterativeCost.iCost = 0; //reset cost

                        long startTime = System.nanoTime();
                        long factAns = iterativeCost.factIterative(arr[i]);
                        long endTime = System.nanoTime();

                        System.out.println("\n" + arr[i] + "! = " + factAns);
                        System.out.println("Iterative Cost for " + arr[i] + "! :: " +
iterativeCost.iCost);
                        System.out.println("Time Taken (nanoTime):: " + (endTime -
startTime));
                    }
                    break;

                case 2:
                    //loop for recursive
                    for (int i = 0; i < recursiveCost.times; i++) {
                        recursiveCost.rCost = 0; // reset cost

                        long startTime = System.nanoTime();
                        long factAns = recursiveCost.factRecursive(arr[i]);
                        long endTime = System.nanoTime();

                        System.out.println("\n" + arr[i] + "! = " + factAns);
                        System.out.println("Recursive Cost " + recursiveCost.rCost);
                        System.out.println("Time Taken (nanoTime):: " + (endTime -
startTime));
                    }
                    break;

                default:
                    System.out.println("Exiting...");
                    System.exit(1);
            }

            //prompt user if he wants to continue or not.
            System.out.print("\nDo you want to do analysis for either way ? [y/n] ");
            answer = sc.next().toLowerCase();

        } while (answer.charAt(0) != 'n');
    }
}

/*
OUTPUT

How many Times you want to do analysis :: 2

1. Iterative Way
2.Recursive Way
Else. Exit
Enter your choice :: 1

17! = 355687428096000
Iterative Cost for 17! :: 16
Time Taken (nanoTime):: 9200

20! = 2432902008176640000
Iterative Cost for 20! :: 19
Time Taken (nanoTime):: 2200

Do you want to do analysis for either way ? [y/n] n
 */
```

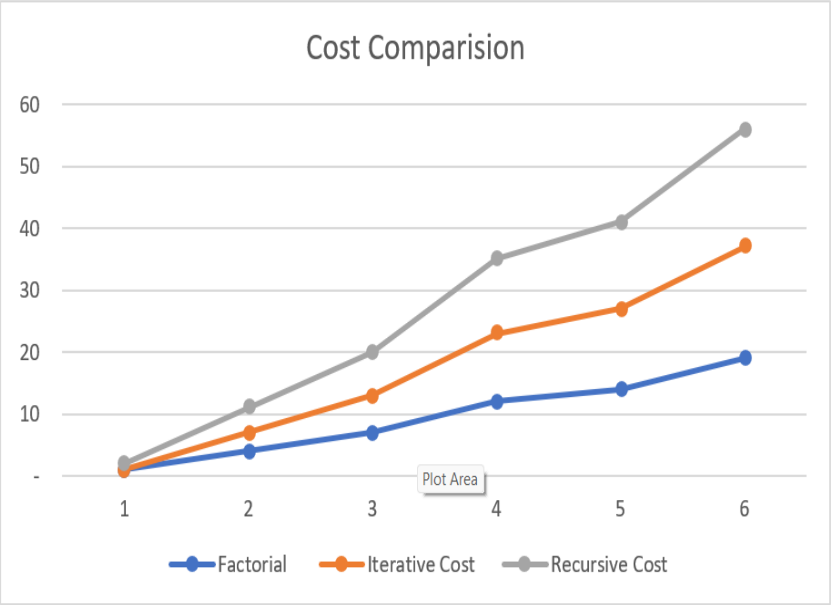
Readings of Time Analysis ::

Iteration Vs Recursion

			Iterative		TIME	
Sr. no	Factorial	Answer	Cost	Time	Cost	Time
1	1	1	0	700	1	600
2	4	24	3	1400	4	900
3	7	5,040	6	1700	7	5040
4	12	47,90,01,600	11	1500	12	1400
5	14	87,17,82,91,200	13	7400	14	1800
6	19	1,21,64,51,00,40,88,32,000	18	1700	19	1900

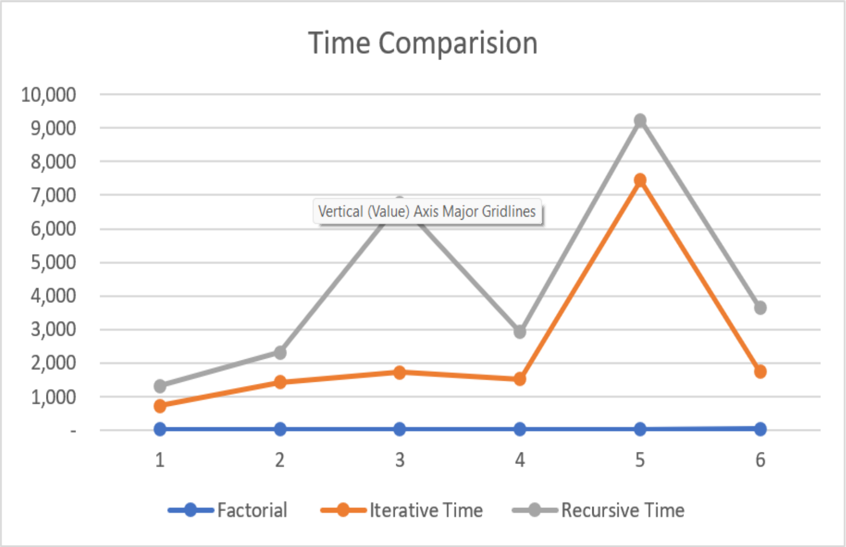
Cost Graph

Factorial	Iterative Cost	Recursive Cost
1	0	1
4	3	4
7	6	7
12	11	12
14	13	14
19	18	19



Time Graph

Factorial	Iterative Time	Recursive Time
22	700	600
26	1400	900
29	1700	5040
33	1500	1400
36	7400	1800
40	1700	1900



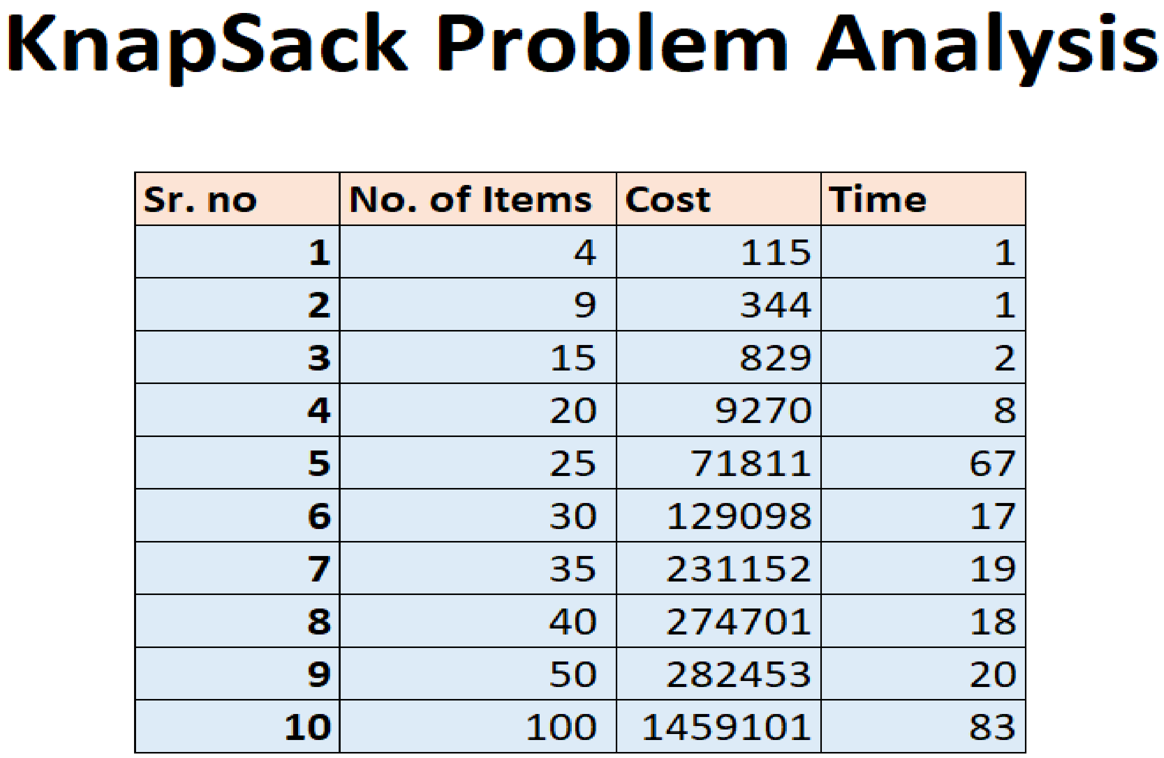
Practical 6:

To implement Knapsack problems using dynamic programming.

Code Implementation::

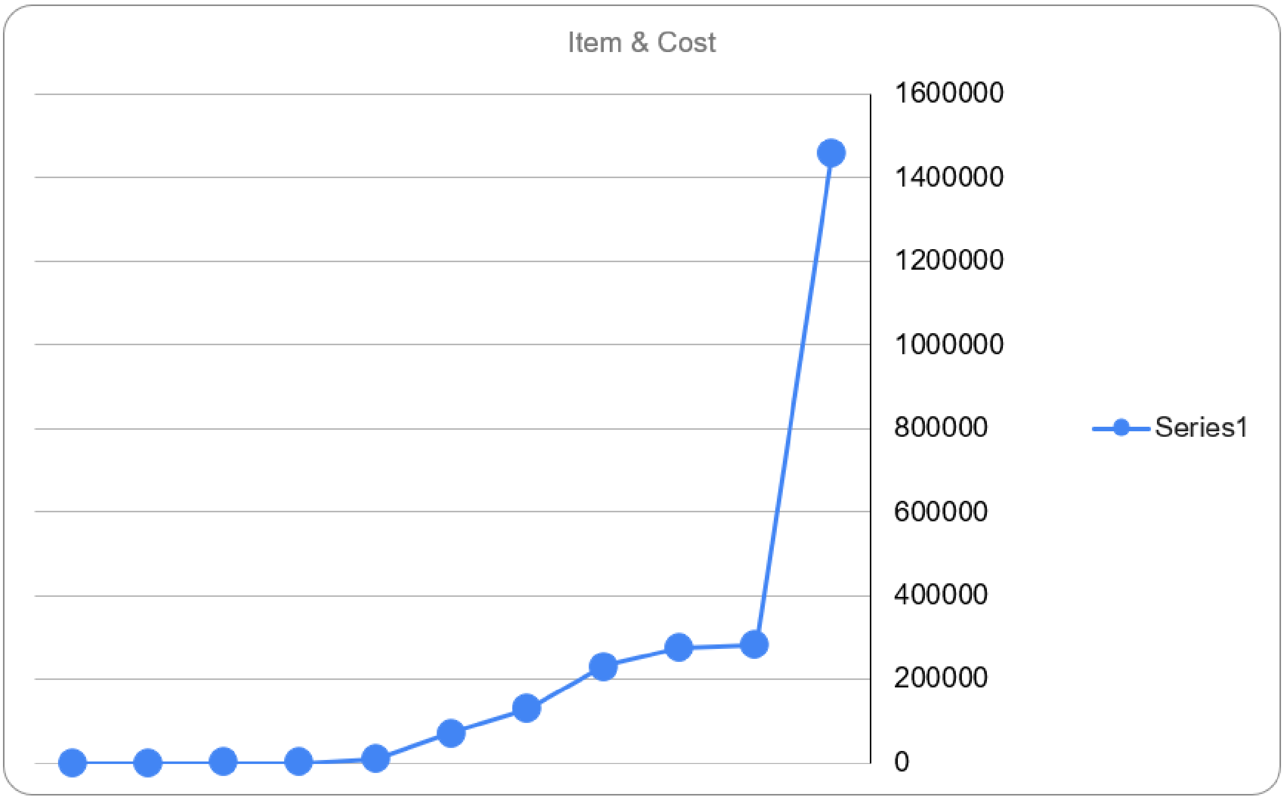
```
package Practical_Package;

import java.util.Locale;
import java.util.Random;
import java.util.Scanner;

public class P6_TimeAnalysisKnapSackProblem {

    public static long cost = 0;

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        Random rd = new Random();
        String answer;

        do {
            System.out.print("How many objects, with you want to do analyze? ::  ");
            int times = sc.nextInt();

            //generate weight array, price array & maxWeight of knapSack
            int[] weightArray = generateArray(times);
            System.out.println("\nWeight array generated.");
            int[] priceArray = generateArray(times);
            System.out.println("Price array generated.");
            int maxWeight = rd.nextInt(1000);
            System.out.println("Max Weight Fixed.");

            long startTime = System.currentTimeMillis();
            int ans = knapSack(priceArray, weightArray, maxWeight,
weightArray.length);
            long endTime = System.currentTimeMillis();

            System.out.println("\nans :: " + ans);
            System.out.println("Cost :: " + cost);
            System.out.println("TIME TAKEN :: " + (endTime - startTime) + "
Millisecond");

            System.out.print("\nDo you want to continue ? [y/n] :: ");
            answer = sc.next().toLowerCase(Locale.ROOT);
        } while (answer.charAt(0) != 'n');
    }

    public static int knapSack(int[] p, int[] wt, int W, int n) {
        cost++;

        // table to store the calculated subProblem's result
        int[][] t = new int[n + 1][W + 1];

        //initialize the table with -1
        for (int i = 0; i < n + 1; i++) {
            for (int j = 0; j < W + 1; j++)
                t[i][j] = -1;
            cost += 2;
        }

        //base condition
        if (W == 0 || n == 0) {
            cost++;
            return 0;
        }

        //check if this particular subProblem is already solved ?
        if (t[n][W] != -1) {
            cost++;
            return t[n][W];
        }

        //Able to choose
        if (wt[n - 1] <= W) //                                  choosing
not choosing
            return t[n][W] = Math.max(p[n - 1] + knapSack(p, wt, W - wt[n - 1], n -
1), knapSack(p, wt, W, n - 1));
        else  // not able to choose as W < wt[n - 1]
            return t[n][W] = knapSack(p, wt, W, n - 1);
    }

    public static int[] generateArray(int times) {
        Random rd = new Random();

        //preparing array to keep random number same for recursive and iterative
        int[] arr = new int[times];
        for (int i = 0; i < arr.length; i++)
            arr[i] = rd.nextInt(1000) + 1; //preventing 0 to be calculated for
factorial

        return arr;
    }
}

/*
OUTPUT
How many objects, with you want to do analyze? ::  100

Weight array generated.
Price array generated.
Max Weight Fixed.

ans :: 2717
Cost :: 139050
TIME TAKEN :: 32 Millisecond
 */
```

Readings of Time Analysis ::



KnapSack Problem Analysis

Sr. no	No. of Items	Cost	Time
1	4	115	1
2	9	344	1
3	15	829	2
4	20	9270	8
5	25	71811	67
6	30	129098	17
7	35	231152	19
8	40	274701	18
9	50	282453	20
10	100	1459101	83



Practical 7:

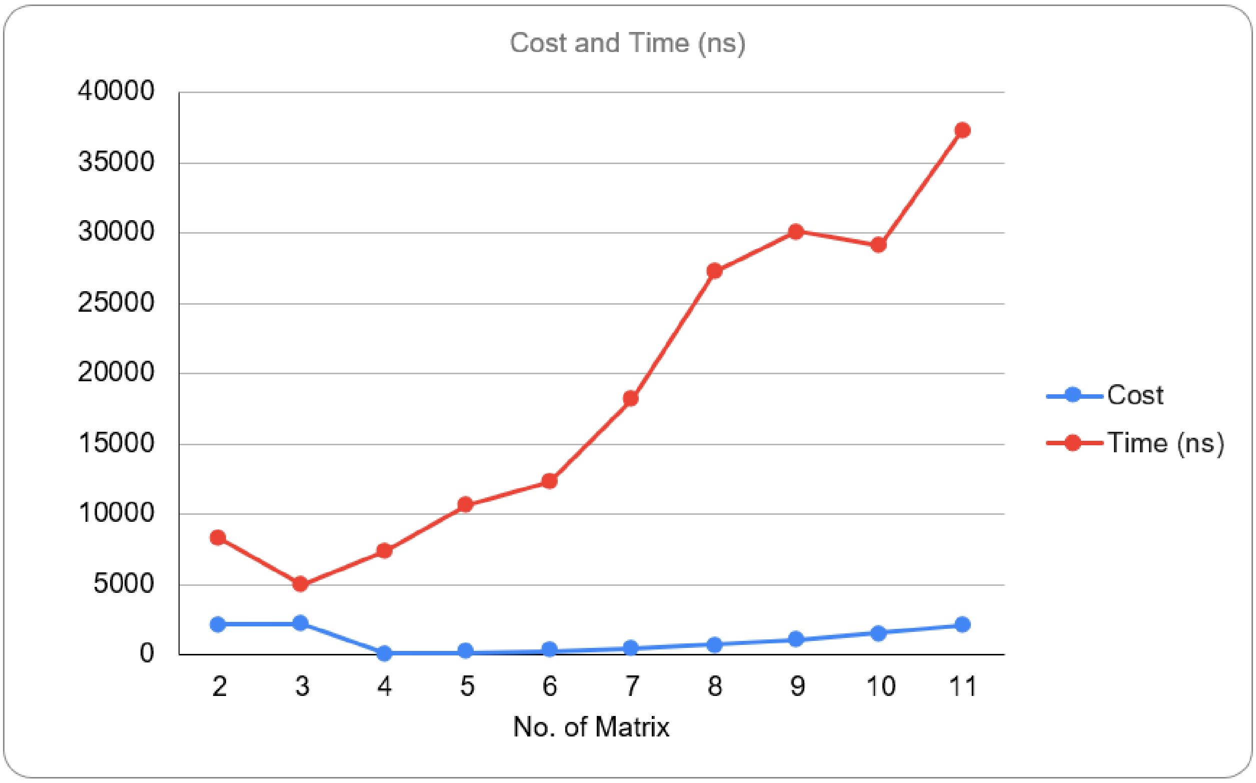
To implement chain matrix multiplication using dynamicprogramming.

Code Implementation::

```
package Practical_Package;

import java.util.Random;
import java.util.Scanner;

public class P7_TimeAnalysisMatrixChainMultiplication {
    public static long cost = 0;

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random rd = new Random();
        String answer;

        do {
            System.out.print("Enter the number of matrix you want to multiply :: ");
            int numOfMatrix = sc.nextInt();
            numOfMatrix += 1; //1 extra dimensions

            System.out.println("\nGenerating dimensions for " + (numOfMatrix - 1) +
" Matrices...");
            int[] arr = new int[numOfMatrix];
            int size = arr.length;

            for (int i = 0; i < size; i++)
                arr[i] = ((rd.nextInt(5) + 1) * 10); // 10 - 50

            System.out.println("\nThis are the Matrices Dimension");
            for (int ele : arr)
                System.out.print(ele + " ");

            long startTime = System.nanoTime();
            int ans = MatrixChainOrder(arr, size);
            long endTime = System.nanoTime();

            System.out.println("\nCost :: " + cost);
            System.out.println("Minimum number of multiplications :: " + ans);
            System.out.println("Time Taken :: " + (endTime - startTime) + "
nanoseconds");

            //prompt user if he wants to continue or not.
            System.out.print("\nDo you want to continue ? [y/n] ");
            answer = sc.next().toLowerCase();

        } while (answer.charAt(0) != 'n');
    }

    static int MatrixChainOrder(int[] p, int n) {
        int[][] table = new int[n][n];

        int i, j, k, L, q;

        for (i = 1; i < n; i++) {
            table[i][i] = 0;
            cost++;
        }

        // L is chain length.
        for (L = 2; L < n; L++) {
            cost++;
            for (i = 1; i < n - L + 1; i++) {
                cost++;
                j = i + L - 1;
                if (j == n)
                    continue;
                table[i][j] = Integer.MAX_VALUE;
                for (k = i; k <= j - 1; k++) {
                    cost++;
                    q = table[i][k] + table[k + 1][j] + p[i - 1] * p[k] * p[j];
                    cost++;
                    if (q < table[i][j]) {
                        table[i][j] = q;
                        cost++;
                    }
                }
            }
        }
        return table[1][n - 1];
    }
}

/*
OUTPUT

Enter the number of matrix you want to multiply :: 11

Generating dimensions for 12 Matrices...

This are the Matrices Dimension
40 20 50 30 30 20 50 40 10 10 30 30
Cost :: 2152
Minimum number of multiplications :: 101000
Time Taken :: 37300 nanoseconds

Do you want to continue ? [y/n] y
 */
```

Chain Matrix Multiplication Problem Analysis

Sr. no	No. of Matrix	Multiplications	Cost	Time (ns)
1	2	6000	2159	8300
2	3	16000	2178	5000
3	4	28000	86	7300
4	5	9000	159	10600
5	6	6000	279	12300
6	7	47000	461	18200
7	8	73000	725	27200
8	9	282000	1075	30100
9	10	90000	1541	29100
10	11	101000	2125	37300



Practical 8:

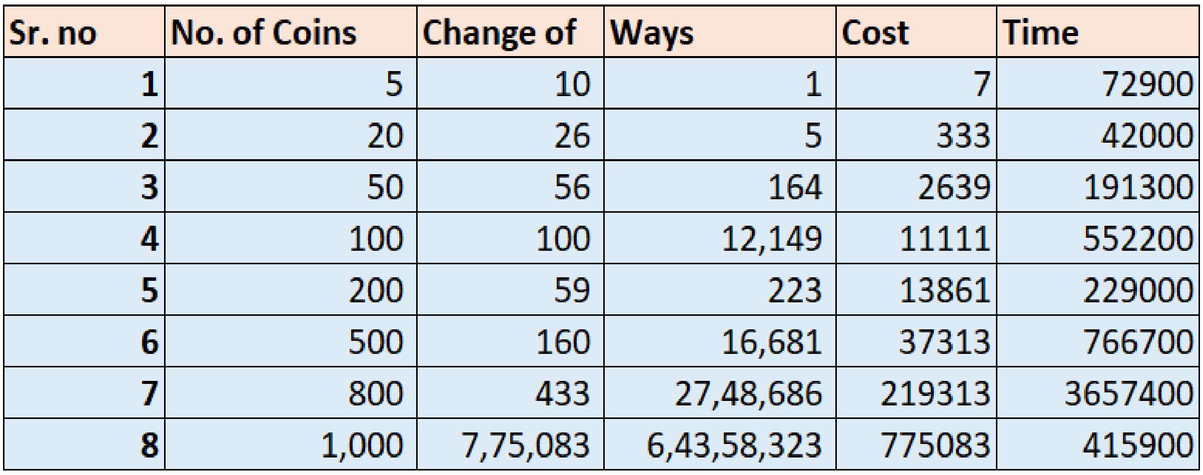
To implement the “Making Change” problem usingdynamic programming.

Code Implementation::

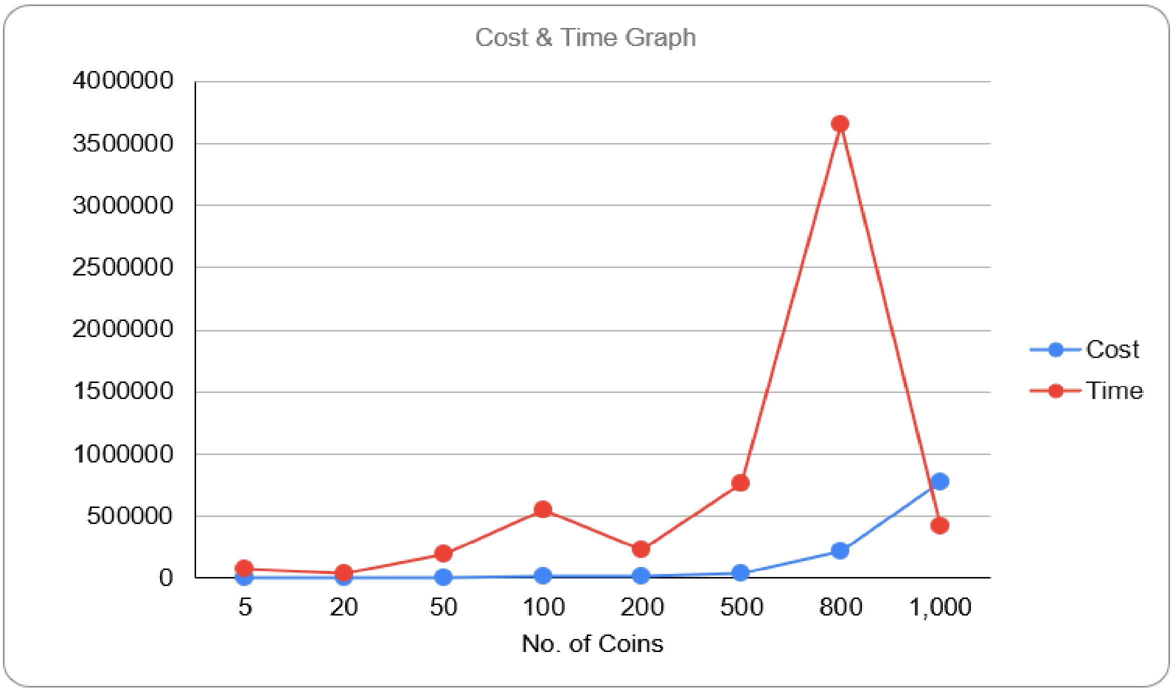
```
package Practical_Package;
import java.util.*;

public class P8_TimeAnalysisCoinChange {
    public static long cost = 0;
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random rd = new Random();
        String answer;
        do {
            System.out.print("Enter the number of coins you have available :: ");
            int arraySize = sc.nextInt();

            System.out.println("\nCollecting the Coins...");
            ArrayList<Integer> arr = new ArrayList<>(arraySize);
            for (int i = 0; i < arraySize; i++) {
                int rNum = rd.nextInt(arraySize) + 10;
                if (arr.contains(rNum)) {
                    i--;
                } else
                    arr.add(rNum); //preventing zero
            }
            System.out.print("Coins Collected are :: ");
            System.out.println(arr);

            int toChange = rd.nextInt(arraySize) + 10; //preventing zero
            System.out.println("\nTrying to make a change of " + toChange + "...");

            long startTime = System.nanoTime();
            long ans = countWays(arr, arraySize, toChange);
            long endTime = System.nanoTime();

            System.out.println("\nCost :: " + cost);
            System.out.println("Total Ways :: " + ans);
            System.out.println("Time Taken :: " + (endTime - startTime) + "
nanoseconds");
            System.out.print("\nDo you want to continue? [y/n] :: ");
            answer = sc.next().toLowerCase(Locale.ROOT);
        } while (answer.charAt(0) != 'n');
    }
    static long countWays(ArrayList<Integer> coins, int size, int targetAmount) {
        long[] table = new long[targetAmount + 1];

        // initially the table contains 0 values
        Arrays.fill(table, 0);
        cost += size;

        //base case
        table[0] = 1;

        for (int i = 0; i < size; i++)
            for (int j = coins.get(i); j <= targetAmount; j++) {
                table[j] += table[j - coins.get(i)];
                cost += 2;
            }
        return table[targetAmount];
    }
}
/*
OUTPUT
```

```
Enter the number of coins you have available :: 20

Collecting the Coins...
Coins Collected are :: [23, 22, 21, 12, 14, 28, 15, 13, 17, 16, 11, 19, 27, 26, 18,
24, 20, 29, 25, 10]

Trying to make a change of 22...

Cost :: 202
Total Ways :: 3
Time Taken :: 67800 nanoseconds

Do you want to continue? [y/n] :: n
 */
```

Coin Change Problem Analysis



Sr. no	No. of Coins	Change of	Ways	Cost	Time
1	5	10	1	7	72900
2	20	26	5	333	42000
3	50	56	164	2639	191300
4	100	100	12,149	11111	552200
5	200	59	223	13861	229000
6	500	160	16,681	37313	766700
7	800	433	27,48,686	219313	3657400
8	1,000	7,75,083	6,43,58,323	775083	415900



Practical 9:

To Implement LCS Problem.

Code Implementation::

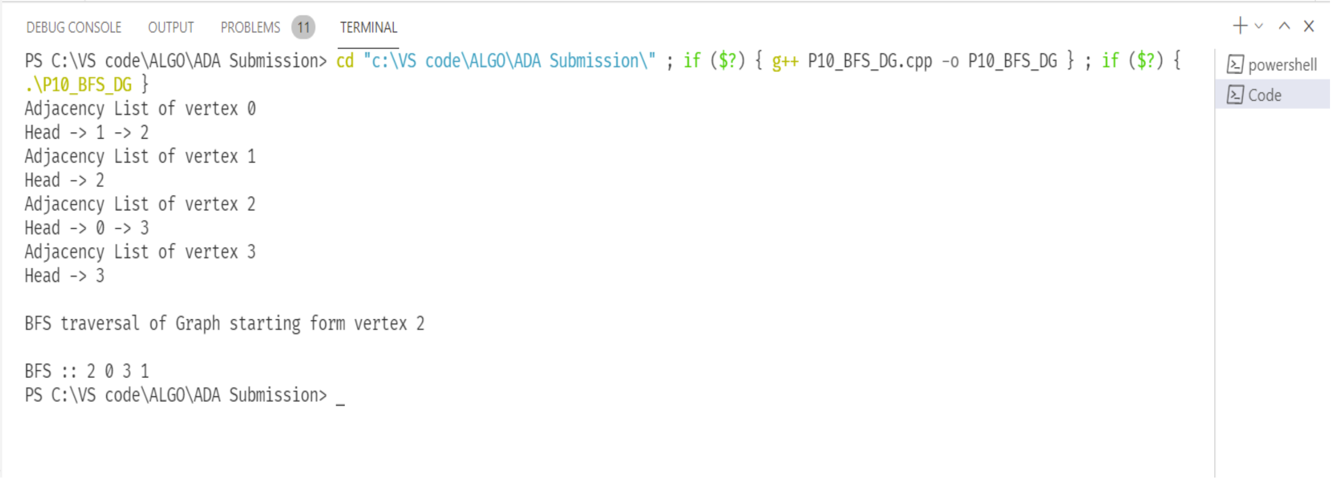
```java
package Practical_Package;
public class P9_TimeAnalysisLCS {
    public static void main(String[] args) {

        String S1 = "IMPOSSIBLE";
        String S2 = "POSSIBLE";
        int m = S1.length();
        int n = S2.length();

        long startTime = System.currentTimeMillis();
        lcs(S1, S2, m, n);
        long endTime = System.currentTimeMillis();
        System.out.println("\n\nTime Taken " + (endTime - startTime) + " 
milliseconds");
    }
    static void lcs(String S1, String S2, int m, int n) {
        int[][] table = new int[m + 1][n + 1];

        // Building in bottom-up way
        for (int i = 0; i <= m; i++) {
            for (int j = 0; j <= n; j++) {
                if (i == 0 || j == 0)
                    table[i][j] = 0;
                else if (S1.charAt(i - 1) == S2.charAt(j - 1))
                    table[i][j] = table[i - 1][j - 1] + 1;
                else
                    table[i][j] = Math.max(table[i - 1][j], table[i][j - 1]);
            }
        }
        int index = table[m][n];
        int temp = index;

        char[] lcs = new char[index + 1];
        lcs[index] = '\0';
        int i = m, j = n;
        while (i > 0 && j > 0) {
            if (S1.charAt(i - 1) == S2.charAt(j - 1)) {
                lcs[index - 1] = S1.charAt(i - 1);
                i--;
                j--;
                index--;
            } else if (table[i - 1][j] > table[i][j - 1])
                i--;
            else
                j--;
        }
        // Printing the subSequences
        System.out.print("S1 : " + S1 + "\nS2 : " + S2 + "\nLCS: ");
        for (int k = 0; k <= temp; k++)
            System.out.print(lcs[k]);
    }
}

/*
S1 : IMPOSSIBLE
S2 : POSSIBLE
LCS: POSSIBLE

Time Taken 24 milliseconds
 */
```

Practical 10.1:

To implement graph searching algorithms using ***BFS*** and DFS algorithms

Code Implementation::

```
//Program to do BFS traversal in directed graph
#include <bits/stdc++.h>
using namespace std;

class Graph
{
private:
    int V;
    vector<int> *adj;

public:
    Graph(int);
    void add_edge(int, int);
    void display();
    void BFS(int);
};

Graph::Graph(int v)
{
    this->V = v;
    adj = new vector<int>[V];
}

void Graph::add_edge(int u, int v)
{
    adj[u].push_back(v);
}

void Graph::display()
{
    for (int i = 0; i < V; i++)
    {
        cout << "Adjacency List of vertex " << i << "\nHead";
        for (int x : adj[i])
            cout << " -> " << x;
        cout << endl;
    }
}

void Graph::BFS(int s)
{
    //mark all vertices as not visited
    bool *visited = new bool[V];
    for (int i = 0; i < V; i++)
        visited[i] = false;

    //make a list/queue to store levels
    queue<int> q;

    //mark the source node as visited and enqueue it
    visited[s] = true;
    q.push(s);

    while (!q.empty())
    {
        //1. get first vertex, print, dequeue it
        s = q.front();
        cout << s << " ";
        q.pop();

        //2 enqueue all the adj vertices of s
        for (auto it = adj[s].begin(); it != adj[s].end(); it++)
        {
            if (!visited[*it])
            {
                visited[*it] = true;
                q.push(*it);
            }
        }
    }
}

int main()
{
    Graph g(4);

    g.add_edge(0, 1);
    g.add_edge(0, 2);
    g.add_edge(1, 2);
    g.add_edge(2, 0);
    g.add_edge(2, 3);
    g.add_edge(3, 3);

    g.display();

    cout << "\nBFS traversal of Graph starting form vertex 2" << endl;
    cout << "\nBFS :: ";
    g.BFS(2);

    return 0;
}

/*
OUTPUT

Adjacency List of vertex 0
Head -> 1 -> 2
Adjacency List of vertex 1
Head -> 2
Adjacency List of vertex 2
Head -> 0 -> 3
Adjacency List of vertex 3
Head -> 3

BFS traversal of Graph starting form vertex 2

BFS :: 2 0 3 1
*/
```



```
PS C:\VS code\ALGO\ADA Submission> cd "c:\VS code\ALGO\ADA Submission\" ; if ($?) { g++ P10_BFS_DG.cpp -o P10_BFS_DG } ; if ($?) {
.\P10_BFS_DG }
Adjacency List of vertex 0
Head -> 1 -> 2
Adjacency List of vertex 1
Head -> 2
Adjacency List of vertex 2
Head -> 0 -> 3
Adjacency List of vertex 3
Head -> 3

BFS traversal of Graph starting form vertex 2

BFS :: 2 0 3 1
PS C:\VS code\ALGO\ADA Submission> _
```

(Output will be same for Practical 10.1 & 10.2)

Practical 10.2:
To implement graph searching algorithms using BFS and ***DFS*** algorithms

Code Implementation::

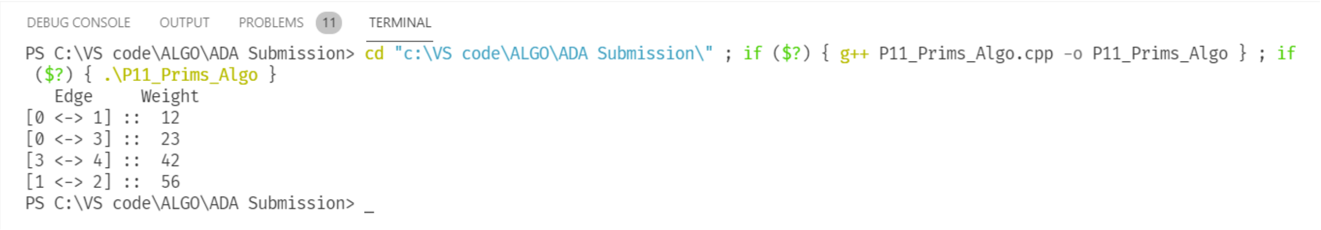
```
//program to do DFS traversal of directed graph
#include <bits/stdc++.h>
using namespace std;

class Graph
{
    int V;
    vector<int> *adj;
public:
    Graph(int);
    void add_edge(int, int);
    void display();
    void DFS(int);

    //maintaining a bool visited vector for DFS traversal
    bool *visited;
};
Graph::Graph(int V)
{
    this->V = V;
    adj = new vector<int>[V];
    visited = new bool[V];

    //initially all nodes must not be visited
    for (int i = 0; i < V; i++)
        visited[i] = false;
}
void Graph::add_edge(int u, int v)
{
    adj[u].push_back(v);
}
void Graph::display()
{
    for (int i = 0; i < V; i++)
    {
        cout << "\nAdjacency List of vertex " << i << "\nHead";
        for (int x : adj[i])
            cout << " -> " << x;
        cout << endl;
    }
}
```

```
void Graph::DFS(int s)
{
    //mark the current source node as visited and print it
    visited[s] = true;
    cout << s << " ";

    // Recur for all the vertices adjacent to this vertex
    for (auto it = adj[s].begin(); it != adj[s].end(); it++)
        if (!visited[*it])
            DFS(*it);
}
int main()
{
    Graph g(4);
    g.add_edge(0, 1);
    g.add_edge(0, 2);
    g.add_edge(1, 2);
    g.add_edge(2, 0);
    g.add_edge(2, 3);
    g.add_edge(3, 3);

    g.display();

    cout << "\nFollowing is Depth First Traversal (starting from vertex 2)" << endl;
    cout << "DFS ";
    g.DFS(2);
    return 0;
}
/*
OUTPUT

Adjacency List of vertex 0
Head -> 1 -> 2
Adjacency List of vertex 1
Head -> 2

Adjacency List of vertex 2
Head -> 0 -> 3

Adjacency List of vertex 3
Head -> 3

Following is Depth First Traversal (starting from vertex 2)
DFS 2 0 1 3
*/
```

Practical 11:
To implement PRIM’s algorithm.

Code Implementation::

```
#include <cstring>
#include <iostream>
using namespace std;

#define INF 9999999
#define V 5

int Graph[V][V] = {
    {51, 12, 89, 23, 0},
    {9, 0, 56, 0, 59},
    {75, 0, 0, 0, 26},
    {11, 19, 0, 0, 42},
    {15, 54, 77, 31, 75}};

int main()
{
    int totalEdges;
    int selected[V];

    memset(selected, false, sizeof(selected));

    totalEdges = 0;
    selected[0] = true;

    int x; //  row number
    int y; //  col number

    // print for edge and weight
    cout << "   Edge" << "   " << "  Weight" << endl;

    while (totalEdges < V - 1)
    {
        int min = INF;
        x = 0;
        y = 0;

        for (int i = 0; i < V; i++)
        {
            if (selected[i])
            {
                for (int j = 0; j < V; j++)
                {
                    if (!selected[j] && Graph[i][j])
                    { // not in selected and there is an edge
                        if (min > Graph[i][j])
                        {
                            min = Graph[i][j];
                            x = i;
                            y = j;
                        }
                    }
                }
            }
        }
        cout << "[" << x << " <-> " << y << "]" << " ::  " << Graph[x][y];
        cout << endl;
        selected[y] = true;
        totalEdges++;
    }
    return 0;
}

/*
OUTPUT

   Edge     Weight
[0 <-> 1] ::  12
[0 <-> 3] ::  23
[3 <-> 4] ::  42
[1 <-> 2] ::  56
*/
```



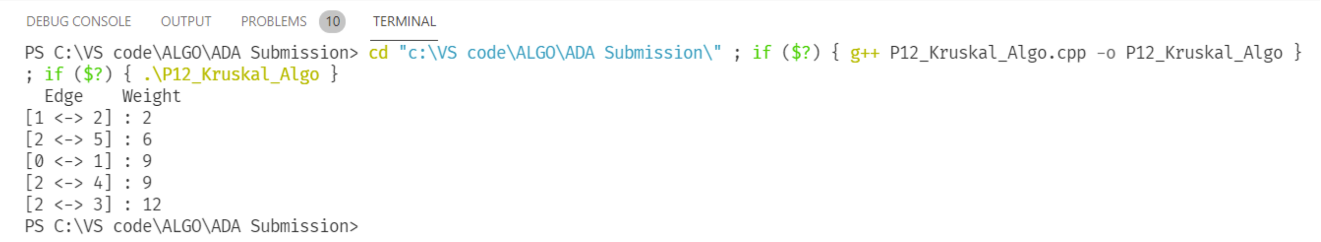
```
PS C:\VS code\ALGO\ADA Submission> cd "c:\VS code\ALGO\ADA Submission\" ; if ($?) { g++ P11_Prims_Algo.cpp -o P11_Prims_Algo } ; if
 ($?) { .\P11_Prims_Algo }
   Edge     Weight
[0 <-> 1] ::  12
[0 <-> 3] ::  23
[3 <-> 4] ::  42
[1 <-> 2] ::  56
PS C:\VS code\ALGO\ADA Submission> _
```

Practical 12:
To implement PRIM’s algorithm.

```
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

#define edge pair<int, int>

class Graph
{
private:
    vector<pair<int, edge>> G;
    vector<pair<int, edge>> T;
    int *parent;
    int V;
public:
    Graph(int V);
    void AddWeightedEdge(int u, int v, int w);
    int find_set(int i);
    void union_set(int u, int v);
    void kruskal();
    void printGraph();
};
Graph::Graph(int V)
{
    parent = new int[V];

    for (int i = 0; i < V; i++)
        parent[i] = i;

    G.clear();
    T.clear();
}
void Graph::AddWeightedEdge(int u, int v, int w)
{
    G.push_back(make_pair(w, edge(u, v)));
}
int Graph::find_set(int i)
{
    if (i == parent[i])
        return i;
    else
        return find_set(parent[i]);
}
```

```
void Graph::union_set(int u, int v)
{
    parent[u] = parent[v];
}
void Graph::kruskal()
{
    int i, uRep, vRep;
    sort(G.begin(), G.end());
    for (i = 0; i < G.size(); i++)
    {
        uRep = find_set(G[i].second.first);
        vRep = find_set(G[i].second.second);
        if (uRep != vRep)
        {
            T.push_back(G[i]);
            union_set(uRep, vRep);
        }
    }
}
void Graph::printGraph()
{
    cout << "  Edge"
         << "    Weight" << endl;
    for (int i = 0; i < T.size(); i++)
    {
        cout << "[" << T[i].second.first << " <-> " << T[i].second.second << "]" <<
" : "
             << T[i].first;
        cout << endl;
    }
}
int main()
{
    Graph g(6);
    g.AddWeightedEdge(0, 1, 9);
    g.AddWeightedEdge(0, 2, 9);
    g.AddWeightedEdge(1, 2, 2);
    g.AddWeightedEdge(1, 0, 9);
    g.AddWeightedEdge(2, 0, 9);
    g.AddWeightedEdge(2, 1, 6);
    g.AddWeightedEdge(2, 3, 12);
    g.AddWeightedEdge(2, 5, 6);
    g.AddWeightedEdge(2, 4, 9);
    g.AddWeightedEdge(3, 2, 12);
    g.AddWeightedEdge(3, 4, 12);
```

```
        g.AddWeightedEdge(4, 2, 9);
        g.AddWeightedEdge(4, 3, 12);
        g.AddWeightedEdge(5, 2, 6);
        g.AddWeightedEdge(5, 4, 12);
        g.kruskal();
        g.printGraph();

        return 0;
    }

    /*
    OUTPUT

      Edge     Weight
    [1 <-> 2] : 2
    [2 <-> 5] : 6
    [0 <-> 1] : 9
    [2 <-> 4] : 9
    [2 <-> 3] : 12
    */
```



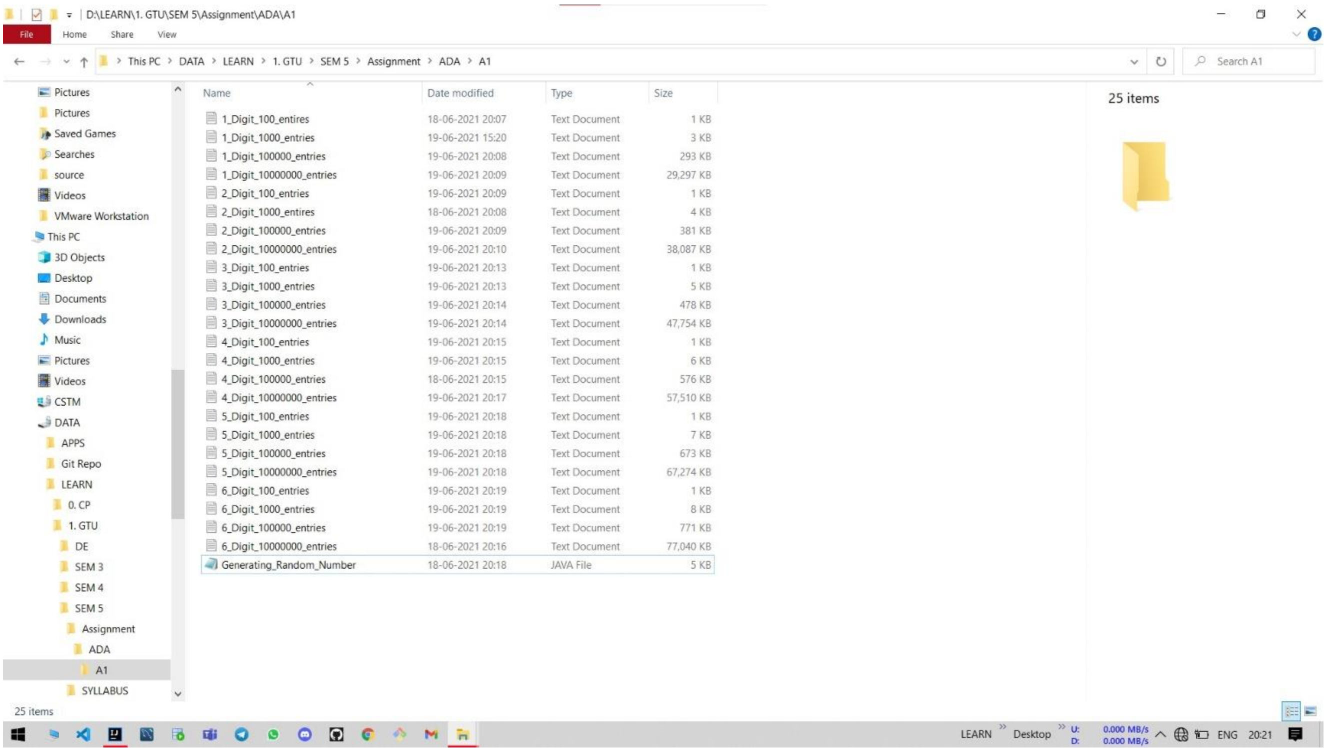
```
PS C:\VS code\ALGO\ADA Submission> cd "c:\VS code\ALGO\ADA Submission\" ; if ($?) { g++ P12_Kruskal_Algo.cpp -o P12_Kruskal_Algo }
; if ($?) { .\P12_Kruskal_Algo }
  Edge    Weight
[1 <-> 2] : 2
[2 <-> 5] : 6
[0 <-> 1] : 9
[2 <-> 4] : 9
[2 <-> 3] : 12
PS C:\VS code\ALGO\ADA Submission>
```

Assignment 1:
To generate sequence of random numbers of 1, 2, 3, 4, 5, and 6 digits.

Code Implementation::

```
package Assignment_Package;

import java.io.File;
import java.io.PrintWriter;
import java.util.Random;
import java.util.Scanner;

public class GeneratingRandomNumber {

    public static void main(String[] args) {
        Random rd = new Random();
        Scanner sc = new Scanner(System.in);
        String answer;

        do {
            //getting total number of random number user want to generate.
            System.out.print("\nHow much entries did you want ? \n[ex. 10000000] :: 
");
            int totalEntries = sc.nextInt();

            //getting file name to generate all random number into it.
            System.out.print("\nEnter file name in which you want to copy all random 
numbers. \n[ex. 2_Digit_1000_entries.txt] :: ");
            String fileName = sc.next();

            //creating a file object
            File file = new File(fileName);

            //exit if file already present
            if (file.exists()) {
                System.out.println(fileName + " Named file already exists on your 
local machine :(");
                System.out.println("exiting...");
                System.exit(1);
            }

            try (
                    //creating Print Writer object to write data to a file
                    PrintWriter output = new PrintWriter(file)
            ) {
                System.out.println("\n1. Integers\n2. Double\nElse. Exit");
                System.out.print("Enter your choice :: ");
                int choice = sc.nextInt();

                switch (choice) {
                    case 1: {
                        //getting the digits for the generating random form the 
user.

                        System.out.print("\n1. 1 digit [1 - 9]\n2. 2 digits [1 - 
99]");

                        System.out.print("\n3. 3 digits [1 - 999]\n4. 4 digits [1 - 
9999]");

                        System.out.print("\n5. 5 digits [1 - 99999]\n6. 6 digits [1 
- 999999]");

                        System.out.print("\nElse give the last limit [ex. 999999]");
                        System.out.print("\n\nEnter your choice :: ");
                        int digit = sc.nextInt();
                        int bound;

                        switch (digit) {
                            case 1 -> bound = 10;
                            case 2 -> bound = 100;
                            case 3 -> bound = 1000;
                            case 4 -> bound = 10000;
                            case 5 -> bound = 100000;
                            case 6 -> bound = 1000000;
                            default -> bound = digit;
                        }

                        //writing random number to the file of total entries' user
want
                        for (int i = 0; i < totalEntries; i++) {

                            //the bound is excluded so adding last limit manually
                            output.println(rd.nextInt(bound));
                        }
                    }
                    break;

                    case 2: {
                        //getting the digits for the generating random form the
user.
                        System.out.print("\n1. 1 digit [0.0 - 0.9]\n2. 2 digits [1.0
- 9.9]");
                        System.out.print("\n3. 3 digits [10.0 - 99.9]\n4. 4 digits
[1.000 - 999.9]");
                        System.out.print("\n5. 5 digits [1.0000 - 9999.9]\n6. 6
digits [1.00000 - 99999.9]");
//                        System.out.print("\n7. give the last limit [ex.
999999]\nElse. Exit");
                        System.out.print("\n\nEnter your choice :: ");
                        int digit = sc.nextInt();
                        double bound = 0.0;

                        switch (digit) {
                            case 1 -> bound = 0.9;
                            case 2 -> bound = 9.9;
                            case 3 -> bound = 99.9;
                            case 4 -> bound = 999.9;
                            case 5 -> bound = 9999.9;
                            case 6 -> bound = 99999.9;
                            case 7 -> bound = (double) digit;
                            default -> System.exit(1);
                        }
                        //writing random number to the file of total entries' user
want
                        for (int i = 0; i < totalEntries; i++) {

                            //the bound is excluded so adding last limit manually
                            output.println(rd.nextDouble() * bound);
                        }
                    }
                    break;

                    default:
                        System.exit(1);
                }

            } catch (Exception e) {
                e.printStackTrace();
            }

            System.out.println("\nFile Created and Random number generated in
it!!");

            //prompt user if he wants to continue or not.
            System.out.print("\nDo you want to continue ? [Y/N] ");
            answer = sc.next().toLowerCase();
            if (answer.charAt(0) == 'n')
                break;

        } while (true);

        System.out.println("\nHave a great day :)");
    }
}

/*
How much entries did you want ?
[ex. 10000000] :: 100

Enter file name in which you want to copy all random numbers.
[ex. 2_Digit_1000_entries.txt] :: sr5.txt

1. Integers
2. Double
Else. Exit
Enter your choice :: 2

1. 1 digit [0.0 - 0.9]
2. 2 digits [1.0 - 9.9]
3. 3 digits [10.0 - 99.9]
4. 4 digits [1.000 - 999.9]
5. 5 digits [1.0000 - 9999.9]
6. 6 digits [1.00000 - 99999.9]

Enter your choice :: 5

File Created and Random number generated in it!!

Do you want to continue ? [Y/N] n

Have a great day :)
 */
```

Screen shot of files created



Name	Date modified	Type	Size
1_Digit_100_entires	18-06-2021 20:07	Text Document	1 KB
1_Digit_1000_entries	19-06-2021 15:20	Text Document	3 KB
1_Digit_100000_entries	19-06-2021 20:08	Text Document	293 KB
1_Digit_10000000_entries	19-06-2021 20:09	Text Document	29,297 KB
2_Digit_100_entries	19-06-2021 20:09	Text Document	1 KB
2_Digit_1000_entires	18-06-2021 20:08	Text Document	4 KB
2_Digit_100000_entries	19-06-2021 20:09	Text Document	381 KB
2_Digit_10000000_entries	19-06-2021 20:10	Text Document	38,087 KB
3_Digit_100_entries	19-06-2021 20:13	Text Document	1 KB
3_Digit_1000_entries	19-06-2021 20:13	Text Document	5 KB
3_Digit_100000_entries	19-06-2021 20:14	Text Document	478 KB
3_Digit_10000000_entries	19-06-2021 20:14	Text Document	47,754 KB
4_Digit_100_entries	19-06-2021 20:15	Text Document	1 KB
4_Digit_1000_entries	19-06-2021 20:15	Text Document	6 KB
4_Digit_100000_entries	18-06-2021 20:15	Text Document	576 KB
4_Digit_10000000_entries	19-06-2021 20:17	Text Document	57,510 KB
5_Digit_100_entries	19-06-2021 20:18	Text Document	1 KB
5_Digit_1000_entries	19-06-2021 20:18	Text Document	7 KB
5_Digit_100000_entries	19-06-2021 20:18	Text Document	673 KB
5_Digit_10000000_entries	19-06-2021 20:18	Text Document	67,274 KB
6_Digit_100_entries	19-06-2021 20:19	Text Document	1 KB
6_Digit_1000_entries	19-06-2021 20:19	Text Document	8 KB
6_Digit_100000_entries	19-06-2021 20:19	Text Document	771 KB
6_Digit_10000000_entries	18-06-2021 20:16	Text Document	77,040 KB
Generating_Random_Number	18-06-2021 20:18	JAVA File	5 KB

Assignment 2:
To implement Insertion Sort Algorithm and Sort the random numbers generated, by Considering Cost and Time.

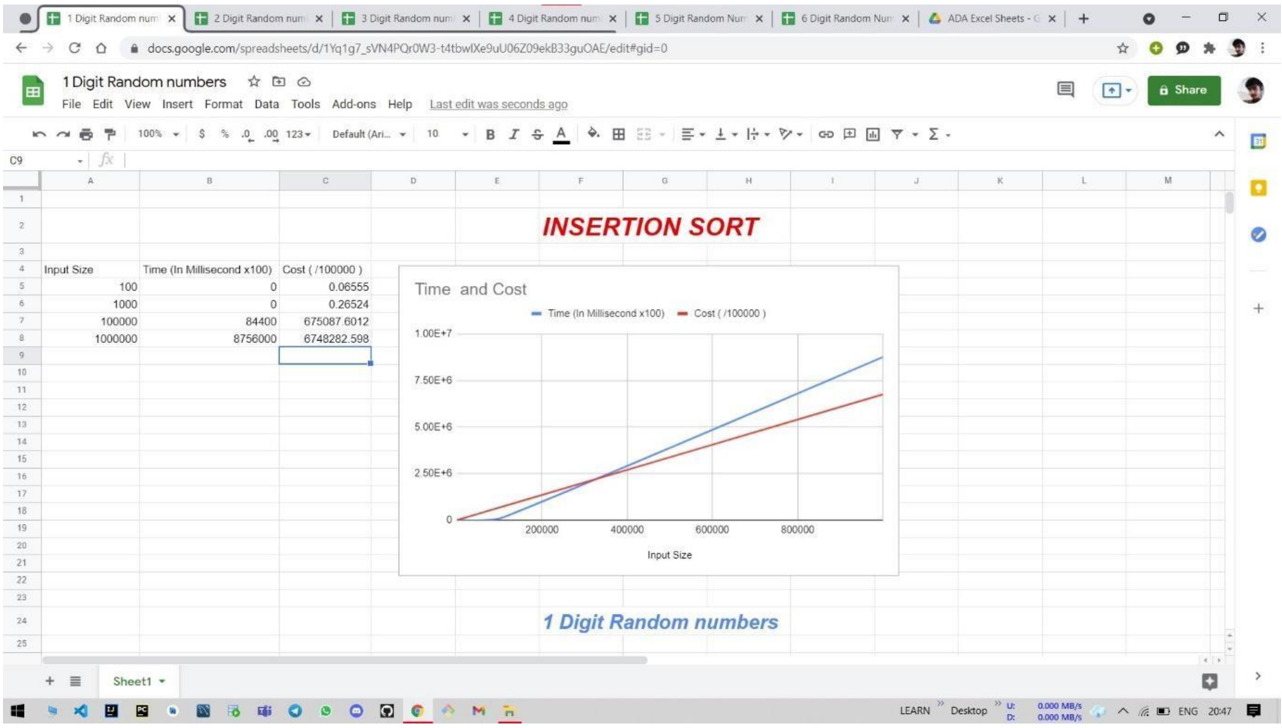
Code Implementation::

Code is same as of Practical 1

Readings::

Graph for 1 Digit Random Numbers

Input Size	***Time (In ms)***	***Cost***
100	0	6555
1000	0	26524
100000	844	6750876012
1000000	87560	674828259774



1 Digit Random numbers

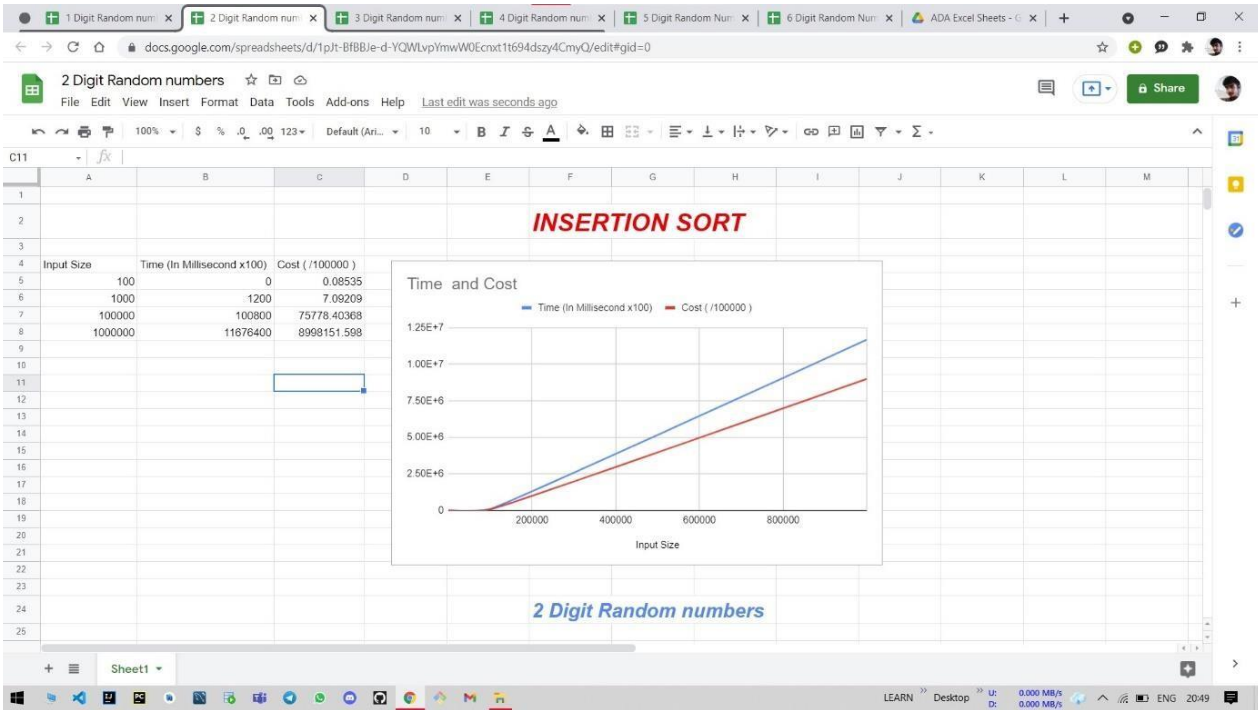
INSERTION SORT

Input Size	Time (In Millisecond x100)	Cost (/100000)
100	0	0.06555
1000	0	0.26524
100000	84400	675087.6012
1000000	8756000	6748282.598

1 Digit Random numbers

Graph For 2 Digit Random Numbers

Input Size	***Time (In ms)***	***Cost***
100	0	8535
1000	12	709209
100000	1008	7577840368
1000000	116764	8998115159847



2 Digit Random numbers

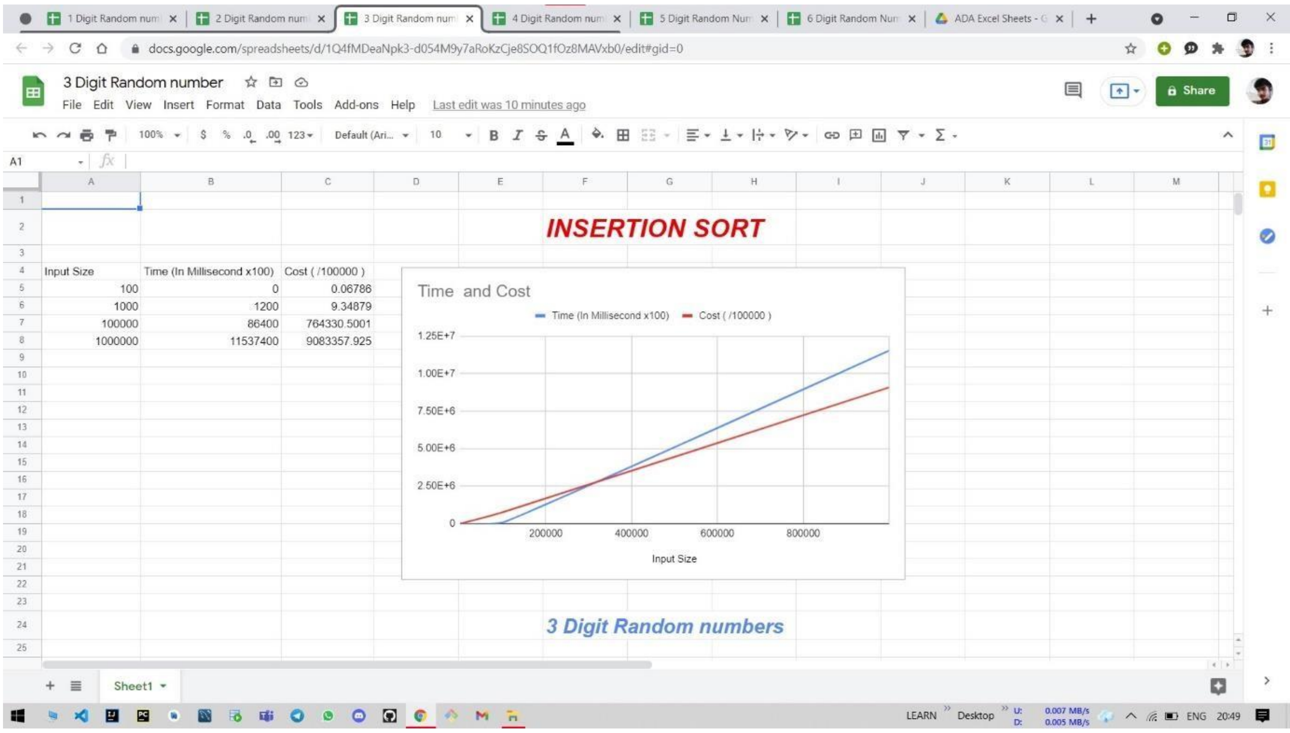
INSERTION SORT

Input Size	Time (In Millisecond x100)	Cost (/100000)
100	0	0.08535
1000	1200	7.09209
100000	100800	75778.40368
1000000	11676400	8998151.598

Time and Cost

Graph For 3 Digit Random Numbers

Input Size	Time (In ms)	Cost
100	0	6768
1000	12	709209
100000	864	7643305001
1000000	115374	908335792536



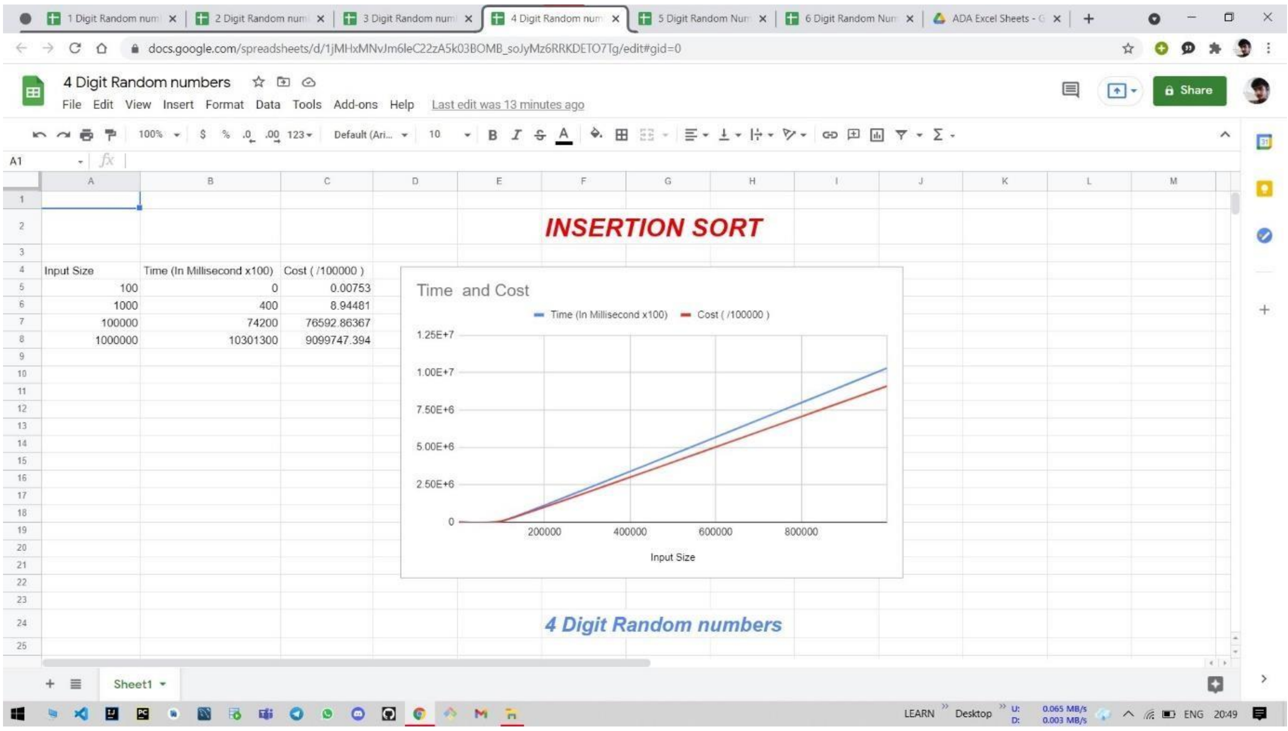
INSERTION SORT

Input Size	Time (In Millisecond x100)	Cost (/100000)
100	0	0.06786
1000	1200	9.34879
100000	86400	764330.5001
1000000	11537400	9083357.925

3 Digit Random numbers

Graph For 4 Digit Random Numbers

Input Size	***Time (In ms)***	***Cost***
100	0	7530
1000	4	894481
100000	742	7659286367
1000000	115374	909974739402


4 Digit Random numbers
File Edit View Insert Format Data Tools Add-ons Help
Last edit was 13 minutes ago
Share
Sheet1

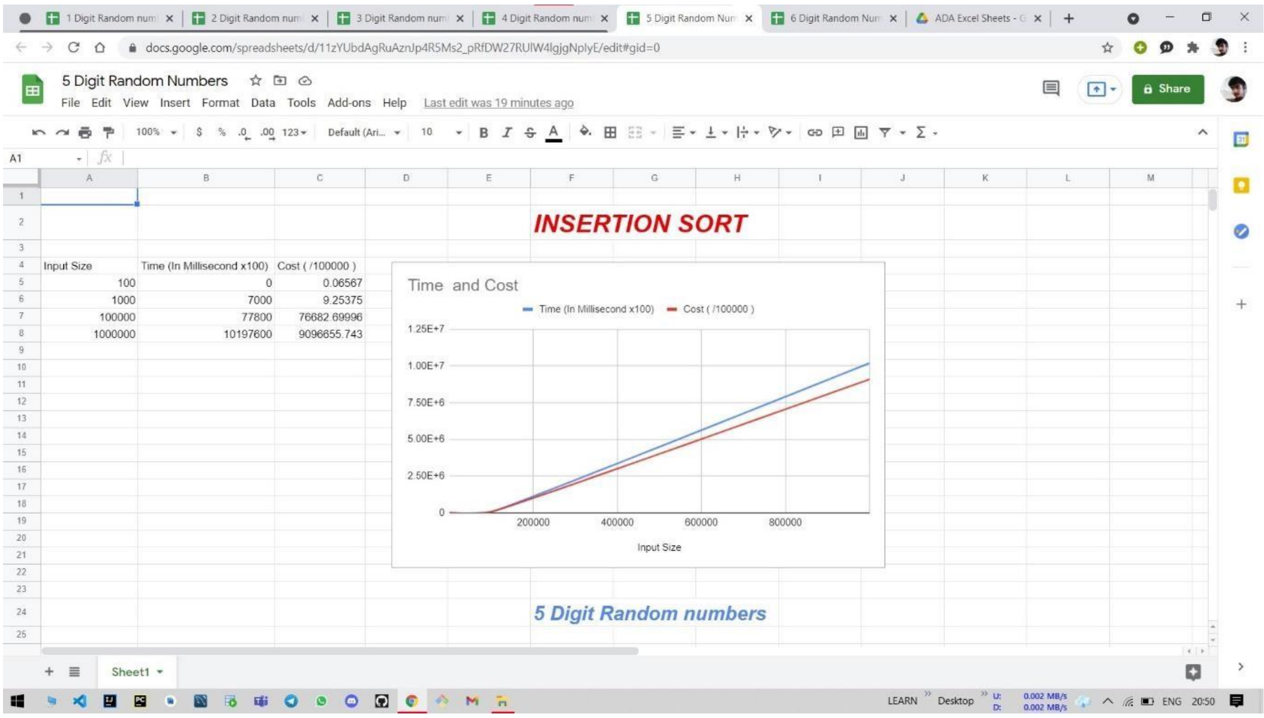
INSERTION SORT

Input Size	Time (In Millisecond x100)	Cost (/100000)
100	0	0.00753
1000	400	8.94481
100000	74200	76592.86367
1000000	10301300	9099747.394

4 Digit Random numbers

Graph For 5 Digit Random Numbers

Input Size	***Time (In ms)***	***Cost***
100	0	6567
1000	7	925395
100000	778	7668269996
1000000	101976	909665574220



5 Digit Random Numbers

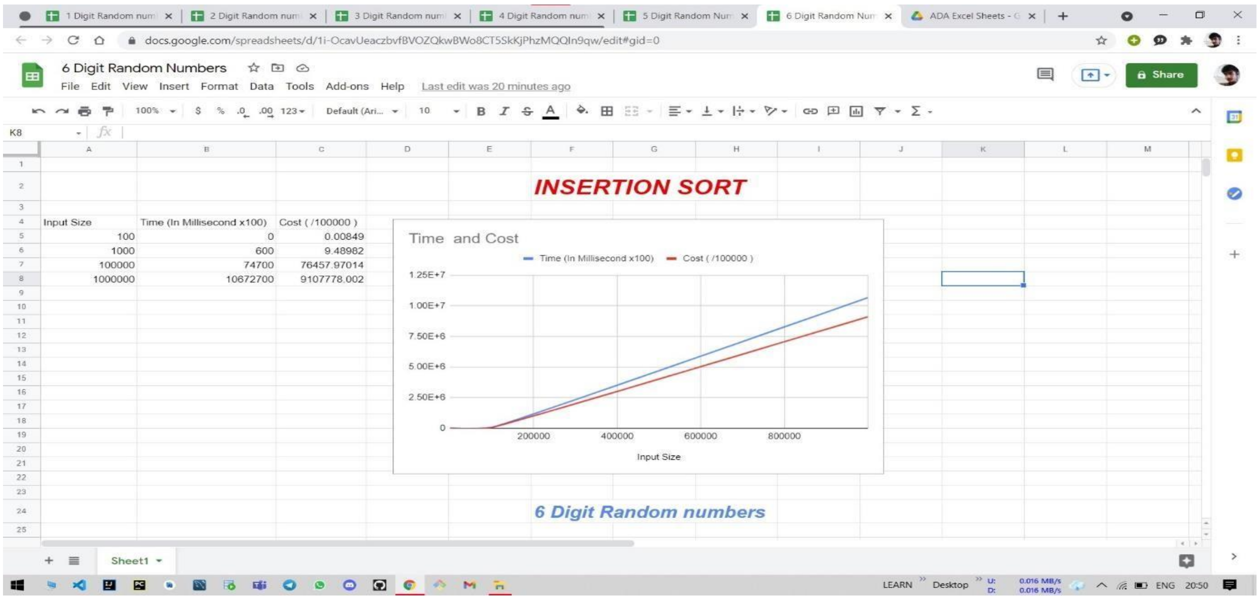
INSERTION SORT

Input Size	Time (In Millisecond x100)	Cost (/100000)
100	0	0.06567
1000	7000	9.25375
100000	77800	76682.69996
1000000	10197600	9096655.743

5 Digit Random numbers

Graph For 6 Digit Random Numbers

Input Size	***Time (In ms)***	***Cost***
100	0	8490
1000	6	948982
100000	747	7645797014
1000000	106727	910777800216



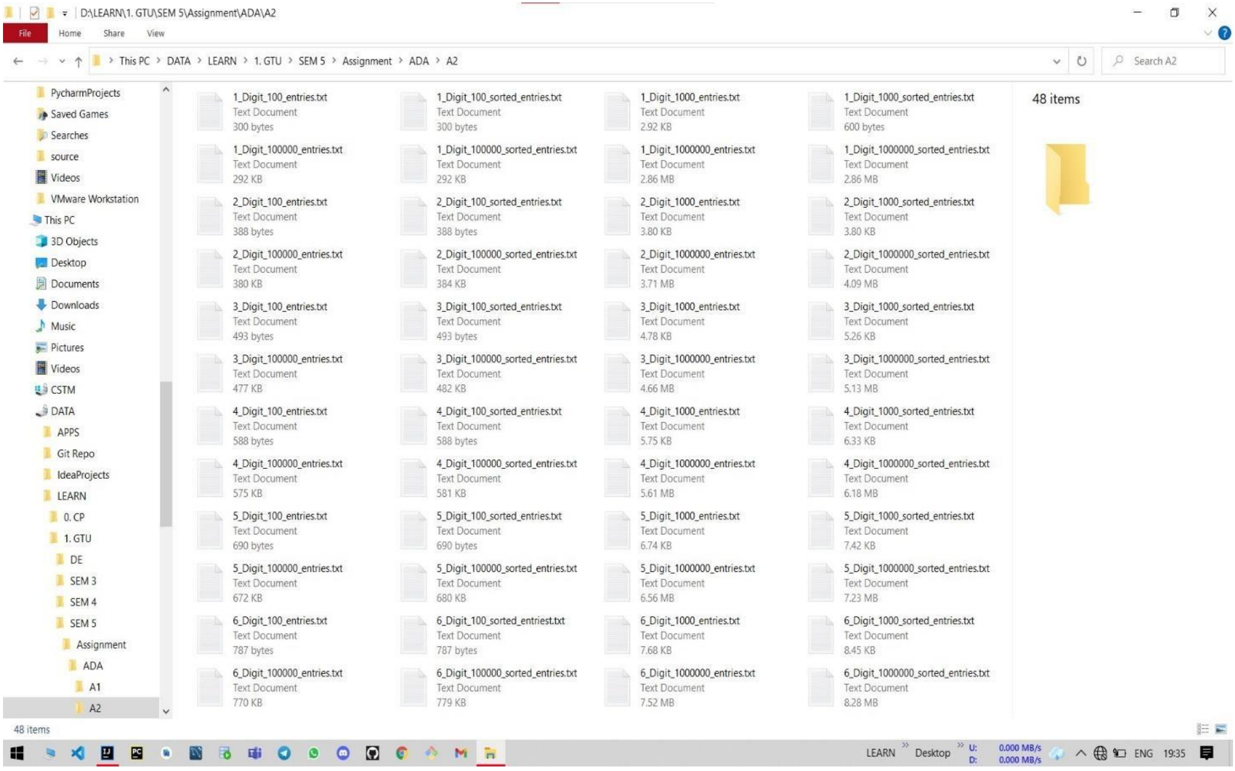
6 Digit Random Numbers

INSERTION SORT

Input Size	Time (In Millisecond x100)	Cost (/100000)
100	0	0.00849
1000	600	9.48982
100000	74700	76457.97014
1000000	10672700	9107778.002

6 Digit Random numbers

Screen shot of sorted and random files



D:\LEARN\1. GTU\SEM 5\Assignment\ADA\A2

This PC > DATA > LEARN > 1. GTU > SEM 5 > Assignment > ADA > A2

1_Digit_100_entries.txt Text Document 300 bytes
1_Digit_100_sorted_entries.txt Text Document 300 bytes
1_Digit_1000_entries.txt Text Document 2.92 KB
1_Digit_1000_sorted_entries.txt Text Document 600 bytes
1_Digit_100000_entries.txt Text Document 292 KB
1_Digit_100000_sorted_entries.txt Text Document 292 KB
1_Digit_1000000_entries.txt Text Document 2.86 MB
1_Digit_1000000_sorted_entries.txt Text Document 2.86 MB
2_Digit_100_entries.txt Text Document 388 bytes
2_Digit_100_sorted_entries.txt Text Document 388 bytes
2_Digit_1000_entries.txt Text Document 3.80 KB
2_Digit_1000_sorted_entries.txt Text Document 3.80 KB
2_Digit_100000_entries.txt Text Document 380 KB
2_Digit_100000_sorted_entries.txt Text Document 384 KB
2_Digit_1000000_entries.txt Text Document 3.71 MB
2_Digit_1000000_sorted_entries.txt Text Document 4.09 MB
3_Digit_100_entries.txt Text Document 493 bytes
3_Digit_100_sorted_entries.txt Text Document 493 bytes
3_Digit_1000_entries.txt Text Document 4.78 KB
3_Digit_1000_sorted_entries.txt Text Document 5.26 KB
3_Digit_100000_entries.txt Text Document 477 KB
3_Digit_100000_sorted_entries.txt Text Document 482 KB
3_Digit_1000000_entries.txt Text Document 4.66 MB
3_Digit_1000000_sorted_entries.txt Text Document 5.13 MB
4_Digit_100_entries.txt Text Document 588 bytes
4_Digit_100_sorted_entries.txt Text Document 588 bytes
4_Digit_1000_entries.txt Text Document 5.75 KB
4_Digit_1000_sorted_entries.txt Text Document 6.33 KB
4_Digit_100000_entries.txt Text Document 575 KB
4_Digit_100000_sorted_entries.txt Text Document 581 KB
4_Digit_1000000_entries.txt Text Document 5.61 MB
4_Digit_1000000_sorted_entries.txt Text Document 6.18 MB
5_Digit_100_entries.txt Text Document 690 bytes
5_Digit_100_sorted_entries.txt Text Document 690 bytes
5_Digit_1000_entries.txt Text Document 6.74 KB
5_Digit_1000_sorted_entries.txt Text Document 7.42 KB
5_Digit_100000_entries.txt Text Document 672 KB
5_Digit_100000_sorted_entries.txt Text Document 680 KB
5_Digit_1000000_entries.txt Text Document 6.56 MB
5_Digit_1000000_sorted_entries.txt Text Document 7.23 MB
6_Digit_100_entries.txt Text Document 787 bytes
6_Digit_100_sorted_entriest.txt Text Document 787 bytes
6_Digit_1000_entries.txt Text Document 7.68 KB
6_Digit_1000_sorted_entries.txt Text Document 8.45 KB
6_Digit_100000_entries.txt Text Document 770 KB
6_Digit_100000_sorted_entries.txt Text Document 779 KB
6_Digit_1000000_entries.txt Text Document 7.52 MB
6_Digit_1000000_sorted_entries.txt Text Document 8.28 MB

48 items

Assignment 3:
OEP Problem

BRTS Route Design Problem

Problem Statement

Bus Rapid Transit System is a very popular system of public transport that takes people on dedicated routes from major city spots to distant locations.
A thorough analysis is necessary to implement this system because there is a huge infrastructure cost for making dedicated lanes, bus stops, signaling technology, and of course the bus itself. If you are given this problem and you are supposed to find an optimal solution where your algorithm provides optimal route design, what kind of input parameters will you require? How many of such specific arguments can be processed by your algorithm? How will you define the optimal solution?

As Per My Observation

This Problem is an optimization problem where objectives are defined, its constraints are determined, and a methodology is selected and validated for obtaining an optimal solution.

Based on this description of the problem, I have proposed a three-layer structure for organizing the problem approach.

1. **Objectives**
2. **Parameters**
3. **Methodology**

My AIM while solving this problem (If I am given this problem and I am supposed to find an optimal solution where my algorithm provides optimal route design) will be as follow:

1. User benefit maximization
2. Operator cost minimization
3. Total welfare maximization
4. Capacity maximization

To Make it MORE optimal I will also include some constraints like:

1. Passenger Maximization
2. Delay Minimization (less than 30 mins.)
3. Each Region must be approachable
4. Fleet Maximization

SOLUTION

THIS IS AN NP-HARD PROBLEM

REASON:

Optimization Problems – An optimization problem asks, “What is the optimal solution to problem X?”

– Examples:

- 0-1 Knapsack
- Fractional Knapsack
- Minimum Spanning Tree
- Decision Problems

A decision problem - is one with a yes/no answer

– Examples:

- Does graph G have an MST of weight ≤ W?

Optimization/Decision Problems

- An optimization problem tries to find an optimal solution
- A decision problem tries to answer a yes/no question
- Many problems will have a decision and optimization versions – Eg: Traveling salesman problem
- optimization: find the hamiltonian cycle of minimum weight • decision: is there a hamiltonian cycle of weight ≤ k

– **Polynomial-time**: O(n2), O(n3), O(1), O(n log n)

– **Not in polynomial time**: O(2n), O(n n), O(n!)

What does NP-hard mean?

A lot of times you can solve a problem by reducing it to a different problem.
I can reduce Problem B to Problem A if, given a solution to Problem A, I can easily construct a solution to Problem B. (In this case, "**easily**" means "**in polynomial time**.“).

- A problem is NP-hard if all problems in NP are polynomial-time reducible to it, ...
- Every problem in NP is reducible to HC in polynomial time. Ex:- TSP is reducible to HC. Example: lcm(m, n) = m * n / gcd 13 (m, n)

SAME WITH THIS PROBLEM, IF WE CONSIDER ALL THE POSSIBLE MODULES THEN THIS PROBLEM IS NOT POSSIBLE TO SOLVE IN POLYNOMIAL TIME IT WILL TAKE EXPONENTIAL TIME.

ADDRESSING THE QUESTIONS

Q1. Does only one design strategy efficiently solve this problem?

No !!, Because Many Optimization Algorithms will be required for the different modules.

Q2. Are we using any data structures?

For single modules like Connecting all Major Areas including Residential and mixed public stops, Yes!! we are using **Graph** Data Structure to represent the **Stops as Node** and **Road as Vertices,** Weights of the graph will be the Cost for reaching the particular destination from the source.

Q3. For the solution to work with an increase in values, what comes out to be the order of growth for my algorithm?

I will be using **Floyd-Warshall Algorithm** for the above particular module Because it will find All Pair Shortest Path from One Location to another Location. So the order of growth of my algorithm will be Theta(n3).

Q4. Am I proving the order of growth for my algorithm using proper mathematical notations?

I think Theta is a perfect notation for this particular problem because it will take n3 time exactly.

Q5. Can I reduce this problem or part of the problem to any solved computational problem in polynomial time?

Yes!! single modules - Connecting all Major Areas including Residential and mixed public stops, we can use **Floyd-Warshall Algorithm** which will take polynomial time and the part of the whole BRTS Problem can be reduced to polynomial time.

Q7. Have I identified properties like loop invariants in my algorithm to prove its correctness?

Partially I Have identified the loop invariants in the Floyd-warshall algorithm**.**

Q8. Am I able to provide the order of growth in BigOh or theta?

Yes, I have provided the order of growth for part of the BRTS problem in theta.

Q9. What is the growth in Omega for my algorithm?

Omega(n3).