```
package com.company;

                    //Problem 1 Assignment2

//Task 1-

 //Time complexity – O(n+k), here k is 100 since range is from 10^5 – 10^7 ->10^5 factor common

//Task2 time complexity – O(nk)

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;


import java.util.Scanner;


public class Main {


  public static void main(String[] args) throws IOException {

        // write your code here

    Scanner scan = new Scanner(System.in);

    int n = scan.nextInt();

    int[] B = new int[101];

    Data[] data = new Data[n];

    Data[] res = new Data[n];

                        //Taking input in array of class data

    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    for (int i = 0; i < n; i++) {


      data[i] = new Data(br.readLine().split(" "));


    }

    int bucketno = scan.nextInt();

    Bucketno[] bucket = new Bucketno[bucketno];

    for (int i = 0; i < bucketno; i++) {

      bucket[i] = new Bucketno();
```

```
            bucket[i].bucketvvalue = scan.nextInt();
        }

        //Code for counting sort on data[i].salary
        for (int i = 0; i <= 100; i++) {
            B[i] = 0;                  //bucket to all 0s
        }

        for (int i = 0; i < n; i++) {   //incrementing bucket counts
                B[data[i].sal_in_bucket] = B[data[i].sal_in_bucket]+1;
        }

        for (int i = 1; i <=100; i++) {     //counting previous elements present
            B[i] = B[i] + B[i-1];
        }
        for (int i = n-1; i >= 0; i--) {       //new sorted array

            res[i] = new Data(data[i].rollno,data[i].name, data[i].salary);
            B[data[i].sal_in_bucket]--;
            System.out.println(res[i].rollno+" "+res[i].name+" "+res[i].salary);

        }
//Task 2 – counting no. of entries in bucket  asked

        for (int i = 0; i < bucketno; i++) {
            for (int j = 0; j < n; j++) {
                if(bucket[i].bucketvvalue == res[j].salary){
                    bucket[i].count++;
                }
            }
```

```
    }
    for (int i = 0; i < bucketno; i++) {
        System.out.println(bucket[i].count);
    }

}

public static class Data{       //Class to store data
    public String rollno;
    public String name;
    public int salary;
    public int sal_in_bucket;

    Data(String[] s){       //Constructors
        rollno = s[0];
        name = s[1];
        salary = Integer.parseInt(s[2]);
        sal_in_bucket = salary/100000;
    }



    public Data(String roll, String nm, int sal) {
        rollno = roll;
        name = nm;
        salary = sal;
        sal_in_bucket = salary/100000;
    }
}
public static class Bucketno{
    int bucketvvalue;
    int count;
    }
}
```

Approach – Made use of Counting sort in sorting the elements that are stored in an array of Data class objects. The sorting is done on sal_in_bucket field in data which ranges from 0 to 100. As the salary range constraints were mentioned. Similarly for task2 the required bucket is found in bucket array and the count in it given as output.

```
package com.company;
                                //Problem 2 assignment 2
//Time and space complexity – O(k) as radix sort is used
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws IOException {

        Scanner scan = new Scanner(System.in);
        int k = scan.nextInt();
        double max = (k*k -1);        //max upperlimit on input allowed – max range
        int max_len = 0,p = 0;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String[] s = new String[k];
        s = br.readLine().split(" ");              //Reading input
        int[] nos = new int[k];
        int[] res = new int[k];
        int[] bucket = new int[10];
        int digit;
        for (int i = 0; i < k; i++) {                  //Stroing input in nos arrray
            nos[i] = Integer.parseInt(s[i]);
        }
        for (int i = 0; i < 10; i++) {                 //Initializing bucket values to 0 for redix sort
            bucket[i] = 0;
        }
        for (int i = 0; i < k; i++) {        //Max no given by user
            if(nos[i]>p){
```

```
        p = nos[i];
    }
}
max_len = (int) Math.log(max);
max_len++;

        //redix sort from lsb to msb
for (int i = 1; p/i>0; i = i*10) {
    //count sort to sort individual position
    for (int j = 0; j < k; j++) {
        digit = (nos[j]/i)%10;
        bucket[digit]++;
    }
    for (int j = 1; j < k; j++) {
        bucket[j] = bucket[j]+bucket[j-1];

    }
    for (int j = k-1; j >= 0; j--) {
        digit = (nos[j]/i)%10;
        int l = bucket[digit] - 1;
        res[l] = nos[j];

    }
    for (int j = 0; j < k; j++) {
        nos[j] = res[j];
    }

}
//nos[k-1]=p;
for (int i = 0; i < k; i++) {
    System.out.print(nos[i]+" ");        //Output of radix sorting
```

```
        }
    }


}
```

```
package com.company;
                    //Problem 3 Assignment 2
//Time complexity – O(n^2)      Space complexity – O(n)
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;

public class Main {

  public static void main(String[] args) throws IOException {
        // write your code here
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String[] s = new String[n];
    int count = 0;
    int[] map = new int[n];    //map for array map to find coverage of sorted data
    Data[] data = new Data[n];
    Data[] sorted = new Data[n];
    int[] arr = new int[n];
    s = br.readLine().split(" ");      //input in arr
    for (int i = 0; i < n; i++) {
      arr[i] = Integer.parseInt(s[i]);
    }
    for (int i = 0; i < n; i++) {      //all ranges of helper stored in Data class object array
      int srange = i - arr[i];
      int erange = i + arr[i];
      int totalrange = 2*arr[i] +1;
      if(arr[i]==-1){        //if value is -1,set ranges -1 and index as i
        srange=-1;
        erange=-1;

        totalrange=-1;

        data[i] = new Data(srange,erange,totalrange,i);


        continue;

    }else {                    //making sure starting range index do not get negative

        while (srange < 0) {

            totalrange--;

            srange++;

        }

        while (erange >= n) {

            totalrange--;

            erange--;

        }

                    //Storing helpers start index and end index where it can provide help in Data class
                    //object  anlong with total range it provides help

        data[i] = new Data(srange, erange, totalrange, i);

        //System.out.println(data[i].index);

    }

}

//sort according to ranges max to min- insertion sort used here, hence O(n^2) time


for (int i = 0; i < n; i++) {

    Data k =new Data(data[i].startrange,data[i].endrange, data[i].range,data[i].index);

     //k = data[i].range;

    int j =i-1;

    while (j>=0&&data[j].range>k.range){

        data[j+1] = data[j];

        j--;

    }

    data[j+1]= k;
```

```
        }
        for (int i = 0; i < n; i++) {
           //System.out.println(data[i].index);
        }
        //Sorting in desc
//        for (int i = n-1; i >= 0 ; i--) {
//           //System.out.println(data[i].range);
//           if(data[i].range ==-1){
//              sorted[n-(i+1)] = new Data(-1,-1,-1,i);
//           }else {
//              sorted[n-(i+1)] = new Data(data[i].startrange,data[i].endrange,data[i].range,data[i].index);
//           }
//        }
        for (int i = 0; i < n; i++) {        //initializing map array
           //System.out.println(sorted[i].index);
           map[i] = -1;
        }

        for (int i = n-1; i >= 0; i--) {
           if(data[i].range==-1)break;
           for (int j = data[i].startrange; j <= data[i].endrange ; j++) {
              if(map[j]==1)continue;
              else {
                 map[j]=1;                  //indices set to 1 from -1 where helper is covering indices
                 count++;
                 data[i].flag=1;            //Putting flag as 1 to helpers that contribute to final answer
              }

//              if(count==n){
//                 System.out.print(i);
//                 return;
//              }
           }


           }
        count = 0;
        for (int i = 0; i < n; i++) {
           if (data[i].flag==1) count++;
           if(map[i]==-1){
              count = -1;
              break;
           }
        }
        System.out.println(count);
        }

     }



class Data{        //To store info on all rangers on a helper
     public int startrange;              //starting index of helpers help range
     public int endrange;                //end index of helpers help range
     public int range;                   //total range of helper where it can serve
     public int index;                   //to store original index-used in keeping track
     public int flag;
                                         //Constructors
     Data(int srange, int erange ,int trange,int indx){

        flag=0;
        startrange = srange;
        endrange = erange;
```

```
        range = trange;

        index=indx;

    }

    Data(){

        startrange=0;

        endrange=0;

        range=0;

        index=0;

        flag=0;

    }


}
```

Approach – The input data is stored in class Data which stores data on each helper such as its starting index where it can provide help, ending index where it can provide help and the total range calculated from the index it is present and the value of index.

After storing all data, the array is sorted using insertion sort which I implemented in O(n^2).

After sorting, the data (helpers) is chosen in descending order of their range to serve and flagged if and only if they cover an index not covered by any other helper before. An extra array of size n is used to keep track to indices.

```
package com.company;
                        //Problem 4 Assignment 2
//Time complexity – O(n^2)        Space complexity – O(n)
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Scanner;
import java.util.Stack;

public class Main {

    public static void main(String[] args) throws IOException {

        Scanner scan = new Scanner(System.in);
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String[] p = br.readLine().split(" ");
        String[] s = new String[p.length];
                        //Taking input
        for (int i = 0; i < p.length; i++) {
            if(p[i].equals("N")||p[i].equals("n")){        //for null cases
                s[i]="-1";
            }
            else {
                s[i]=p[i];            //all data in s
            }
            //System.out.print(s[i]);
        }

        Tree tree = new Tree(s);       //Tree constructed by passing s

        int informant = scan.nextInt();
```

```
                    //Doing dfs from index informant in the data structure
        tree.dodfs(informant);
        //dfs on node informant and tree and count max size of queue-- answer


    }
}


class Node{
    public int data;
    public Node link1;
    public Node link2;
}


class Tree {
    static int max = 0;
    static int count = 0;
    public Node[] nodes;
    Stack<Integer> stack = new Stack<Integer>();
                    //constructor
    public Tree(String[] s) {
        nodes = new Node[s.length];
        for (int i = 0; i < s.length; i++) {
            nodes[i] = new Node();              //Initialising nodes with all data from s
            nodes[i].data = Integer.parseInt(s[i]);
            nodes[i].link1 = null;
            nodes[i].link2 = null;
            for (int j = 0; j < i; j++) {
                if (nodes[j].data == -1 && j == 0) continue;        //continue when ip “N” encounters
                else {              //Linking nodes to one another
                    if (nodes[j].link1 == null && nodes[j].data != -1) {
                        nodes[j].link1 = nodes[i];
                        //System.out.println("Parent of"+nodes[i].data+" is "+nodes[j].data+"-Left child");
                        break;
                    } else {
                        if (nodes[j].link2 == null && nodes[j].data != -1) {
                            nodes[j].link2 = nodes[i];
                            //System.out.println("Parent of"+nodes[i].data+" is "+nodes[j].data+"-right child");
                            break;
                        }
                    }
                }
            }
        }
    }

    public void dodfs(int informant) {
                //dfs algorithm but only calculating count of deepest branch

        Boolean[] visited = new Boolean[nodes.length];
        for (int i = 0; i < nodes.length; i++) {
            if (nodes[i].data == informant) {
                //stack[top++] = nodes[i];
                //if(top+1>max)max = top+1;
                dfsperform(i, visited);
            }
        }



    }

    public void dfsperform(int nodeindex, Boolean[] visited) {
        visited[nodeindex] = true;
```

```
        count++;

        stack.push(nodeindex);

        if (count > max) max = count;


        while (!stack.isEmpty()) {

          int n = stack.pop();

          if (visited[n] == false) {

            dfsperform(n, visited);

            count++;

          }

        }

        System.out.println(max);

    }


}
```

Approach – All the data is stored in individual nodes with data field as value of nodes and all links link1 and link2 as null initially, After creating all nodes with input values given by user, tree is constructed. Tree is constructed from element by element filling links of each node from first to last and filling link1 then link2 to maintain order of tree. Next the informant variable is checked against all data values of nodes[] to find the node for dfs. dfs is run on node informant and calculated max depth as it will be the max number of hours we have to wait. Other nodes will end data parallely to it