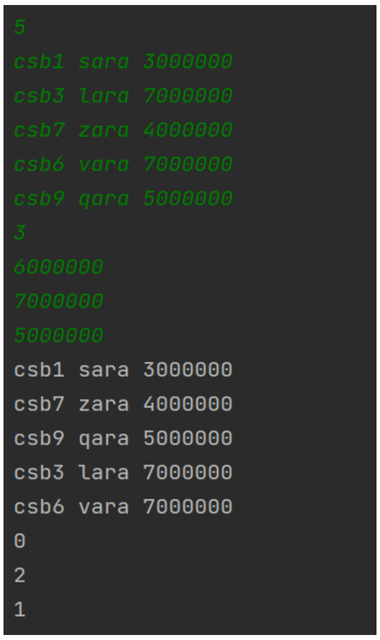
1. **Placement Sort:**
   Time Complexity: O(nlogn) for sorting
   O(n) Searching. Even though I am applying binary search, if the number of similar element is order of n, the sear will run for O(n) time in worst case and O(logn) time in best case.
   Overall: O(nlogn).

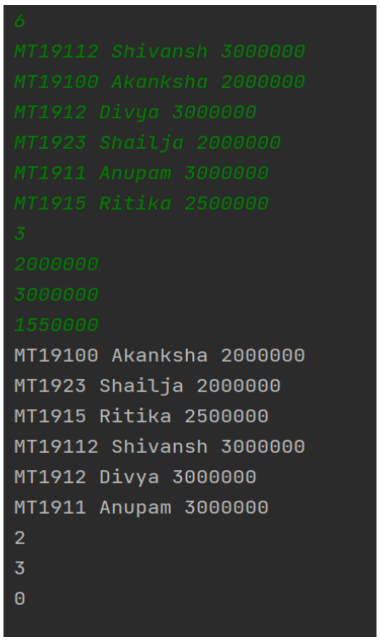
```java
package placementsort;

import java.util.Scanner;

class student{
    String rn;
    String name;
    int p;
}
public class placementSort {

    //search
    //since this search function will get a sorted array
    //as the array is being sorted as part 1 of this question
    //we can implement binary search to search in this array

    public static int _search(int key, student[] array, int l, int r){
        int count = 0;
        int c = (l+r)/2;
        if (array[c].p==key) {
            //if the middle element is the key,
            //all other elements equal to it must be present in it's
vicinity only
            // look left, look right, count
            count++;
            for(int x =c-1; x>-1; --x) {

                if (array[x].p == key) {
                    count++;
                }
                else break;
            }
            for(int x =c+1; x<array.length; x++) {
                if (array[x].p == key) {
                    count++;
                }
                else break;
            }
            return count;

        }
        //if one or less element remains and then also it's not a match,
        //the key is not present
        else  if(r-l<=1) return count;
        else{
            if(array[c].p>key) {
                count = _search(key, array, l, c);
            }
            else if(array[c].p<key) {
                count = _search(key, array, c, r);
            }
            return count;
        }

    }


    //implementing a mergesort for the sorting part
    //selected to implement merge sort because it is stable
    //which is a requirement in the question
    public static void merge(student[] array,int l, int r, int c){

        if(array[c-1].p>array[c].p){

            int size = r-l;
            int i=l,j=c,ti=0;
            student[] aux = new student[size];
            for(int s = 0; s< size; s++) aux[s] = new student();
            while(i<c && j<r){
                if(array[i].p<=array[j].p)aux[ti++]=array[i++];
                else aux[ti++]=array[j++];
            }
            System.arraycopy(array,i,array,l+ti,c-i);
            System.arraycopy(aux,0,array,l,ti);

        }
        else {
            return;
        }

    }
    //implementing mergesort as the stablesort algorithm
    public static void stablesort(student[] array, int l, int r){
        if(r-l<=1)return;
        int centre = (l+r)/2;
        stablesort(array,l,centre);
        stablesort(array,centre,r);
        merge(array,l,r,centre);

    }

    public static void main(String[] args){

        Scanner s = new Scanner(System.in);
        int n = s.nextInt();

        student[] array = new student[n];

//      getting input and constructing the student array
        for(int i=0;i<n;i++){
            student x = new student();
            x.rn = s.next();
            x.name = s.next();
            x.p = s.nextInt();
            array[i]=x;
        }


//      sorting the student array
        stablesort(array,0,n);
```

```
//      getting number of test cases to search
        int k = s.nextInt();
        int[] res = new int[k];
        //getting search test cases and passing them to the searching 
algorithm
        //result is stored in the res[] array
        for(int i=0;i<k;i++){
            int key = s.nextInt();
            res[i]=_search(key,array,0,n);
        }
//        printing the outputs
        for(int i =0;i<n;i++)
            System.out.printf("%s %s 
%d%n",array[i].rn,array[i].name,array[i].p);
        for(int i=0;i<k;i++){
            System.out.println(res[i]);
        }



        }
    }
```

Output:



```
5
csb1 sara 3000000
csb3 lara 7000000
csb7 zara 4000000
csb6 vara 7000000
csb9 qara 5000000
3
6000000
7000000
5000000
csb1 sara 3000000
csb7 zara 4000000
csb9 qara 5000000
csb3 lara 7000000
csb6 vara 7000000
0
2
1
```



```
6
MT19112 Shivansh 3000000
MT19100 Akanksha 2000000
MT1912 Divya 3000000
MT1923 Shailja 2000000
MT1911 Anupam 3000000
MT1915 Ritika 2500000
3
2000000
3000000
1550000
MT19100 Akanksha 2000000
MT1923 Shailja 2000000
MT1915 Ritika 2500000
MT19112 Shivansh 3000000
MT1912 Divya 3000000
MT1911 Anupam 3000000
2
3
0
```

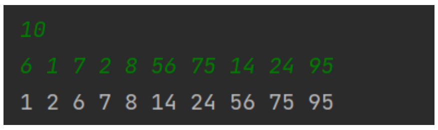
2. Deadline Sort:

Time Complexity: O(n)

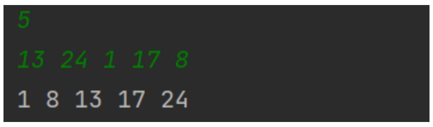
```
package deadlinesort;
import java.lang.Math;
```

```java
import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;
import java.util.Scanner;

public class deadlineSort {
    public static int digitcount(int n){
        n = Math.abs(n);
        int digit = 0;
        if(n/10>0){
            return (1+digitcount(n/10));
        }
        else return 1;
    }

    //This function implements the radix sort algorithm
    public static void dsort(int[] array, int n, int d){

        //auxiliary space complexity = 10n = O(n)
        ArrayList<ArrayDeque<Integer>> buckets = new ArrayList<>(10);
        for(int i = 0; i<10;i++){
                buckets.add(new ArrayDeque<Integer>(n));
            }
        //total number of passes = maximum digit
        for(int i = 0; i<d; i++){
            //inner loop complexity = O(n)
            int extractor = (int)Math.pow(10,i);
            for(int j = 0; j < n; j++){
                //extract digit at
                int x = array[j]/extractor;
                //to the bucket no x%10
                buckets.get(x%10).addLast(array[j]);
            }

            int a = 0; //variable to loop through array

            //popping buckets and adding to array
            //from all 10 buckets from 0 to 9
            //this part is also 10n times = O(n) complexity
            for(int q = 0; q<10; q++){
                //empty bucket q
                while(!buckets.get(q).isEmpty())
                    array[a++]=buckets.get(q).removeFirst();
            }
        }
        //end of outer for loop
        //complexity is O(dn), where d is the maximum digit count among all
the numbers
        //time complexity is considered for large value of n
        //even if we have 10 digit numbers, complexity is just O(10n)
        //Since d << n, we can consider this algorithm O(n)

    }

    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        int k = s.nextInt();
        int maxdigit = 0;
        int[] deadlines = new int[k];

        for(int i =0; i<k;i++){
            deadlines[i]=s.nextInt();
            maxdigit = Math.max(digitcount(deadlines[i]),maxdigit);
        }

        dsort(deadlines,k,maxdigit);
        for(int i =0; i<k;i++){
            System.out.printf("%d ",deadlines[i]);
        }
    }

}
```

Output:



```
10
6 1 7 2 8 56 75 14 24 95
1 2 6 7 8 14 24 56 75 95
```

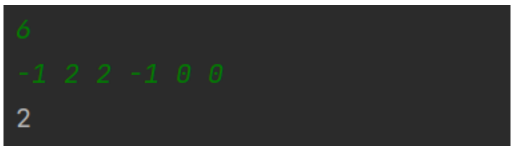


```
5
13 24 1 17 8
1 8 13 17 24
```

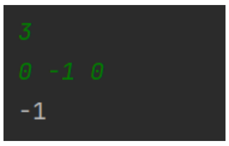
3. **Helping Hands:**
   Time Complexity: O(nlogn)

```java
package helpinghands;
import java.util.*;
class p{
    int x;
    int y;}


class xsort implements Comparator<p>{
    public int compare(p a, p b){
        if(a.x==b.x) return  0;
        else if(a.x>b.x) return 1;
        else return -1;
    }
}

class ysort implements Comparator<p>{
    public int compare(p a, p b){
        if(a.y==b.y) return  0;
        else if(a.y<b.y) return 1;
        else return -1;
    }
}

//0-4,5-5


public class helpingHands{
    public static int helpinHands(ArrayList<p> pairs, int k){

        int minimumHelpers = 1;
        int cub,ii,mub;

        if (pairs.get(0).x != 0) return -1;
        else if(pairs.get(0).y==k) return minimumHelpers;
        else {
            cub = pairs.get(0).y;
        }
//          cub is the current upper bound
        for(int i = 0; i< pairs.size();i++){
            ii = i+1;
            mub=cub;
            // greedily selecting the next best helper
            //who has the highest range
            //but also does leave any uncovered people in the middle
            //if our present helper is providing till kth person,
            //the next helper must start on or before k+1'th person.
            //this step ensures that we cover maximum number of people
            //using least number of helpers.
            while( ii < pairs.size() && pairs.get(ii).x<=(cub+1)){
                if(pairs.get(ii).y>mub)mub = pairs.get(ii).y;
                ii++;
            }
            //once we find the next best helper, we increment the number of
helpers
            ++minimumHelpers;
             if (mub == k) return minimumHelpers;
             else{
                 cub = mub;
                 i = --ii;
             }
        }
        //if at the end of the procedure the upper bound of our coverage by
helpers
        //matches the maximum people (0 to n-1) then we can say that
        //it is possible to feed all the people with our helper's help
        //if not, then it is not possible
        if(cub == k) return minimumHelpers;
        else return -1;

    }

    public  static void main(String[] args){
        ArrayList<p> pairs = new ArrayList<>();
        Scanner s = new Scanner(System.in);
        int t = s.nextInt();
        for(int i =0; i<t;i++){
            int x = s.nextInt();
            if(x!=-1){
                p helper = new p();
//                while entering, we are clipping the
//                range of our helpers to minimum of 0
//                and maximum of n-1
                helper.x = Math.max(0,i-x);
                helper.y = Math.min(x+i,t-1);
                pairs.add(helper);
            }
        }
        s.close();
        Collections.sort(pairs,new xsort().thenComparing(new ysort()));
        System.out.println(helpinHands(pairs,t-1));
    }


}
```

Output:



```
6
-1 2 2 -1 0 0
2
```



```
3
0 -1 0
-1
```

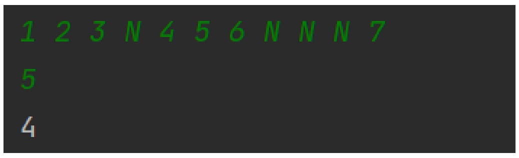
4. Free Ryloth:

Time Complexity: O(n)

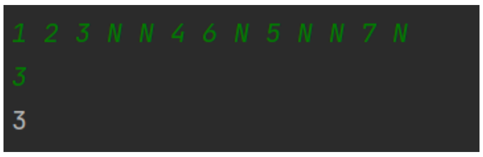
```
package freeryloth;

import java.util.ArrayDeque;
import java.util.ArrayList;
import java.util.Deque;
import java.util.Scanner;

public class freeRyloth {
    public static void makeTree(String input, String nodes,
ArrayList<ArrayList<Character>> tree ){

        for (int i = 0; i < input.length(); i++){
            int idx;
            char x = input.charAt(i);
            char rchild,lchild;
            //if ith node in the input array is not 'N'
            if(Character.isDigit(x)){
                //get its numeric value
                int xi = Character.getNumericValue(x);
               idx = nodes.indexOf(x);
               //if no child, break out of the loop
               if(idx*2+1>=input.length()) break;
               lchild = input.charAt(idx*2+1);
               rchild = input.charAt(idx*2+2);
               //if its left child is also a int
                //add this link to the lists of both
                //the current node and its child
               if(Character.isDigit((lchild))){
                   //list for the current node
                   tree.get(xi).add(lchild);
                   //list for the left child node
                   tree.get(Character.getNumericValue((lchild))).add(x);
               }
                //if its right child is also an int
                // add this link to the lists of both
                //the current node and its child
               if(Character.isDigit((rchild))){
                   //list for the current node
                   tree.get(xi).add(rchild);
                   //list for the left child node
                   tree.get(Character.getNumericValue((rchild))).add(x);
```

```java
                }

            }
        }
    }
    public static int spreadInfo(ArrayList<ArrayList<Character>> tree, char
houseno, String nodes, int m){
        int[] levels = new int[m+1];
        for(int i = 0; i<m+1; i++){
            levels[i]=-1;
        }
        int k = Character.getNumericValue(houseno);
        int hours = 0;
        levels[k]=0;
        Deque<Character> X = new ArrayDeque<>();
        X.addLast(houseno);
        //outer loop runs for all the unvisited nodes, therefore max n
times
        while(true){
            //since it's a binary tree, this inner loop will run maximum 3
times.
            for(int i = 0; i <tree.get(k).size();i++){
                int x =  Character.getNumericValue(tree.get(k).get(i));
                if(levels[x]==-1) { //System.out.printf("k here is %d
%n",k);
                    levels[x] = levels[k]+1;
                X.addLast(tree.get(k).get(i));
                hours = Math.max(hours,levels[x]);}
            }
            if(!X.isEmpty()) {
                k = Character.getNumericValue(X.removeFirst());
            }
            else break;
        }
        //runs at maximum 3*n. Therefore it is O(n).

    return hours;
    }

    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        String input = s.nextLine();
        char firstHouse = s.next().charAt(0);
        s.close();
        //preprocess theinpt to construct the tree
        input = input.replaceAll(" ","");
        String nodes = input.replaceAll("N","");
        int maxnode = 0;
        for(int i = 0; i<nodes.length();i++)
            maxnode =
Math.max(maxnode,Character.getNumericValue(nodes.charAt(i)));
        //initialising the tree
        ArrayList<ArrayList<Character>> tree = new ArrayList<>(maxnode+1);
        for(int i = 0; i< maxnode+1; i++) tree.add(new ArrayList<>());
        //constructing the tree
        makeTree(input,nodes,tree);
        //traversing
        System.out.println(spreadInfo(tree,firstHouse,nodes,maxnode));
        }
}
```

Output:



```
1 2 3 N 4 5 6 N N N 7
5
4
```



```
1 2 3 N N 4 6 N 5 N N 7 N
3
3
```