某比赛已经进入了淘汰赛阶段,已知共有 n 名选手参与了此阶段比赛，他们的得分分别是 a_1,a_2….a_n,小美作为比赛的裁判希望设定一个分数线 m，使得所有分数大于 m 的选手晋级，其他人淘汰。
但是为了保护粉丝脆弱的心脏，小美希望晋级和淘汰的人数均在[x,y]之间。
显然这个 m 有可能是不存在的，也有可能存在多个 m，如果不存在，请你输出 -1，如果存在多个，请你输出符合条件的最低的分数线。

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.Arrays;

public class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        String[] params = br.readLine().trim().split(" ");
        int n = Integer.parseInt(params[0]);
        int x = Integer.parseInt(params[1]);
        int y = Integer.parseInt(params[2]);
        String[] strArr = br.readLine().trim().split(" ");
        int[] scores = new int[n];
        for(int i = 0; i < n; i++) scores[i] = Integer.parseInt(strArr[i]);
        Arrays.sort(scores);
        // 每位选手要么就晋级，要么就淘汰，是互斥的
        int m = -1;
        int i = 0;
        int count = n - 1;
        for(i = 0; i < n; i++){
            m = scores[i];
            if(count >= x && count <= y && n - count >= x && n - count <= y){
                // 检查一下是不是真的能晋级 count 人
                if(judge(scores, i, m, count)) {
                    break;
                }else
                    m = -1;
            }else{
                m = -1;
                count --;
            }
        }
        System.out.println(m);
    }

    private static boolean judge(int[] scores, int start, int m, int target) {
        start ++;
```

```
        if(start == scores.length) return false;
        while(start < scores.length){
            if(scores[start] == m) return false;
            start ++;
        }
        return true;
    }
}
```

小美和小团所在公司的食堂有 N 张餐桌，从左到右摆成一排，每张餐桌有 2 张餐椅供至多 2 人用餐，公司职员排队进入食堂用餐。小美发现职员用餐的一个规律并告诉小团：当男职员进入食堂时，他会优先选择已经坐有 1 人的餐桌用餐，只有当每张餐桌要么空着要么坐满 2 人时，他才会考虑空着的餐桌；
当女职员进入食堂时，她会优先选择未坐人的餐桌用餐，只有当每张餐桌都坐有至少 1 人时，她才会考虑已经坐有 1 人的餐桌；
无论男女，当有多张餐桌供职员选择时，他会选择最靠左的餐桌用餐。现在食堂内已有若干人在用餐，另外 M 个人正排队进入食堂，小团会根据小美告诉他的规律预测排队的每个人分别会坐哪张餐桌。

```
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws IOException{
        BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
        BufferedWriter writer = new BufferedWriter(new
OutputStreamWriter(System.out));
        int T = Integer.parseInt(reader.readLine());
        for (int i = 0; i < T; i++) {
            int N = Integer.parseInt(reader.readLine());
            String tables = reader.readLine();
            int M = Integer.parseInt(reader.readLine());
            String enters = reader.readLine();

            int[] res = solve(tables, enters);
            for (int r : res) {
                writer.write(Integer.toString(r));
                writer.newLine();
            }
        }
        writer.flush();
    }

    private static int[] solve(String tables, String enters) {
```

```
        List<PriorityQueue<Integer>> pqs = new ArrayList<>(3);
        pqs.add(new PriorityQueue<>());
        pqs.add(new PriorityQueue<>());
        pqs.add(new PriorityQueue<>());
        for (int i = 0; i < tables.length(); i++) {
            pqs.get(tables.charAt(i) - '0').add(i);
        }
        int[] res = new int[enters.length()];
        for (int i = 0; i < enters.length(); i++) {
            int table;
            if (enters.charAt(i) == 'M') {
                if (pqs.get(1).isEmpty()) {
                    table = pqs.get(0).poll();
                    pqs.get(1).add(table);
                } else {
                    table = pqs.get(1).poll();
                    pqs.get(2).add(table);
                }
            } else {
                if (!pqs.get(0).isEmpty()) {
                    table = pqs.get(0).poll();
                    pqs.get(1).add(table);
                } else {
                    table = pqs.get(1).poll();
                    pqs.get(2).add(table);
                }
            }
            res[i] = table + 1;
        }

        return res;
    }

}
```