

小团的蛋糕铺长期霸占着美团 APP 中“蛋糕奶茶”栏目的首位，因此总会吸引各路食客前来探店。

小团一天最多可以烤  $n$  个蛋糕，每个蛋糕有一个正整数的重量。

早上，糕点铺已经做好了  $m$  个蛋糕。

现在，有一个顾客要来买两个蛋糕，他希望买这一天糕点铺烤好的最重的和最轻的蛋糕，并且希望这两个蛋糕的重量恰好为  $a$  和  $b$ 。剩余的  $n-m$  个蛋糕可以现烤，请问小团能否满足他的要求？

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

public class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in))
);
        String line;
        String[] params;
        while((line = br.readLine()) != null) {
            params = line.trim().split(" ");
            int n = Integer.parseInt(params[0]);
            int m = Integer.parseInt(params[1]);
            int a = Integer.parseInt(params[2]);
            int b = Integer.parseInt(params[3]);
            params = br.readLine().trim().split(" ");
            int[] weight = new int[m];
            HashSet<Integer> set = new HashSet<>(); // 保存现有蛋糕的重量
            for(int i = 0; i < m; i++) {
                weight[i] = Integer.parseInt(params[i]);
                set.add(weight[i]);
            }
            Arrays.sort(weight);
            // 保证 a<b
            if(a > b){
                int temp = a;
                a = b;
                b = temp;
            }
            if(weight[0] < a || weight[m - 1] > b){
                // 现有蛋糕中，重量最小的小于 a，最大的大于 b，肯定完成不了需求
                System.out.println("NO");
            }else{
                if(set.contains(a) && set.contains(b)) // 如果现有蛋糕中已经包
```

含了 a 和 b, 就没问题

```
        System.out.println("YES");
    }else{
        if(set.contains(a) || set.contains(b)){
            // 如果只包含 a 或 b, 检查一下 n-m 是否大于等于 1, 即还有一个重量需要现烤
            System.out.println(n - m >= 1 && weight[m - 1] <= b? "YES": "NO");
        }else{
            // 否则需要检查 n-m 是否大于等于 2, 即两个重量都需要现烤
            System.out.println(n - m >= 2? "YES": "NO");
        }
    }
}
}
```

小团是某综艺节目的策划, 他为某个游戏环节设计了一种晋级规则, 已知在这个游戏环节中每个人最后都会得到一个分数 `score_i`, 显而易见的是, 游戏很有可能出现同分的情况, 小团计划该环节晋级人数为 `x` 人, 则将所有人的分数从高到低排序, 所有分数大于等于第 `x` 个人的分数且得分不为 0 的人都可以晋级。

请你求出本环节的实际晋级人数。显然这个数字可能是 0, 如果所有人的得分都是 0, 则没有人满足晋级条件。

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

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]);
        params = br.readLine().trim().split(" ");
        int[] scores = new int[n];
        for(int i = 0; i < n; i++)
            scores[i] = Integer.parseInt(params[i]);
        Arrays.sort(scores);
        int count = x;
        int baseline = scores[n - x];
        if(scores[n - 1] == 0){
```

```

        count = 0;
    }else{
        if(baseline > 0){
            count --;
            while(scores[n - x] == baseline){
                count ++;
                x ++;
            }
        }else{
            while(scores[n - x] == baseline){
                count --;
                x --;
            }
        }
    }
    System.out.println(count);
}
}

```

小美请小团吃回转寿司。转盘上有  $N$  盘寿司围成一圈，第 1 盘与第 2 盘相邻，第 2 盘与第 3 盘相邻，...，第  $N-1$  盘与第  $N$  盘相邻，第  $N$  盘与第 1 盘相邻。小团认为第  $i$  盘寿司的美味值为  $A[i]$ （可能是负值，如果小团讨厌这盘寿司）。现在，小团要在转盘上选出连续的若干盘寿司，使得这些寿司的美味值之和最大（允许不选任何寿司，此时美味值总和为 0）。

```

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;

```

```

public class Main {
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        int T = Integer.parseInt(br.readLine().trim());
        while(T-- > 0){
            int n = Integer.parseInt(br.readLine().trim());
            String[] strArr = br.readLine().trim().split(" ");
            int[] yummy = new int[n];
            int sum = 0;
            for(int i = 0; i < n; i++){
                yummy[i] = Integer.parseInt(strArr[i]);
                sum += yummy[i];
            }
            // 为了降低时间复杂度，可以两种情况一起求
            int max = yummy[0];
            int min = yummy[0];
            int dpMax = yummy[0];
            int dpMin = yummy[0];

```

```
    for(int i = 1; i < n; i++){
        dpMax = Math.max(dpMax + yummy[i], yummy[i]);
        max = Math.max(max, dpMax);
        dpMin = Math.min(dpMin + yummy[i], yummy[i]);
        min = Math.min(min, dpMin);
    }
    System.out.println(Math.max(sum - min, max));
}
}
```