L10: Ad Hoc Problem Practice Questions

a-Live: https://codeforces.com/problemset/problem/1037/C

1-Tut : Rectangular Area

Send Feedback

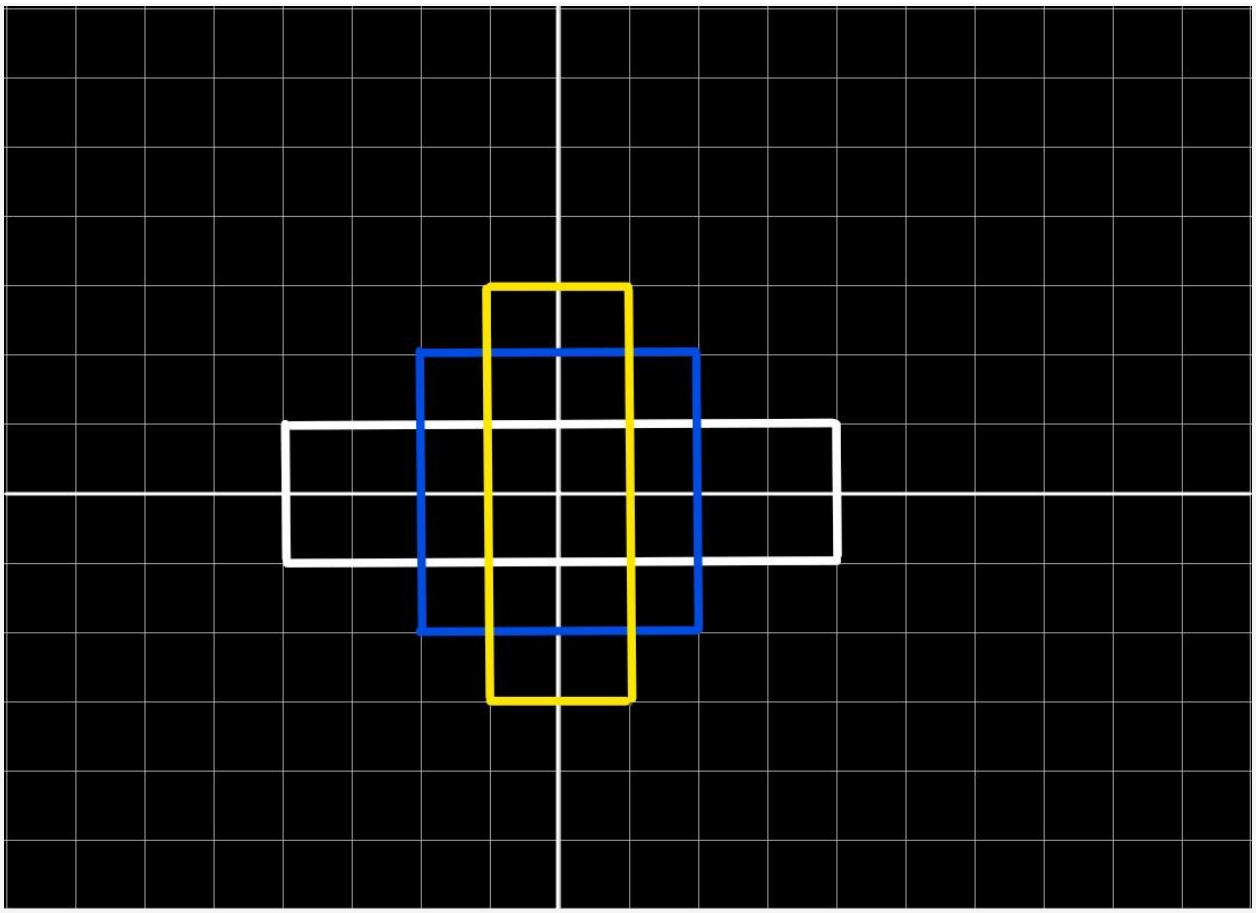
You are given N rectangles, which are centered in the center of the Cartesian coordinate system and their sides are parallel to the coordinate axes. Each rectangle is uniquely identified with its width (along the x-axis) and height (along the y-axis). Navdeep has coloured each rectangle in a certain colour and now wants to know the area of the coloured part of the paper. Please refer to the sample test case 1 and image used in it for better understanding.

Sample Input 1 :

3
8 2 (represented by white coloured rectangle)
4 4 (represented by blue coloured rectangle)
2 6 (represented by yellow coloured rectangle)

Sample Output 1 :

28



In other words, he wants to know the number of unit squares that belong to at least one rectangle. This can also be seen in the image. There are 28 unit squares, which come in the bounds of at least one rectangle.

Input Format :

The first line of input contains the integer N, that denotes the number of rectangles.
Each of the following N lines contains even integers X and Y, dimensions (width and height, respectively) of the corresponding rectangles.

Output Format :

The first and only line of output must contain the required area, as described in the task.

Constraints:

1 <= N <= 10^5
1 <= X,Y <= 10^5
All values of X and Y are even
Time Limit: 1 second

Sample Input 1 :

3
8 2
4 4
2 6

Sample Output 1 :

28

Sample Input 2 :

5
2 10
4 4
2 2
8 8
6 6

Sample Output 2 :

68

```
#include<bits/stdc++.h>
using namespace std;
int main(){

    // write your code here
    int n; cin >> n;
    int *height = new int[1000000]();
    int max_x = 0;
    int x,y;
    for(int i = 0; i < n; i++){
        cin >> x >> y;
        if(height[x/2] < y){
            height[x/2] = y;
        }
```

```
        if(x/2 > max_x){
            max_x = x/2;
        }

    }
    long area = 0;
    for(int i = max_x; i > 0; i--){
        if(height[i] < height[i+1]){
            height[i] = height[i+1];
        }
        area += height[i];
    }

    cout << 2*area << endl;
    return 0;
}
```

2-Ass : Light Up the Bulbs

Send Feedback

Navdeep is given a binary string (string consists of only 0's and 1's) of size 'n'. The string represents the state of n bulbs. '0' represents that the bulb is in 'OFF' mode and '1' represents the bulb is in 'ON' mode. Navdeep is given the task to change all the characters to 1. In other words, Navdeep has to light up all the bulbs.

He can perform two operations on the given string:

1. In the first operation, he can choose any substring and reverse it. For example, in the binary string: 0110001, if we reverse substring from 1st to 3rd index in the given string, then the resultant string will be: 0011001. Here, 110 is reversed to form 011. This operation will cost Rs. 'X'.
2. In the second operation, he can choose any substring and toggle each character of that substring. For example, in the binary string: 0110001, if we toggle characters of the substring from 1st to 3rd index in the given string, then the resultant string will be: 0001001. Here, 110 is converted to form 001. This operation will cost Rs. 'Y'.

Can you help Navdeep to complete the task in the minimum amount possible.

Input Format:

The first line will contain three space separated integers: 'n', 'X', 'Y', denoting the number of bulbs, cost of first operation and cost of second operation respectively.

The second line contains a binary string of length 'n', consisting of 0's and 1's, representing whether the bulb is 'OFF' or 'ON'.

Output Format:

Print a single integer denoting the minimum cost required to light up all the bulbs.

Constraints:

1 <= n <= 3,00,000

0 <= X, Y <= 10^9

Time Limit: 1 second

Sample Input:

5 1 10
01000

Sample Output:

11

Explanation:

First, Reverse substring (0, 1): “01000” -> “10000”, COST = 1
Second, Invert substring (1, 4): “10000” -> “11111”, COST = 10

Total cost = 1+10 => 11

```
#include<iostream>
#include<string>
#include<algorithm>
using namespace std;
int main()
{
        long long n, x, y;
        cin >> n >> x >> y;
        string s;
        cin >> s;
        long long count = 0;
        if(s[0]=='0')
        {
                count = 1;
        }
        for(long long i=1; i<n; i++)
        {
                if((s[i-1]=='1') && (s[i]=='0'))
                {
                        count += 1;
                }
        }

        if(count==0)
        {
                cout << 0;
        }
        else
        {
                cout << (count-1) * min(x, y) + y;
        }
}
```

3-Ass : Interesting Sequences

Send Feedback

Professor Jain has a class full of notorious students. To get anything done from them is a herculean task. Prof Jain wanted to organize a test. He gave this responsibility to Aahad. Aahad did an excellent job of organizing the test. As a reward, the professor gave him an sequence of numbers to play with. But Aahad likes playing with "interesting" sequence of numbers, which are sequences that have equal elements.

Now, the problem is - Prof Jain has a sequence of numbers, but that sequence isn't always "interesting".

To ensure sequence has equal elements, Prof Jain has 2 options:

1) Choose two elements of sequence. Decrease the first element by 1 and increase the second element by 1. This operation costs 'k' coins.

2) Choose one element of array and increase it by 1. This operation costs 'l' coins.

You have to find the minimum number of coins that Prof Jain needs to turn his sequence into a “interesting" sequence for Aahad?

Input Format

The first line of input contains three space-separated integers: n, k, l. Integer n is the size of array. Integer k is the number of coins needed to perform the first operation. Integer l is the number of coins needed to perform the second operation.

The second line of input contains n space separated integers: (a1, a2, a3... an), that denote the numbers of the sequence.

Constraints:

1 <= n <= 1000

1 <= k, l <= 10^9

1 <= a[i] <= 1000

Time Limit: 1 second

Output Format

The first and only line of output prints an integer, that denotes minimum number of coins required to make "interesting" sequence.

Sample Input 1:

4 1 2

3 4 2 2

Sample Output 1:

3

Explanation Output 1 :

The professor has a sequence with 4 elements. To perform the first operation, he must pay 1 coin and to perform the second operation, he must pay 2 coins. The optimal strategy is:

-Perform the second operation on the fourth element. Now the sequence is {3, 4, 2, 3}. This costs 2 coins.

-Perform the first operation on the second and third element. The sequence is now "interesting", and it looks like {3, 3, 3, 3}. This costs 1 coin.

The total amount of coins needed is 2 + 1 = 3.

Sample Input 2:

3 2 1

5 5 5

Sample Output 2:

0

Explanation Output 2 :

The given sequence is already "interesting". The professor would spend 0 coins.

Sample Input 3:

5 2 1
1 2 3 4 5

Sample Output 3:

6

Explanation Output 3 :

The professor has a sequence with 5 elements. To perform the first operation, he must pay 2 coins and to perform the second operation, he must pay 1 coin. The optimal strategy is:
-Perform the first operation on the first and last element. Now the sequence is {2, 2, 3, 4, 4}. This costs 2 coins.
-Perform the first operation again on the first and last element. Now the sequence is {3, 2, 3, 4, 3}. This costs 2 coins.
-Perform the first operation on the second and second last element. Now the sequence is {3, 3, 3, 3, 3}. This costs 2 coins.

The total amount of coins needed is 2 + 2 + 2 = 6.

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

int minCost(const vector<int>& a, const int& k, const int& l, int curr) {
	int inc = 0, dec = 0;
	for(int x : a) {
		if(curr > x) {
			inc += (curr - x);
		}
		else {
			dec += (x - curr);
		}
	}
	if(inc >= dec) {
		return (dec * k) + ((inc - dec) * l);
	}
	return (int) 1e15;
}

int32_t main()
{
	ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

	int n; cin >> n;
	int k, l; cin >> k >> l;

```

```
29.         vector<int> a(n);
30.         for(int i = 0 ; i < n ; ++i) {
31.                 cin >> a[i];
32.         }
33.
34.         int rangeMax = *max_element(a.begin(), a.end());
35.         int rangeMin = *min_element(a.begin(), a.end());
36.
37.         int ans = (int) 1e15;
38.         for(int i = rangeMin ; i <= rangeMax ; ++i) {
39.                 ans = min(ans, minCost(a, k, l, i));
40.         }
41.
42.         cout << ans << '\n';
43.         return 0;
44. }
```

4-Ass : Winning Strategy

Send Feedback

The college team, along with their coach, is going to the sports fest to play a football match. There are n players in the team, numbered from 1 to n.

Someone gives a paper to the coach. The paper elaborates on the positions and strategies of the opponent team. Based on it, the coach creates a winning strategy. In that strategy, he decides and gives a particular position to every player.

After this, the coach starts swapping two players at a time to make them stand according to new positions decided on paper.

He swaps players by applying following rules:

1. Any player can swap with the player standing next to him.
2. One player can swap with at most two other players.

Given that initially all the players are standing linearly, numbered from 1 to n, you have to tell whether it is possible for the coach to create new positions by swapping within the constraints defined in the task.

Input Format

The first line of input will contain an integer, that denotes the value of the number of test cases. Let us denote the number of test cases by the symbol T.

Each of the following T test cases consists of two lines. The first line of each test case contains an integer n, that denotes the number of players in the team. The following line contains n space separated integers, denoting the specific position of players in winning strategy.

Output Format

For each test case, if it is possible to create winning strategy positions, then print "YES" (without quotes) and in the next line, print the minimum numbers of swaps required to form the winning strategy order, otherwise print "NO"(without quotes) in a new line.

Constraints

1 <= T <= 50

1 =< N <= 10^5
1 <= A[i] <= n
Time Limit: 1 second

Sample Input 1:

1
5
2 1 5 3 4

Sample Output 1:

YES
3

Explanation

In this case, we can achieve winning strategy positions in 3 swaps. Initial state of positions: 1 2 3 4 5
Three moves required to form winning strategy positions:
1 2 3 4 5 -> 1 2 3 5 4 -> 1 2 5 3 4 -> 2 1 5 3 4

Sample Input 2:

1
5
2 5 1 3 4

Sample Output 2:

NO

Explanation:

In the second case, there is no way to form the specific winning strategy positions by swapping within the constraints mentioned in the task.

```
#include <bits/stdc++.h>
using namespace std;
#define int long long
#define double long double

void solve(vector<int> a, int n) {
        for(int i = 0 ; i < n ; ++i) {
                if(abs(a[i]-i-1) > 2) {
                        cout << "NO\n";
                        return;
                }
        }

        int count = 0;
        for(int i = n-1 ; i >= 0 ; --i) {
                if(a[i] == i+1) {
                        continue;
                }

                if(i-1 >= 0 && a[i-1] == i+1) {
                        ++count;
                        swap(a[i], a[i-1]);
                }
```

```
                else if(i-2 >= 0 && a[i-2] == i+1) {
                        count += 2;
                        a[i-2] = a[i-1];
                        a[i-1] = a[i];
                        a[i] = i+1;
                }
                else {
                        cout << "NO\n";
                        return;
                }
        }

        cout << "YES\n" << count << '\n';
}

int32_t main()
{
        ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

        int t; cin >> t;
        while(t--)
        {
                int n; cin >> n;
                vector<int> a(n);
                for(int i = 0 ; i < n ; ++i) {
                        cin >> a[i];
                }

                solve(a, n);
        }
        return 0;
}
```