FINAL EXAM #1
April 14th 2015
solutions

Task ŽARULJE	**Author: Gustav Matula**

Let us fixate the initial position **p** and observe the schedule of turning off the light bulbs.
The most important observation in the task is the following:
if there exist two adjacent light bulbs to the right of **p**, their positions being **A** and **A+1**, their powers being $\mathbf{W_A}$ and $\mathbf{W_{A+1}}$, such that $\mathbf{W_A} < \mathbf{W_{A+1}}$ then the light bulb at **A+1** is going to be turned off in the immediate next step after turning off light bulb **A**. The reason for this is because light bulb **A** was of greater or equal power to the light bulb on the other (left) end when it was being turned off, so then the light bulb **A+1** is going to be greater. Of course, because of symmetry the similar thing holds for the two adjacent light bulbs located to the left of position **p**.
The light bulbs that will be determined uniquely and turned off can be ignored, so we get an array of light bulbs of non-descending power to the left and to the right, i.e.: DDBAAAXACCDD where X denoted the initial position and the remaining letters denote the light bulbs and it holds D > B > A te A < C < D.
Let us now notice that we will first turn off light bulb X, then all the light bulbs of power A (in some order), then all the light bulbs of power B or all the light bulbs of power C, depending on which ones have greater power and so on until light bulbs of power D. Therefore, we only need to make a choice when we're turning off a group of light bulbs of the same strength, as long as they exist on both sides, we can choose from which side we want to turn off the light bulb. If the have **L** of them on the left side and **R** on the right, then we can turn them off in (**L** + **R** choose **L**) ways (in total, we have **L**+**R** operations of turning off, and **L** of them have to be on the left side).
Therefore, the total number of ways is the multiplication of binomial coefficients on all the different powers of the important light bulbs. This solution is of the complexity O(**KN** log **N**) and it was fast enough for the first two subtasks.

For all points, it was required to speed up the described procedure and there are multiple ways of doing that. One of them is going through all the possible initial positions from the left and keep the important light bulbs on the left side using a stack. Each change on the stack should be remembered as change per light bulb power among the light bulbs on the stack. The similar thing is done for the right side, going through the initial positions from the right and remembering the changes per light bulb power on the stack. Now, using a single pass through the initial positions, we can apply the changes in the amount of light bulb power and maintain the multiplication of binomial coefficients. For example, if the amount of light bulb of a certain power on the left changed from L to L', and on the right side it appears R times, then the multiplication of binomial coefficients will first be divided by the old binomial coefficient, with (L+R chose L) and multiplied by the new binomial coefficient (L'+R choose L').

The complexity of this algorithm is O(N), but before the algorithm itself, we need to calculate all the factorials and their inverses by the given modulo. The complexity of that part of the solution is O(N log N).

Task RUKA	**Author: Luka Kalinovčić**

It is easy to see that we can observe the x and y coordinates of the robot arm separately. Therefore, we need to support two operations over an array of integers $x_1, x_2, \ldots, x_N$:

1. change the value of number x_k
2. count how many times the sign of partial sums of the array changed

We will continue to observe the partial sum array $p_1, p_2, \ldots, p_N$. One of the ways in which we could solve the task is to represent pairs of consecutive partial sums with points in the plane (p_k, p_{k+1}) and build a structure that supports the following:

1. translate all the points from the kth onward for vector (d,d)
2. change the kth point
3. count the number of points in the II. and IV. quadrant (those points correspond to the changes of the sign of partial sums)

As it is often the case, it turns out that the structure mentioned above is difficult (or impossible) to implement so that it meets all the requirements of this task, so we need to simplify the problem.

One of the ways we can calculate the number of pairs in the II. and IV. quadrant is to count the number of points that have at least one negative coordinate and subtract from that number the number of points that have both negative coordinates. The points that have at least one negative coordinate are exactly those points that have the **smaller** coordinate negative, and points that have both negative coordinates are those which larger coordinate is negative.

When we had a structure that could do the following:

1. change the kth number
2. increase all the number from the kth onwards for d
3. count the number of negative numbers

we could solve this task using two instances of this structure. Let's call them L and H. L would contain the smaller coordinates of all the points, and H would contain all the larger coordinates of all the points.

When we change the kth vector from x_k to x_k' we need to do the following:

1. remove $\min(p_{k-1}, p_k)$ from L and $\max(p_{k-1}, p_k)$ from H
2. add $\min(p_{k-1}, p_k - x_k + x_k')$ to L and $\max(p_{k-1}, p_k - x_k + x_k')$ to H
3. increase all the numbers from k+1th onwards for $x_k' - x_k$ in structures L and H (we know that it won't come to a swap between the smaller and greater because both numbers are increasing for the same value)

When we receive a query, we calculate the answer as:

the number of negative numbers in L - the number of negative numbers in H

We are left with implementing the abovementioned structure. The fact that makes this easier is that the requirements for vector changes cannot "jump", but there is a cursor.

Let's first solve the simpler variant that doesn't have a cursor and we can only change the first vector - x_1. Then our structure must support the query "add d to **all** the numbers" instead of "add d to the numbers from kth onwards" and "remove/add the number s/from the beginning" instead of "change the kth number".

We are going to remember the following: stack T, sorted set of numbers S and number D that represents how much all the numbers in the set are changed. Therefore, if the set contains the number x, that means that we actually have the value x+D in the structure. We implement the operations in the following way:

1. remove from the beginning - remove the number x from the top of the stack and remove the number x from the set S
2. add to the beginning - if we add the number x, we add x-D to the top of the stack and in the sorted set
3. adding the value d to all the numbers - increase D by d
4. how much negative numbers are there - count the numbers smaller than -D in the set

It is slightly troublesome to maintain the sorted set of numbers and answer the 4th query - it can be solved using a binary search tree or a logarithmic structure with a map or compression.

Now that we know how to solve the variant in which only the first number changes, it is easy to adapt this solution to work with a cursor. We only need to remember the number of times the sign to the left of the cursor changes and store it in a variable P.

When the cursor moves left, we need to insert the pair into the structure and reduce the variable P if there was a sign change on that pair.
When the cursor moves right, we need to remove the pair from the structure and increment the variable P if there was a sign change on that pair.
When we receive a request to change a vector, we change only the pairs to the right of the cursor - the ones that are in the structure.
When we receive a query, we will just add P to the answer for the pairs to the right of the cursor.

The time complexity of this solution is O(N lg N) or O(N lg^2 N), depending on what we used to maintain the sorted set.