Epiphany 10.1
Editorial
ACM NIT Surat

Total Path

- It is easy to see that the total path to reach (x,y) is the same as (x,-y),(-x,-y),(-x,y) due to symmetry.So assume given point is in 1st quadrant by taking x = abs(x),y=abs(y).
- To reach point (x,y) from (0,0) it will require moves equal to manhattan distance.
- For example take x=3,y=4. So to reach (3,4) we have to move 3 times in upward direction(+Y) ,and 4 times in the right direction(+X).
- So now answer will be number of ways to arrange 3U,4R. which is $\frac{7!}{3!4!}$.
- In general, number of ways to reach (x,y) is $\frac{(x+y)!}{x!y!}$.

Defend the Walls

- If you read the question clearly, the question refers to rotating the Matrix clockwise.
- So, for every point in the Matrix (i,j) it is linked to 3 other points, which are (j,n-i) , (n-i,n-j) and (n-j,i) , in the same order.
- As far as K jumps are considered, you come to the same position after every 4 jumps,so the answer you get by K jumps is equal to the answer you get by K%4 jumps
- The answer for each test case can be calculated in O(1).
- Total Time Complexity is O(T) ;where T is the number of Test Cases.

  PS: For those of you who were unable to find the required set of coordinates, observe the following example:
  Let us say our current position is (x,y) where $x<y$ and $x+y<n$, then the next position will be the coordinate obtained after reflecting this point using line $x+y=n-1$ as axis and then reflecting the resulting point about line $x=0$ (Concept is derived from Coordinate Geometry, studied in High School :P)
  So (x,y) -> (n-y,n-x) -> (n-y,x)

  The same goes for coordinates in other quadrants (which other quadrants? Well, when $x>y$ & $x+y<n$, $x>y$ & $x+y>n$, $x<y$ & $x+y>n$)

MAX XOR

- Sort the given array.
- Now for each query find the max index R in the array such that arr[i] ≤ M.
- If no element is there print -1.
- Another possible answer is having an index between L=0 to R.
- Now iterate bits of X from MSB to LSB and do following:
    1. Let's say the current bit index is j.
    2. if jth bit in X is 0 , then xor will be maximum possible if arr[i] has current bit 1.
    3. if jth bit in X is 1, then xor will be maximum if arr[i] has current bit 0.
    4. We will look for the desired bit and change our range L and R.
        - For all the array elements in range [L,R], all the bits till (j+1)index are the same.(try to observe)
        - So all the jth bits in range [L,R] are sorted and we can do binary search and can accordingly change range [L,R].
- overall Time Complexity : O(T * Q * $\log 10^9$ * logN)
- $\log 10^9$ for iterating bits of X.
- logN for binary search.

Maggie in the city

- In this problem you are to construct a connected graph, which contains n vertices and m edges, and if we delete vertex with number v, our graph stops being connected or to report that such a graph doesn't exist. Moreover, each pair of vertices can have no more than one edge connecting them.
- Obviously, a connected graph doesn't exist if the number of edges is less than n-1. It's easy to notice that the maximal possible number of edges reaches when there is a vertex connected to v and doesn't connect to any other vertex, those can form up to complete the graph. So the maximal number of edges is (n-1)*(n-2)/2+1.
- If m is in that range then the required graph always exists. Then you should place one vertex on the one side of v (let it be 1), and other vertices - on the other side. First, you should connect all this vertices to v and then connect them between each other (except 1).

Tree Under Control

- Let's fix a vertex v. This node adds +1 to all the ancestors whose depth $depth[v] - a[v] \le depth[p]$ ($depth[v]$ = the sum of the weights of edges on the path from the root to the vertex v). It's a segment of the ancestors, ending in v, as the depth increases when moving to the leaves.
- It remains to find the first ancestor on the way up, it does not hold for him - so you can make a binary lifting or binary search, if you will be storing the path to the root in dfs.
- With the partial sums you can calculate the answer for each vertex.

Mike’s Array

- Let's solve this problem using a binary search. We need to check whether we can achieve an array when $fun(a)$ will be at most x. Let’s make dp. $dp[i]$ means a minimal number of elements with indices less than i, which we need to change, but we don't change i-th element. Let's iterate the next element j, which we don't change. Then we know that we can change all elements between i and j. It is equivalent to such a condition
    - $|a_j - a_i| \leq (j - i) \cdot x$
- The difference between neighbouring elements can be at most x. The maximal possible difference increases by x exactly $j - i$ times between elements i and j, so this inequality is correct.
