Problem A: Lord of the Curses

Author: Raihat Zaman Neloy
Alternate writers: Sabit Zahin, Tarango Khan, Nafis Sadique
Analysis:
For finding the boss of an Uruk, we will maintain a segment tree of the health capacities so that we can query the boss in Log(N). We will also need another data-structure to maintain multiple disjoint-sets (let's talk about it later). Now let's analyse the problem through every query:

For query 1, we will iterate over the Uruks. When one Uruk is dead, we will make this Uruk's boss its parent in disjoint-set, so that if we land on this Uruk again in the future, we can easily jump to one of the Uruk who is still alive by finding the root parent of disjoint-set. We will also need to set the health capacity of the dead Uruk to 0 in the segment tree.

For query 2, the X'th Uruk's boss is forcefully set to Y. Nothing to worry when the X'th Uruk is alive. We can just set its boss Y. But if X'th Uruk was dead before this query, then it already had a parent in the disjoint-set. So, we need to destroy that edge. Here comes the scenario where we need to handle dynamic trees. So, the other data structure we will use is **Link-Cut Tree**.

For query 3, needs an update in segment tree, reset its previous boss and also do a cut in the Link-Cut Tree if the X'th Uruk was dead earlier.

Query 4 is the hardest query ever. So skipping the analysis of it. Will take a lot of pages to describe.

Off Topic: So, Judging director renamed the problem as "Lord of the curses".

Data-structures needed: Segment Tree, Link Cut Tree
Complexity: Q*Log(N)
Fun-fact 1: This problem's idea was derived from the scenario of the Game **- *Middle Earth Shadow of War***
Fun-fact 2: Problem author has no clue why the description is two pages long.
Fun-fact 3: This problem was initially called Lord of the Rings. But, most of the judges were sure about the fact that- ***definitely the setter will be cursed by the contestants after the contest***. So, we figured it'd make more sense if it was called Lord of the Curses, and we took the liberty to change it without asking the author.

Problem B: I know Recursions!

Author: Md. Imran Bin Azad
Alternate writers: Hasnain Heickal, Shadman Shadab, Md. Shiplu Hawlader
Analysis:
Step 1. Find out the number of recursive function calls needed to compute the *P-bonacci* value for each ***n*** between **[0,N]**. This can be done with an **O(N)** loop (very similar to how Fibonacci series is generated with loop). Note that it won't take long for these values to exceed **K**.
Step 2. Now you know how many calls are required for each *P-bonacci* value. You can use this info to skip through some function calls without performing them.
Also as you have generated the series in **Step 1**, it should not be hard to figure out the **Invalid** case.

Problem C: Colored Development

Author: Sabit Zahin
Alternate writers: Mohammad Ashraful Islam, Raihat Zaman Neloy, Shadman Shadab, Md Mahbubul Hasan
Analysis: Looks like we all missed the simpler solution :)

Suppose there are n colors and we are to color m layers. What's the probability that i'th color will be used? That is: 1 - the probability the i'th color is not used. What's the probability that the first layer is not colored with color i? (n - 1)/n. So what's the probability that none of the layers are colored with color i? [(n - 1)/n]^m. So the i'th color is used by probability: 1 - [(n - 1)/n]^m. For each color, the probability that, that color will be used is the same. So the expected number of colors is: n[1 - [(n-1)/n]]^m.

That was the simpler solution. The complex solution (and hence the time limit of over 10s) involved, FFT, Modular computation of ncr and stirling number.

Problem D: Array Permutation

Author: Mohammad Ashraful Islam
Alternate writers: Rumman Mahmud, Md. Imran Bin Azad , Md. Kazi Nayeem
Analysis:
First, we need to check whether such an array is possible or not!

For each X = 1,2,...,N
There can be several segments claiming X is the minimum value within range (L_{x1}, R_{x1}) , (L_{x2}, R_{x2}), ...

1) We will find the intersection of ranges for each (L_{xmax}, R_{xmin}): It is guaranteed that x must be within the range. The range must be valid, L_{xmax} <=R_{xmin}, otherwise, the answer is 0.
2) All the given segments for x must have a common intersection, and L_{xmax} <=R_{xmin} Otherwise, there is no such permutation.
3) We will find the intersection of ranges for each (L_{xmin}, R_{xmax}): It is guaranteed that x is the smallest number within the range.

If the given constraints are satisfied for each value of X, then we will calculate the number of ways of permutation.

Now, for each X = N,N-1,N-2,... , 1

We have (L_{xmax}, R_{xmin}) and (L_{xmin}, R_{xmax}).
So, solution of sub-part with x is,
A = Indices that are empty within range (L_{xmax} to R_{xmin});
B = Indices that are empty within range (L_{xmin} to R_{xmax});
C = how many integers >= X
if(B>C) answer is 0 , No such permutation possible. (**Why ? Figure it out**)

Otherwise, answer = answer * $^{(A)}\mathbf{Combination}_{(1)}$ * $^{(C-1)}\mathbf{Permutation}_{(B-1)}$
Correspondingly, update C as C=C-B, and will mark the range (L_{xmin}, R_{xmax}) as visited/ used/ filled up.

So, this thing will repeat for rest of x = N-1, N-2, …, 1
Finally, the answer will be answer * factorial of (C)

Problem Category: Greedy, segment Tree, Combinatorics.

Problem E: Trees are Beautiful

Author: Mohammad Ashraful Islam
Alternate writers: Rezwan Mahmud, Rumman Mahmud, Monirul Hasan
Analysis:
Step 1: How can we find all pair distance summation?
It is very obvious that we can find all pair distance summation by the following formula:
For each edge E_{xy} where E_{xy} connects node x and y,

P_x = number of nodes with subtree tree rooted x
Q_y = number of nodes with subtree tree rooted y.
W_{Exy} is the edge cost.

So,all pair distance summation, **APDS** = $\sum W_{Exy} * P_x * Q_y$

Step 2: What will happen if we increase the weight value of an edge by 1?

Lets, edge E_{xy} connects node x and y. The weight of the edge is W_{Exy}.
P_x = number of nodes with subtree rooted x
Q_y = number of nodes with subtree rooted y.
So, it provides $W_{Exy} * P_x * Q_y$ value in **APDS**,
If we increase the weight value of the edge, it will be $(W_{Exy} + 1) * P_x * Q_Y$
So, increasing the value of wight of an edge E_{xy} by 1 adds $(P_x * Q_y)$ value to APDS.

Now, the idea is simple,
For each edge, E_{xy} calculate corresponding $(P_x * Q_y)$.
Sort the array in descending order. Then its greedy choice.

Category : Greedy, Tree, Observation, Ad hoc.

Problem F: What happens if you sum?

Author: Md Mahbubul Hasan
Alternate writers: Hasnain Heickal, Md. Kazi Nayeem, Tarango Khan
Analysis: The answer is: sum of LCM(a[i], a[j]) = sum of (a[i] * a[j] / gcd(a[i], a[j])). If we can find T[g] = sum of all a's where the a[] is divisible by g then we can easily solve the problem.

How to find T? First initialize the array T with: T[a[i]] += a[i]. Next run a loop of i in descending order. While you are at i, run a loop of j over it's multiples and add T[j] to T[i]. Once the loop of i ends, you have the desired array of T in O(nlogn).

So we have the array T, how to solve the problem? Let S[g] = T[g] * T[g]. S[g] now contains *almost* sum of all a[i]*a[j] where gcd(a[i], a[j]) = g. Actually we know what does S[g] contain. It is, sum of all a[i]*a[j] where gcd(a[i], a[j]) is a multiple of g. Now, in the same way we computed T array, we can compute the case for gcd(a[i], a[j]) = g. Again run a backward loop, and subtract S[j] from S[i]. That's it!

If you sum all S, you will find the desired sum *almost*. Just note that, you counted the pair (a[i], a[i]) and also each pair twice once as (a[i], a[j]) and the second time as (a[j], a[i]). So you can do necessary subtraction and division by 2.

Problem G: Where are Nordoma's Kit?

Author: Md Mahbubul Hasan
Alternate writers: Hasnain Heickal, Sabit Zahin, Nafis Sadique
Analysis: Not much to say. Learn the solution to the problem alien from IOI 2016. The solution technique is known as "alien trick" hence on.

Problem H: Droplets

Author: Shafaet Ashraf
Alternate writers: Sabit Zahin, Rumman Mahmud, Shadab
Analysis: Let's define a 'square' as an area surrounded by 4 cells. This problem can be simplified to 'find the number of squares that are accessible from an 'open-top square' in first row. Scan the first row and whenever you find a square which doesn't have any line on the top, run a dfs/bfs to count the number of squares reachable from there. Time complexity per case is O(n * m) as you will visit each square once.

Problem I: Round Table

Author: Hasnain Heickal
Alternate writers: Raihat Zaman Neloy, Rezwan Mahmud, Md Mahbubul Hasan
Analysis: Main observation to solve this problem is that one of the circles will cover one continuous segment in the convex hull, and the other circle the remaining part. To find minimum enclosing circle of a bunch of points, there is a randomized algorithm that runs in linear time. Once you have these observations, you can solve this problem in many different ways.

Solution 1: Fix a point as the start point of a segment. Next you can binary search for the end point of the segment. For each guess, find the Minimum Enclosing circle for each of the segments. You have n candidates for starting point, you are binary searching for the end point and you can find the candidate circles with O(n) time. In total O(n^2logn).

Solution 2: You can do ternary search instead of binary search in the above solution and the author was generous enough to let such solution pass.

Solution 3: If you look at the randomized algorithm of the minimum enclosing circle you will see there is the scope of implementing DP. Do that and you can overcome the time limit.

Solution 4: First find the MEC of [i, j]. Next attempt to find MEC of [i, j + 1]. Instead of calling MEC, first check if the j+1'st point is inside the MEC of [i, j]. If so, then you know that MEC of [i, j] is the MEC of [i, j + 1]. This optimization makes the solution pass too.

Problem J: Greatest GCD Ever

Author: Hasnain Heickal
Alternate writers: Ashraful Shovon, Sabit Zahin
Analysis:
Method 1: The output always will be **abs(A).** Because ***gcd(0, A) == abs(A)*** and we can never find a scenario where **gcd(i, A) > A (i ∈ [0, N])**. Complexity **O(1).**

Method 2: As the range of **N** was small (N <= 100), we can also iterate over **N** and find the maximum answer. Complexity **O(N)**.