Problem A

Setter: Anik Sarker
Alter writer: Raihat Zaman Neloy, Mohammad Nafis Sadique
Solution:
Build suffix Array on **S**. For each query, find the suffix with id **C** in the suffix array. Let's say its rank is **R**. Now in the rank array from 1 to R, find the position **P** (where P is the smallest one) such that **LCP(P, R) >= D-C+1**. For the rank array from R+1 to N, find **Q** (where Q is the largest one) such that **LCP(R, Q) >= D-C+1**. Now, in the range of **P** to **Q** find the number of suffix ids that fell in the range **A** to **B-(D-C+1)**. This count will be the answer.

Problem B

Setter: Raihat Zaman Neloy
Alter writer: Md. Shafiul Islam, Hasnain Heickal
Solution:
The problem was so hard to solve that the setter isn't willing to provide any solution sketch.

Problem C

Setter: Hasin Rayhan Dewan Dhruboo
Alter writer: Sourav Sen Tonmoy, Sabit Anwar Zahin
Solution:
Let's try to find the solution for a fixed number of bracket types **x.**
If there was no restriction on using them **at least once** the answer is = **Catalan(n) * x^n**
But for our current version we can use inclusion and exclusion and write it like this :

$$\sum_{0 <= i <= x} (-1)^i * \text{Catalan}(n) * {}^{x}C_i * (X - i)^n$$

It can be easily seen that it is actually = **Catalan(n) * Stirling2nd{n, x} * x!**
So we need to find Stirling2nd(n, x) for all 1 <= x <= K. It can be done using NTT.
Stirling2nd(n,k) is the coefficient of x^k in the product of polynomials A and B, Where A(i) = (-1)^i / i! and B(i) = i^n / i!

Problem D

Setter: Jannatul Ferdous
Alter writer: Imran Bin Azad, Shahed Shahriar, Raihat Zaman Neloy
Solution:
The first thing to do is to sort the adjacency lists so that we can get the children in ascending order. Now we will need to run a DFS so that we can get the starting and ending pointer for each node's subtree.
For each update, we can easily brute force over the nodes which are still empty. But we will need to delete the nodes from our list which are being filled. For maintaining this list, we can use different types of data structures:

- STL set: Complexity O(N * LogN)
- Disjoint set union: Complexity O(N)
- Segment tree / BIT: Complexity O(N * LogN) or O(N * Log^2N)

Problem E
Setter: Sabit Anwar Zahin
Alter writer: Md. Nafis Sadique, Tanzir Islam Pial
Solution:
For K=2, this is basically the prime counting function that can be solved in sub-linear time or faster using Legendre's or Lehmer's algorithm.

For K > 1600, the answer will be always 0 because no numbers less than 2^31 exist with more than 1600 divisors. This number can be found using another program, but it's not strictly required. One can use the fact that the maximum number of divisors of any number between 1 to N is roughly cube root of N. But point being, for large K, the answer will be 0.

For K >= 3 && K <= 1600, need to use backtracking and use the prime counting function for the last loop. Like if K=6, then these are all the valid possible choices: p^5, p1*p1*p2, or p1*p2*p2 (assuming p1 < p2), where p, p1, p2 are primes. When we have multiple choices for p, we will use a loop to iterate and backtrack across all of them and for the last part, we can calculate and use the prime counting function directly to get the count.

Problem F
Setter: Md. Shafiul Islam
Alter writer: Md. Ashraful Islam, Sourav Sen Tonmoy
Solution:
dp[i] = Minimum number of stones needed to keep the maximum jump length equals to **i**
Then for each query just find out the minimum x such that **dp[i] <= S**.
Now, dp[] can be precalculated. Let **diff**$_i$ = **A[i] - A[i - 1]** and **A** is sorted in ascending order. For this difference (**diff**$_i$ **)** frog's jump length can only be from **1** to **diff**$_i$. For each jump **length (j)** the number of stones required = **(diff**$_i$ **/ j) - 1**. And if **diff**$_i$ is not a multiple of **j** then we will need one more stone. Add the number of stones required to **dp[j]**. Do this for all the **diff**$_i$ **(**$1 \le i \le n + 1$**)** Key observation is sum of all the **diff**$_i$ **<= M**.

Problem G
Setter: Md. Farhan Hasin
Alter writer: Rezwan Mahmud, Abdullah Al Munim
Solution:
Up to a certain day, the answer will be always yes, and after that day it will be always no. So we can do a binary search on the days. After we've fixed a day, we can build two convex hulls for malignant and benign data points individually and check if they intersect or not. If they do then they are not linearly separable and we go left. Otherwise, they are linearly separable and we go right.

To check if two convex polygons intersect, we can check two cases:

1. If any vertex of a polygon is inside the other polygon, the polygons intersect. We can check if a point is inside a convex polygon in O($\log N$).
2. Otherwise, we can take the overall convex hull of the two polygons and check if the points from two polygons form two different ranges on the hull. If they form more than two ranges, they intersect.

We also have to take care of the edge cases where one set of points doesn’t form a polygon or all the points are collinear or one set of points is empty.

The overall complexity is O($N \log^2 N$).

Problem H

Setter: Rezwan Mahmud
Alter writer: Shadman Shadab, Mesbah Tanvir
Solution:
Let cxor [i] be equal to a[1] ^ a[2] ^ .... ^ a [i] (just cumulative xor from position 1 to position i)

Now xor from L-th position to R-th position is equal to cxor[R] ^ cxor[L - 1]. So we need to create the array such that cxor[R] ^ cxor[L - 1] is equal to X for every (L, R, X) query.

Now let's think of a graph having N + 1 nodes. Nodes are numbered from 0 to N. Then for every query (L, R, X), let's add an edge between R & (L - 1) and give it's weight equal to X. Now check for inconsistency.

If the graph is consistent then for every component - put the value 0 on the leftmost index. Other values will be fixed automatically.

Now to check consistency - you need to create a DSU and ignore the edges which create any cycle (as those edges are redundant - as the weight of such an edge can be inferred from the current tree). But before ignoring them - make sure that they appeared with proper weight. If that's not the case then there exists no solution.

Problem I

Setter: Saad Muhammed Junayed
Alter writer: Md Shiplu Hawlader, Tanzir Islam Pial
Solution:
Consider Mozzarella as a graph. There will be at most one cycle in the graph.
Let find the probability that Mozaralle will be connected. It will happen if both Batman and Joker selected the same set of roads or Batman missed exactly one of the roads that Joker selected and that road is in the cycle. These cases can easily be handled mathematically.
Let x = no of edges in the cycle
First case : C(n,k)
Second case: C(x,1) * C(n-1,k-1) * C(n-k, 1)

Problem J

Setter: Shahed Shahriar
Alter writer: Raihat Zaman Neloy, Mesbah Tanvir
Solution:
So, each contestant is given a bitmask of solved problems and a time penalty.

If two contestants have the same set of solved problems then the time penalty will determine their position.

If contestant A solved all the problems that contestant B solved and one or a few more, then contestant A must be over contestant B.

If there is some problem that contestant A solved but contestant B didn't and there is some other problem that contestant B solved but contestant A didn't then their position can be changed between them.

Because, for a particular contestant, we can give X (infinite/very large) points for solved problems and 1 for unsolved problems. So, if this contestant has at least one problem solved that the other contestant didn't then his score will be larger than the other contestant. This is true for every other contestant for this particular contestant.

So, the problem boils down to this:
You are given some bitmasks. For each mask (contestant), you have to find how many proper super-masks are there + number of exact same masks having less time penalty than this contestant.

You can do this with Dynamic Programming. There is a technique called Sum of Subset DP (SOS DP).

Problem K

Setter: Hasnain Heickal
Solution:
If there is exactly one maximum value, then the company will be biased for world finalists, otherwise it’s not.