Problem A (Average of Combination)

Setter: Hasnain Heickal
Requires some observation, first of all, we don't need to use all the values from 1..N, if you write a brute-force then you will see that you need the values N-72..N in the worst case, which also takes at most 25 different numbers. Lets say N <= 72, Which gives us the following clues, the sum of these numbers will always be less than 3000. Now we can run a dp with the state (sum, last taken number, how many taken) and calculate how many different selections we can make from here. At the end we do path printing, to output the Kth set.

For bigger N, we can just generate Kth set for 72, and add offset compared to N and it will be the expected result(Just add N-72 with all the numbers :)).

(Solution Sketch from Setter, when the value of K was 500, which was later increased)
Let's solve the problem for minimum average first. Then we can easily adapt it for maximum averages.
As the value of K can be no more than 500, we can see that the average of those first 500 combinations cannot be more than 9. Because the first 9 (1 to 9) integers has 512 combinations among them. And the highest average among them is 9.
So, what is the smallest value that we can include in the combination to ensure an average more than 9? Turns out the value is 46. Why? Because the lowest average with all the possible combinations of 46 comes from (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 46)/9 = 82/9 = 9.xx. If you include any integer greater or equal to 9 in the combination, it will just increase the average more than 9.
Combining these two observations, we can see that to get the first 500 combinations with minimum average, we only need consider integers from 1 to 45. So we can run an n^4 dp (n at most 45) to find all those combinations. The dp states are: length of the current combination (n), current integer (n) and the sum we trying to have (n^2). Using the length and sum, we can determine the average.
The above algorithm will work in the same way for maximization problem.

Problem B (Counting Inversion)

Setter: Ashraful Shovon
You can solve this problem in DigitDP style. First of instead of calculating the answer from x to y, compute from 1 to y, and then subtract the answer for 1 to x-1. So now we need to worry about just one number. Consider dp state as "How many digits in the prefix (from msb) are matching to the input number". You can then run a loop for which digit to place on the current place. If you place a smaller digit, you know you can place any number in your right side. So it's pretty easy to compute how many inversions are there in the right side in total, how many inversions are there between left-right and so on. If you place equal digit, just move on to the

next phase of the DP and you solve it recursively. Actually, you don't need DP here. Since you are not calling same state more than once. Complexity O(length of x * 10).

Another solution exists if you just to a DP with the frequency of the digits used as state. So if you use the digits 1, 2, 3, 1, it doesn't matter the order you used them. You can just keep these digit counts in a hashmap and do a standard digit DP. Since this was not the intended solution, it was required to do a little bit optimization in order to squeeze it through the TL. Using unordered_map or writing your own hash table helps.

Problem C (Divisors of the Divisors of An Integer)

Setter: Shahriar Manzoor
The way I would approach this problem is to, first try to understand what happens for n = p^k for a prime p. The divisors are: p^0, p^1, … p^k. The number of divisors of them are: 1, 2, … k+1. And there summation is something like ncr(k+1, 2). Now try to find the answer for: p^a*q^b where p and q are primes. And from there I think it's not that difficult to arrive at the final answer. The only catch is that, for each prime p we need to find the power of p in n!, which is a common number theory subproblem.

Problem D (Faketorial Hashing)

Setter: Sabit Zahin
Problem type: Backtracking, greedy, linear/integer programming
Analysis: Too detailed to post here, check the following link:
https://docs.google.com/document/d/1buDTPnY-LJwkxK0WASQvdhZLZjKLg8cIEt3qvWodOiE/edit?usp=sharing

Problem E (Helping the HR)

Setter: Shahriar Manzoor/Monirul Hasan
Easy problem. Just read the story carefully and try to understand the sample. You need to parse the input string, that's a small challenge for this problem.

Problem F (Path Intersection)

Setter: Md Imran Bin Azad
If you know how to find the intersection of two paths, you can solve it for K paths. And how to solve for two paths? Some case analysis and LCA. It also helps if you break down paths into line segments and try to intersect these segments. Because a path is either one line segment,

or two line segments joined by a single node. You can intersect them separately to make the solution less tricky.

Another solution exists in log^2 N with HLD. Increment all nodes in the paths by 1 and then check how many nodes have exactly K value. This was a bit tricky but acceptable, and some teams solved it this way too.

But perhaps the easiest acceptable solution was using bitset. Idea is very simple, select any node as root and run a dfs. For each node, keep a bitset B of size N which contains how many nodes are there in the path from the root to this node. Since N is only 10000, this is feasible. For each path a and b, let l be the LCA of a and b. You can then find the path between a and b in O(N/32), using (B[a] | B[b]) ^ B[parent[l]]. To find the intersection between K paths, just keep doing a bitwise AND operation on the bitset between two paths. It's basically the brute force solution, but 32 times faster. If you look carefully at the constraints, you can see why it should pass comfortably :)
Only 1 or 2 teams solved it in this way, although it was the easiest to implement.

Problem G (Techland)

Setter: Raihat Zaman Neloy
Classical Centroid Decomposition problem. After centroid decomposition, from the query node (third type query) you go up through the auxiliary tree (the centroid decomposition tree) and ask for each node and for each candidate company, where is the closest shop of this company in your subtree? To answer this query fast, you need to keep a segment tree at each of the nodes of the auxiliary tree. And you will also need lca technique to compute the distance between two nodes quickly.

Problem H (Tile Game)

Setter: Md Mahbubul Hasan
We need some observation to solve it. First of all, the operations are reversible, that is, if you apply say RLU then performing DRL you will get back to the initial state. So it's enough to consider only clockwise rotations. Second, the shape at bottom right corner will always be same. Why? Consider a # going to another #, the second # going to third # and so on. Can a # end up at a dot? No, because then a dot needs to end up at **#** which is not possible. So The **#**'s form cycles. So just lcm of the cycle lengths is the answer.

Problem I (Triangles)

Setter: Mehdi Rahman
You should find min distance of {triangle, line, point} x {triangle, line, point}. Actually if you think carefully, it's enough to solve for "line x triangle" case. First check if the line goes through the triangle. That should be easy! Just play with parametric equation of the line. The other case is to, find the min distance from a point on the line to the triangle (the previous one is a special case). One option is to run a ternary search over the segment to find the closest point to the triangle which gets AC. But if you think a bit carefully you can avoid ternary search. If the segment and the triangle does not intersect, then one of the end points will have the closest distance or the closest distance can be found consider "line x line" case. And both of these cases are quite easy to handle.

Problem J (VAT Man)

Setter: Shahriar Manzoor/Monirul Hasan
Give away. Just print 1.15*input