Very short analysis. Feel free to discuss and ask back.

Problem A:
Setter: Shahriar Manzoor
Alternate Writer: Derek Kisman
Category: Adhoc
Difficulty: Easy
Solution: One of the very interesting problem statements. One way to solve this problem is to notice that, difficulty is only 1-10. So, you can make an array count[difficulty][leaked / not leaked]. Then you can run (d = 10)^2 to count number of pairs. But there are many different solutions possible. For example, you can do the later part in O(d = 10). Or you can keep two lists for leaked and not leaked, sort them by difficulty, then do line sweeping. And so on.

Problem B:
Setter: Shahriar Manzoor
Alternate: Monirul Hasan, Muhammed Hedayet
Category: Adhoc
Difficulty: Easy
Solution: Easy problem. Keep a few arrays. First one: month -> number of days. Second one: day name -> day number (say: monday = 0, tuesday = 1…). It will help you if you keep these day numbers consecutive over week days and start from 0. Because, now you can increase day number and get the next day (with mod 7). So you just loop for number of days of the month and count number of weekends.

Problem C:
Setter: Shiplu Hawlader
Alternate: Md Mahbubul Hasan
Category: Graph, Data structure, Union find, Lca.
Difficulty: Medium
Solution: Nice problem. First, we need to find out Ai and Bi for non-tree edges. The costs can be increased indefinitely. But can be decreased until it changes the MST. So it is the highest cost tree edge between two nodes of this non-tree edge. Which can be easily found by modified-lca. Now we need to find Ai and Bi for tree edges. First they can be decreased indefinitely. Now how much we can increase? Well for that we need to find minimum weight non-tree edges that cover this edge. Let's split every non-tree edges into two edges of type lca-end. Now here comes the tricky solution with union-find. Sort the non-tree-split edges according to their costs. Take one by one from lowest to largest. For each one "merge" all the nodes in the chain of tree edges inside the range of this non-tree edges. When you pick a new non-merged-tree-edge update its answer. As I said it is a bit tricky :-)

Problem D:
Setter: Muhammed Hedayet
Alternate: Shahriar Manzoor

Category: Adhoc
Difficulty: Easy
Solution: The answer is count of minimum appearing character in the string. Why? How? It is easy to prove necessity, but sufficiency may be difficult. Try to prove by contradiction, or may be take a leap of faith :-)

Problem E:
Setter: Shahriar Manzoor
Alternate: Derek Kisman
Category: Math
Difficulty: Easy-Medium
Solution: First, let's try to find out which numbers can contribute to the set where lcm is N. They are divisors of N. All of the divisors of N can contribute to this set. And none other numbers can contribute. So MSLCM(N) is sum of N's divisor. However, it turns out that this gives TLE. To be honest, if I were contestant I would have tried in this way and could not believe this happened. But it is true that some constant optimization code takes ~1s while this code runs > 5s. So we need to do some optimization. One of the optimizations that pop up in my mind is, I would send a table of: 2e7 / 1000 = 2e4 numbers and then per case we need to compute sum of divisors of 1000 numbers, which should not be very difficult. I am not sure if that passes TL. Another way can be, come up with some mathematical formula and run the inner loop only for prime. That is, for each prime p, go to all of its multiple, and do some computation. And judge solution does so. There are other solutions as well without any precomputation. For every query then do some sqrt(N) loop and some mathematical formula inside. Sorry that I could not be of much help with detailed mathematical formula.

Problem F:
Setter: Anindya Das
Alternate: Shiplu Hawlader, Md Mahbubul hasan
Category: Adhoc
Difficulty: Medium
Solution: For every query depending on number >= or < h we can replace the numbers with +1 or -1. Then the problem turns out to be: find out largest rectangle with non-negative sum. The trick is to limit two rows and solve for the strip. Or actually, we can convert this strip to 1d array and thus the problem is to solve for 1d: find the maximum length where the sum is non-negative. One way to solve is, O(nlogn). But my solution gets TLE. One of the solutions from contestants is also O(nlogn) type but there they optimized log(n) part. Judge solution is O(rc^2 or r^2c) type which runs for 2s / 3.3s etc, so the time limit of 6s sounds legit. One nice idea can be: two hand technique. Initially i and j both are at the beginning of the array. Now, check, what is the maximum number after j, if it is bigger than num[i], move j right. Once you stop, update your answer and move i to right until it becomes lower than num[j]. Then again move j right. This is O(n) solution to find out maximum length non-negative segment, which is quite nice.

Problem G:
Setter: Shiplu Hawlader
Alternate: Md Mahbubul Hasan
Category: Adhoc, Geometry (not quite)
Difficulty: Easy-Medium (tricky)
Solution: First my attempt was to solve it like 2d problem. That is given points in normal 2d cartesian plane and solve for rectangle / square. The solution is easy, you just need to find out max(dx) and max(dy). In hexagonal grid, you can represent the third axis as z = x + y or x - y (depending on direction of all the axes). Now figure out max(dx), max(dy), max(dz). You will get the diameter of the answer hexagon and then you need to figure out a quadratic equation to count number of cells inside a hexagon and plug the diameter / radius in it. But it does not solve the problem. There are some tricky cases where this solution does not pass. Now think in terms of algebra. We need to find a (center_x, center_y) such that distance to any (x, y) is minimized. What is the distance between two nodes in hexagon? Well, you can try to figure out, but at the end it will become: max(difference of x), max(difference of y) and max(difference of z). First two are easy but the third one is a bit tricky. But at the end of the day, you can write some inequality for (center_x, center_y) and radius D with max(x), min(x) … [actually you don’t need center_x, center_y at the end] The inequalities are very nice in my opinion, so you should try to figure them out by yourself. You can now binary search on D to see which value of D satisfy the inequalities or you can write one single if-else to figure out the optimal value of D. In my opinion this is the cutest problem in the set.

Problem H:
Setter: Md Mahbubul Hasan
Alternate: Shiplu Hawlader
Category: Number Theoretic FFT
Difficulty: Hard
Solution: It is obvious that (0, 0) has to be median. To be so, count(+, +) + count(+, -) = count(-, +) + count(-, -). And similarly we can find out another equation for dividing by y axis. And at the end of the day, you will see that: count(+,+) = count(-,-) and count(+,-) = count(-,+). So figure out how many ways you can select 2x points from count(+,+) and count(-,-) which is: ncr(total(+,+), x) * ncr(total(-,-), x). Similarly for other two quadrants you find out same thing “how to choose 2y points”. Now the rest is nothing but “multiplication of these two polynomials”. And we need to do this by number theoretic FFT. Please note, karatsuba will get TLE.

Problem I:
Setter: Tasnim Imran Sunny
Alternate: Shiplu Hawlader, Md Mahbubul Hasan
Category: Bitmask DP + Digit DP
Solution: You have 10 different digits. Initially all of the digits are available, then you take some set of these digits and will try to use them. Rest of the digits goes to the same function (bitmask dp). How many ways to use these digits? That is another dp, this time you need to use all these digits and must not exceed N, so digit dp. You just need to be careful with implementation.

Problem J:
Setter: Monirul Hasan
Alternate: Zobayer Hasan, Derek Kisman, Md Mahbubul Hasan
Category: Geometry
Solution: Typical geometry, but way more corner cases than you can imagine. The data was "finally" prepared just 4 hours before the main contest. Initially after couple of give and take, two solutions(A and B) from judges agreed with each other. But then two more judges(C and D) wrote solutions and none of these solutions matched with each other, neither with previous two agreed solutions. We also could find some cases where each of the codes was giving wrong answer. One of the judges(B) rewrote his code in a simulation "sense", another judge(C) removed some optimization from his code and finally they agreed on output. But the happy moment did not last much, one of the other judges(D) sent his code which disagreed with (BC). So after some random small input, we found a small case where code of D fails. D fixed his code again and BCD lived happily ever after. And all these happened 48 hours before the contest, mostly 12 hours before the contest.

Back to the problem, the problem is easy unless you try to be smart. Just be stupid. First, for the each of the given points (r, t) find out what is the original t (say t'), that is: t + 2n*PI, you need to find the value of n. Also given t, find out the two lines between which this point lies. Suppose line L1 lies before the point and L2 after the point. So now, find out the spiral segment by simply looping (note a can be 1 in worst case and r is 100 in worst case). You could do binary search or even O(1) computation which yielded to so many corner cases. Also don't forget that you may need to wrap rotation to find out L1 and L2. Once you find out all the 4 corners of the spiral segment, rest is just integration. You don't need any fancy integration, just some easy polar coordinate integration.