Problem A: A Directed Graph
Problem setter: Md Mahbubul Hasan
Alternate Solution Writer: Md Shiplu Hawlader
It is a puzzle. And yes it is slightly modified from real life scenario. Unfortunately there were lots of submissions with just the sample code printing “10 20 30”. Once you read the main part of the problem, you should identify that all the money are paid by Alpha. So the correct output is “68 0 0”.

Problem B: Wrong Solution
Problem setter: Hasnain Heickal (Jami)
Alternate Solution Writer: Md. Nafis Sadique
One of the smallest case that fails the algorithm given is {1, 5, 3, 7}. One can adopt many different ways to make the test case working. One such solution would be to put 10 ninety six times then append 1, 5, 3, 7 at the end.

Problem C: Grid Construction
Problem setter: Shafaet Ashraf
Alternate Solution Writer: Muhammad Ridowan

Consider one character at a time and find the total cost for that character only. Thus you can imagine that the grid consists of only 0 and 1 and for each 1 you have to find the closest one before current cell.
You can solve this using simple dynamic programming. Let dp[i][j] denote the closest one before cell(i, j). You can update the value of dp[i][j] from the value of dp[i-1][j], dp[i][j-1].

```
def  calc() {
    res = 0;
    dst[ ] ← inf
    for(i = 0; i < n; i++) {
        for(j = 0; j < m; j++) {
                if(i > 0) dst[i][j] = min(dst[i][j], dst[i - 1][j] + 1); //update from left
                if(j > 0) dst[i][j] = min(dst[i][j], dst[i][j - 1] + 1); //update from top
                if(matrix[i][j]) {
                        res += min(dst[i][j], k);
                        dst[i][j] = 0;
                }
        }
        for(int j = m - 2; j >= 0; j--) dst[i][j] = min(dst[i][j], dst[i][j + 1] + 1);
//update from right now
    }
    return res;
}
```

Complexity is O(n*m*number_of_different_letters)

A small side note, in this solution the constant factor is as big as 26. But it is possible to reduce it to 2 if one wants.

Problem D: Double Trouble
Problem setter: Sadia Nahreen
Alternate Solution Writer: Md. Nafis Sadique
Expected solution was O(n) DP, although some solved it by centroid decomposition technique. First root the tree at arbitrary node. Let dp(i, j) = number of nodes in the subtree of i which is j distance from i. Here distance = sum of value(i) [value(i) = 0 or 1 (input)]. Note, we are not interested in dp(i, j) for j > 2. So in total we have O(n) states. By dynamic programming technique you can easily compute all the necessary dp values.

The only case remaining to handle is “cross subtree paths”. Note, all the valid “cross subtree paths” have a unique “nearest to root” vertex. We will handle that case when they are at “that vertex”. How? It is simply case analysis (you can avoid case analysis also, if you want) like: “how many ways to 0/1/2 cost way through this branch and other branches” and do some addition, multiplication (and division if you over count).

You might want to solve the problem for general value of k meer zones (instead of k = 2 in the problem). Above mentioned solution can be adopted to O(kn) solution.

Problem E: Bigger Picture
Problem setter: Md Mahbubul Hasan
Alternate Solution Writer: Mohammad Kaysar Abdullah, Md. Nafis Sadique

First sort the rooms by their size (descending order). Suppose Hi is the size of the rooms after sorting and Pi is the original index of the room.

Now imagine how the selection process goes on. First H1 - H2 will be selected from room P1. Then 2(H2 - H3) will be selected from room {P1, P2}, then 3(H3 - H4) from {P1, P2, P3} and so on.

Let’s consider the i’th step where i(Hi - H(i+1)) will be selected from room {P1, … Pi}. Suppose if you sort {P1, … Pi} it becomes {S0, … S(i-1)}. Then it is quite easy to see that k’th will be from S(k % i) [the pattern is: S0, S1 … S(i-1), S0, S1, … S(i-1)...].

If you understand the above steps, then the next stage is to combine all these steps together. Let’s sort the queries in ascending order. We will find the answers one by one. We will keep a variable to denote which step we are currently in (see above for the definition of step). Say it is

x'th (initially x = 1). If the current query is from x we will proceed to finding out the answer. Otherwise we will increase x. In this way we confirm that, when we find the answer for current query we are in the correct step. Now, for x'th step maintain a sorted set {P1, P2 … Px} and using some modulo operation you know the value of k (see above for definition of k) and thus find out k'th smallest number from the sorted set. Unfortunately stl::set does not support k'th smallest operation. You can use segment tree, binary indexed tree, nested binary search to solve this part. Author used order statistics stl set. If you are interested you can read it here.

Problem F: Counter RMQ
Problem setter: Mir Wasi Ahmed
Alternate Solution Writer: Md. Mahbubul Hasan

It's easy to see that it is possible to decompose all the given RMQ intervals to a set of non-overlapping intervals, which are always inside the given ones (no partial covering). For instance: [1 7] [3 9] can be decomposed into [1 2], [3 6] and [7 9]. Let's call them *decomposed interval.* With a little thought, it's apparent that for each *decomposed interval* **d,** assigning the *maximum* given answer of all the given RMQ intervals that contain **d** to all index of **d** is enough to generate a valid array. Fill all the unused indices with arbitrary values from 1 to 20,000.

If you think a little further you will understand that, decomposing intervals are not necessary. You can do the "taking max" for each index. The constraints were small enough for just to brute force for each index. Complexity O(QN) where Q is the number of given RMQ segements and N is the size of the array.

A side note, you can use segment tree to get O(QlgN) solution.

Problem G: Repeater
Problem setter: Muhammad Ridowan
Alternate Solution Writer: Shafaet Ashraf

The judge solution is highly optimized. I would not wonder if many teams suffered with TLE in this problem. The main idea is, a repeater will hit D if the parameter of the repeater R is divisor of D. Said that, the rest of the solution is bitmask dynamic programming. The state is 2^n size, denoting which vehicles are destroyed. On a dp iteration, find out the first "not destroyed" vehicle. Then loop through all the divisors of this number. You need to precalculate all the divisors of the numbers and the bitmask of the vehicles that would be destroyed on application of each of these divisors.

Problem H: An Interesting Game
Problem setter: Md. Nafis Sadique
Alternate Solution Writer: Hasnain Heickal

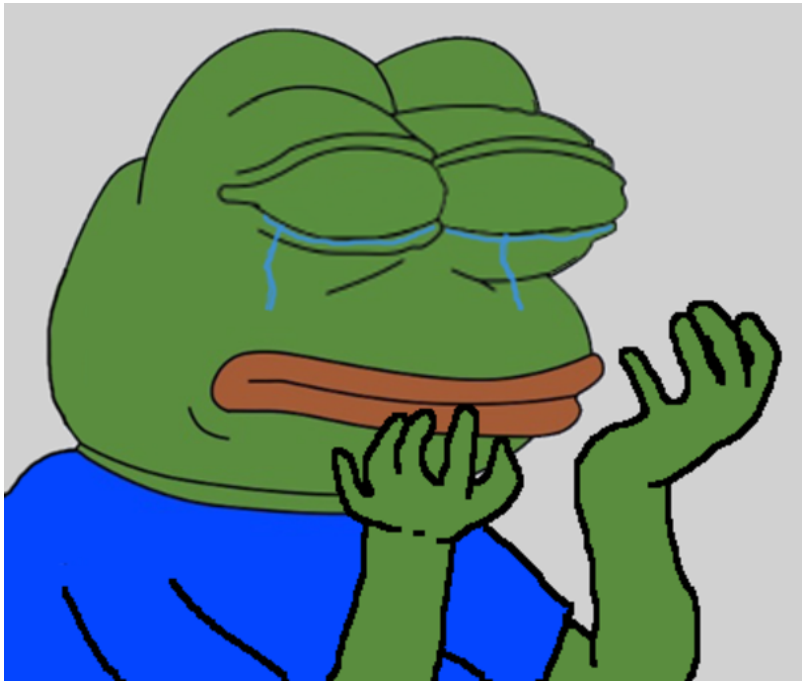
The idea is to use square root decomposition. That means, we need to group the permutation into K buckets, where K = √N . After that there are two types of operations for each query. Either the result will be inside one of the buckets or it will be coming from two or more continuous buckets. We can handle both of the cases separately.

For the case of results coming from a single bucket we can check all possible (i,j) segments where i<=j and both i and j are in the same bucket. For each (i,j) pair we can know the range of value it will satisfy. Now notice that the length of a segment(j-i+1) would be K at most. So we can run a swipe line for each segments length. The number of segments will be at most N for a particular length. Then we can run a swipe line. We need to be careful & creative here so that the complexity doesn't increase.

For the other case, we need to know for each bucket and for each query, how far right we can go if we start from the left end of the bucket, and how far left if we start from left. We can do it pretty fast in a dp kind of way. Overall we need to pre calc and optimize a lot. After that we can find the result from these values. Complexity is O((n+q)*√N).

Solution here: https://drive.google.com/open?id=0B31eriEM9iVbdFM3NXo2NXhLTlk

Nobody solved this problem in the contest. And this made the author really sad.



Problem I: Gadgets of Tomishu
Problem setter: Md Mahbubul Hasan
Alternate Solution Writer: Md Nafis Sadique

It is quite straightforward to notice that, the number of valid numbers of n length follows fibonacci pattern. For example, 2 of 1 length, 3 of 2 lengths, 5 of 3 lengths and so on. So the problem reduces to finding K^fib(n) % M for arbitrary M. Since n is quite big it is not feasible to find out fib(n) and then do the K^fib(n) % M using big mod. Also you can not do K^(fib(n) % M) % M. Apparently many walked in this two ways and got WA.

We know, fib(n) = fib(n - 1) + fib(n - 2). So we can write, K^fib(n) = K^(fib(n - 1) + fib(n - 2)) = K^fib(n-1) * K^fib(n - 2).

So if we denote, K^fib(n) by f[n], then f[n] = f[n - 1] * f[n - 2]. We just need to apply modular operation on top of it. n was small enough to perform O(n) operations per test case.