Fall 23 Div II Week 3 - Solution Sketches

(taken from editorials of original problems)

Problem A

(Source: Codeforces Round 238 (Div. 2) Problem A)
Observe that in the final configuration the heights of the columns are in non-decreasing order. Also, the number of columns of each height remains the same. This means that the answer to the problem is the sorted sequence of the given column heights.

Solution complexity: O(nlogn), since we only need to sort. Make sure you know how to use the default sorting methods from your favorite language: here is a helpful link.

Problem B

(Source: Problem by Pedro Paredes)
There is a dent if the position of two consecutive cars is overlapping. So to check this, sort the car positions and then check if any pair of consecutive cars overlaps.

Problem C

(Source: Codeforces Round #661 (Div. 3) Problem A)
Firstly, let's sort the initial array. Then you can observe that the best way to remove elements is from smallest to biggest. And if there is at least one i such that $2 \le i \le n$ and $a[i]-a[i-1]>1$ then the answer is "NO", because we have no way to remove $a[i-1]$. Otherwise, the answer is "YES".

Problem D

(Source: Codeforces Round 888 (Div. 3) Problem B)
Note that after doing any number of operations, all the indices containing odd numbers still contain odd numbers and all indices containing even numbers also contain even numbers. Consequently, since the parity of the elements in the sorted array is preserved, the even and odd subsequences of the elements can be sorted separately, and the answer is YES. If the parity of the elements is not preserved after sorting, the answer is NO. Another way of seeing this is that if we sort the array and check that the positions containing odd elements are still the same (and the same for even), then the answer is YES.

Problem E

(source: Codeforces Round 521 (Div. 3) Problem E)
First note that we don't care about the problems themselves, only their count. So convert the input into a sorted sequence of the counts of each problem type. You can do this using a symbol table/map/dictionary, or by sorting the array and looking at consecutive runs of elements.

Now, note that the total number of problems is at most *n*, so we can do the following: for each number *i* between 1 and *n*, let's compute the maximum number of problems in a set of thematic contests where the last contest contains exactly *i* problems. Let's do it greedily by adding more contests one by one until we can't:

Let the number of problems in the current contest be *cur* (at the beginning of iteration this is just *i*). Let's look at the problem types we haven't used yet and pick the one with the most problems. Since we sorted the array of problem counts, this is doable in constant time. If the number of problems of this type is less than *cur*, then we are stuck and we can't add any more contests to this set. Otherwise, we add that contest to the set and we set *cur* to *cur/2* if it is even, otherwise we are stuck and can't add any more contests to this set. And now we repeat until we run out of contests.

We do the previous method *n* times, but notice that since we *cur* is initially at most *n* and we halve its value in each iteration, we can only do at most *log n* iteration before *cur* is 0. So the overall runtime is O(nlogn).

One last question: why can we always pick the unused problem type with the maximum number of problems in each step of the greedy method? Suppose we have two contests with numbers of problems *x* and *y* with $x < y$. Let the number of problems of the first contest topic be *ctx* and the number of problems of the second contest topic be *cty*. If $ctx > cty$, we can swap these topics (because $x < y$ and $ctx > cty$) and it still satisfies the thematic problems rule, which means we can always pick the problem type with most problems first.