Solution sketches for Day 2

Tower:

- A feasible brute force solution for 30 points: Use backtracking, build the tower from the bottom to the top. Pruning: There is a simple O(N) greedy check whether there is at least one possible way to build the rest of the tower using the remaining cubes. If we use this check whenever we make a recursive call, the runtime is guaranteed to be proportional to the actual number of valid towers.
- For 45 points, improve the recursive search using memoization. The state of the search is the set of unused cubes, and the index of the cube on the top of the tower so far. This gives us O(N*2^N) states, and each of them can be computed in O(N) from smaller states.
- The two approaches above can be combined to score 55 points.
- For 100 points, make the following observation: Remove one largest cube A and consider any valid tower built from the other N-1 cubes. Suppose we now want to insert the largest cube somewhere into the tower. Regardless of how the tower looks like, the number of ways in which we can do it is always the same: we can place A on top of any of the cubes that are within D of its size.
  Let C(N-1) be the count of valid towers using the first N-1 cubes, and let X(N) be the number of ways how to insert the largest cube into any such tower. It is easy to see that all X(N)*C(N-1) towers obtained in this way are distinct.
  On the other hand, if we take any valid tower with N cubes and remove the largest cube, it can easily be seen that we will always obtain a valid tower with N-1 cubes. Hence there are exactly X(N)*C(N-1) towers built using all N cubes.
  This solution can be easily implemented in O(N log N), for example by sorting and then traversing the sorted array using two indices to compute all values X(i).

PIN:

- For 15 points, try out all possible pairs if N is small.
- For 30 points (case D=1), store all PINs as boolean flags in a 36^4 array. Then, for each PIN generate all strings that differ in one digit, and use the array to check whether you hit another PIN.
- For another 30 points (case D=2) use another 36^4 array. For each of the given PINs generate all that differ in a single digit, and use the array to count how many times each string appeared. From these values the answer can easily be computed.
- For 100 points, use the principle of inclusion-exclusion. For each combination of places, compute the number of PINs that match on those places (and may or may not match on the others). For example, C(1,0,1,0) will be the count of pairs of PINs that match on the first and third place. From these values answers for all D can be reconstructed.
  For example, for D=4 (pairs that differ in all places) can be computed as (all pairs) – (for each place, the number of pairs that match on that place) + (for each pair of places, the number of pairs that match on those places) – (for each three places, the number of pairs that match on those places)

MP3Player:

- Plenty of slow correct solutions, starting with „for each T, try all V_1", which can then be improved by restricting „all T" to reasonable values of T only.
- For a fixed T, the function that returns V_2 for a given V_1 is non-decreasing, hence we can use binary search to check whether a valid V_1 exists.
- For any segment of keypresses the above function has the form: for V_1 from 0 to A-1 the function is constant, for V_1 from A to B-1 it increases by 1, and then from B to V_max it is constant again. If we have two segments for which we know their functions and concatenate them, the new function for the longer segment can be computed in O(1).
- There are several solutions with slightly different time complexities that score 100 points. One of them is based on the following idea: Use an interval tree to represent the current function for segments of the input. Start at T=infinity, then decrease T, update the functions of 1-key segments for keys that become activated, and each time update the function computed by the entire sequence. Stop as soon as this function can produce the output value V_2. The time complexity of this approach is O(N log N).