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).