Conference
The first realization is that any power of 2 is strictly greater than the sum of all the powers of 2 less than it. In fact, \( \sum_{i=0}^{n-1} 2^i = 2^n - 1 \). This means that we can always greedily take the conference that starts earliest, without worrying about the length of the conference or the other conferences it may prevent you from scheduling. So, sort the data by starting conference time, and greedily take the one that starts earliest. Once you take one, go through the list in order, skipping over anything that is blocked by the last conference you grabbed. Since no two conferences start on the same day, you never have to make a decision between two choices where the greedy selection may not work.

Dot Game Dominator
Once again a greedy strategy can be used here. You want to eat the largest dot that is strictly smaller than your current size, since at each step, this makes your size grow the most. No competing sequence which eats a smaller dot can beat this greedy strategy. To implement this, a TreeSet can be used, since the class TreeSet has a method lower. However, be careful though - TreeSets don't store duplicates and an error will be caused in situations where there are multiple dots of the same size. To avoid this error, create a TreeSet of objects, where each object stores both the size of the dot and its ID number (you assign this as you read in the dots to be unique). Alternatively, it's very difficult to make data where the number of dots you eat is very large. Or if it is, where it takes you a long time to find the largest dot smaller than you. Thus, a relatively naive implementation which maintains a sorted list of the dot sizes will pass the data that was generated for this problem.

Dueling Philosophers
This is a slight twist on the regular topological sort problem. Instead of just answering whether or not there is a valid topological sort, in the case that a topological sort exists, we just determine if there are more than one possible topological sort. Take the usual algorithm and run it to completion. This will distinguish between cases with and without a topological sort. Now, edit as follows: as the algorithm is running, keep track of if there were ever two possible vertices that could have been placed in a particular slot (ie. two vertices with a current in-degree of 0) in the topological sort. This can just be a boolean flag that is initially set to false and toggles to true any time two 0s are noted for in-degrees when the algorithm looks to choose the next vertex of minimum in-degree. At the very end, in the case that a topological sort has been found, we distinguish between outputs of 1 and 2 by simply looking at the value of the boolean flag. If it's false, we return 1, if it's true we return 2.

Strange Lottery Simulator
A regular trie where you store the number of words stored in each subtrie can handle the query asking for the number of participants with a particular prefix. However, if the names can be reversed, that same trie is not helpful. However, what can be done is that two different tries can be maintained simultaneously. Notice that reversing a name twice just brings it back to its original form. Thus, Whenever a name is added, add it to a trie forward as is and ALSO add its reversed version to a trie reverse. (So, "sharon" would get added to the trie forward AND "norahs" would get added to the trie reverse.) Finally, whenever you receive a query, you just need to know if the current state of the names is regular or reverse. Just keep either a boolean variable or an integer that toggles between 0 and 1, and then answer the query in the appropriate trie based on the value of the variable storing the current state of the names (forward or reverse).