Code Jam 2011 - Round 1A

Analysis: The Killer Word

Every year, contestants come to us saying they implemented a program correctly but it ran too slowly. Unfortunately, being too slow is the same thing as being wrong. The Google Code Jam is an *algorithms* contest, and once you get past the early questions, coming up with a fast algorithm is often the whole point of the problem.

Why do we bring this up? Because this problem is a trap. On first glance, it looks just like the first two problems from the Qualification Round. We tell you an algorithm, and you implement it exactly as described:

- Loop through every one of Sean's lists.
- Loop through every word in the dictionary.
- See how many mistakes Sean makes while guessing this word.

Since the last step requires iterating over every word again, the running time here is O($\mathbf{N}^2\mathbf{M}$).

We will always give you the hardest inputs we can fit within the stated limits, which means you *will* see **N** = 10000 and **M** = 100. Neither a fast computer nor a fast language will save you here - the whole approach is simply too slow. For example, our C++ implementation takes over 20 minutes for a single test case. To succeed on an algorithms contest, you need to see these issues in advance, either by looking at the Big O complexity or by trying a worst-case input of your own.

Once you do see the issue, it isn't too hard to dramatically speed things up. The main idea is to combine the last two steps:

- Loop through every one of Sean's lists.
- Divide the words into different classes by length. If you focus on a single class, Sean will make the same first guess for any word in that class. This is because he has exactly the same information in each case.
- For each class, figure out what letter Sean will guess, and further divide the class based on each of the different responses you could give to Sean's guess.
- Repeat for each of the sub-classes until you are down to a single word, at which point Sean will finish with no mistakes.

The running time here is O(**NM**), and it will go hundreds of times faster than the more straightforward algorithm.

This technique is similar to Dynamic Programming - the game begins in the same way for many different words, so we should not have to redo all that work every time.

Fun facts: this problem was originally called Bloodthirsty Executioner Man, but was renamed because the hangman is made out of paper, and has no blood. Sean himself answered several of the clarifications for this problem.