**Analysis**

My algorithm is slightly better than brute force with its top down approach. For example in puzzle #3, the program tries to find a first thread through all 26 cubes, but it gets stopped at the 16th cube. The program then skips to looking at all subsets of 15 cubes, instead of having to look at all subsets of 25, 24, 23, and so on. It eventually makes its way down to looking through all subsets of 6 cubes. Since no obstacle is found for that size, the previous size of 7 cubes, where an obstacle was found, is the minimal obstacle.

However it usually fails when looking through subsets near 13. There are too many combinations to look through and if an obstacle is not found, my program crashes because of stack overflow. I thought this might be because my CubeCombinations function was recursive instead of iterative. Iterative functions are much more efficient and use less memory. Converting to iterative was extremely difficult however. I was able to, but it actually did worse than the recursive version. It did what it was supposed to do, which was look through all the combinations, but there is probably a better way to implement it iteratively. Since I was low on time and had other finals I decided to stick with the recursive function because I don’t think I’ll be able to correctly implement a good iterative version.

This was a very good learning experience on how important efficiency and memory management is. I’ve never had a project in school that was large enough where any of it mattered. I’ve only had to deal with efficiency and memory management when I was making my own 3D online game.