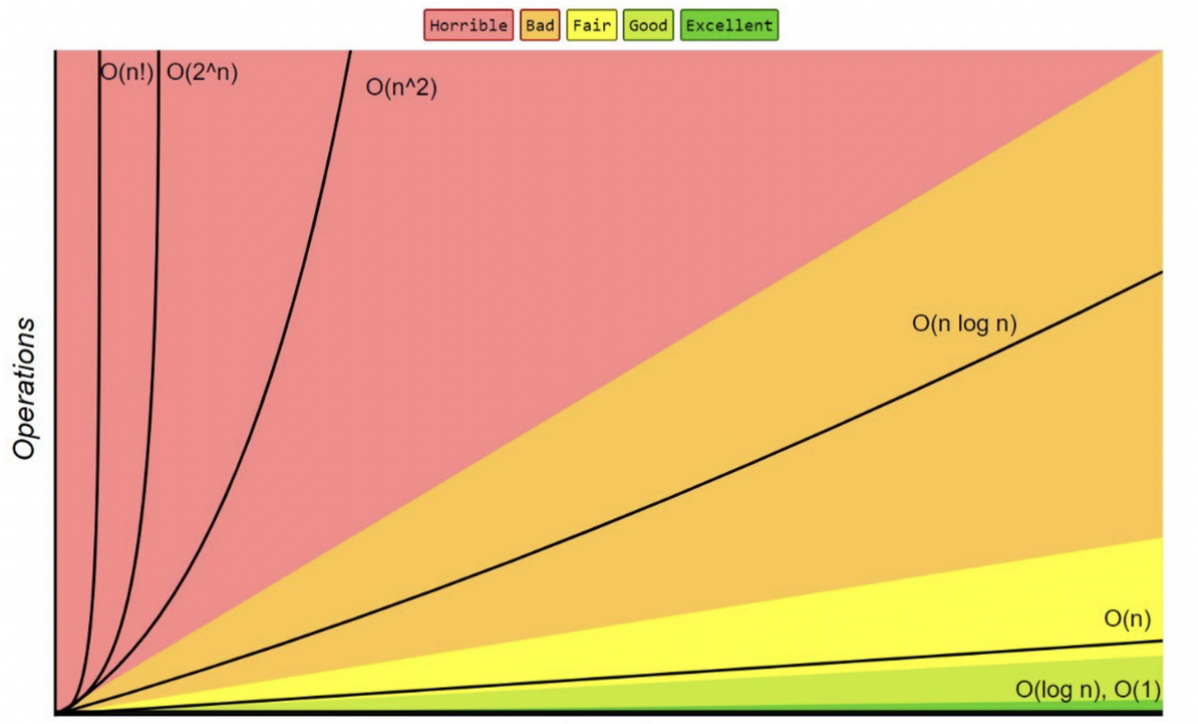
My process to find

Single Number

What is the single number problem?

The single number problem is a coding problem/exercise/challenge(whatever you want to call it) where you are given a list of integers. In this list, each integer appears twice, except for one integer, which appears only once in the list. The aim of the solution/algorithm is to return the number which is the “single number”. The number without a pair.

The challenge with this problem was that it had to be of linear time complexity and constant space complexity. This meant I was not allowed to have nested loops, or anything like that.



This is a very helpful graph that shows you time complexities and how they scale. I really like how the creator used colours to represent multiple properties of each complexity, such as how good it is. This is something I definitely want to learn to do with python, more complex graphs that display information clearly.

My though process

Always before I attempt to solve a problem following the rules of the challenge, I like to find a solution that is not the most time or space efficient, and slowly go improving it from there. This gives me multiple advantages from attempting to solve the problem with the rules first. By doing it this way, I am able to get a very good understanding of the problem, and how it works. But more importantly, this allows me to view the problem from different angles, because the best solution is never the worst solution. By approaching it from a different angle, I can learn what type of solution this program will be, and mistakes in the best solution that would not be very clear at first, become very noticeable.

Initially, I thought it would be a good idea to create a hash table, where each key-value pair would be the number of times its showed up, then the integer value in the list (not the other way around, for reasons I will explain later). I would then be able to search for the key with value 1, and return its value.

This would have had a time and space complexity of O(n), because creating the dictionary would be O(n) as we fill it up, then O(n) time complexity for looping through the list.

This seemed to work, which was good, but not great.

The next thing I attempted was to sort the list, then loop through it two items at a time, if any two pairs were not the same, I would know I hit the lonely number.

This worked, and I learned many ways to loop through a list any number of items at a time. However the way I chose to use was just the third parameter of the loop function, allowing me to choose the step I was taking each loop.

I got stuck at the end of the loop however, because what if my lonely number is last? No matter what, the list will always be odd numbered because of the lonely number. This means I would never complete a full loop around the list, as a temporary solution, I just made it return the last number if it was at the end of the loop without finding the lonely number. This doesn't work for all cases, and I might come try to fix it later but for now it passes the leetcode inputs so I guess it still technically counts as a solution.

Not quite sure what the time complexity of this algorithm is, I feel like it should be O(½n), because in worst case scenarios, we are looping through half the items in the list (+1 but i dont think thats that important)

Not quite sure about the space complexity of this either, at first I thought it would be constant, but after further research it turns out its not because the even though the sort() method sorts a list in place, its still O(n) apparently. For now I will count this as a solution because it still finds the lonely number every time, the only thing is its space complexity is O(n) instead of constant. The other way to lower my space complexity would be bit manipulation (according to leetcode hints and other people), but im not sure thats something I want to try.