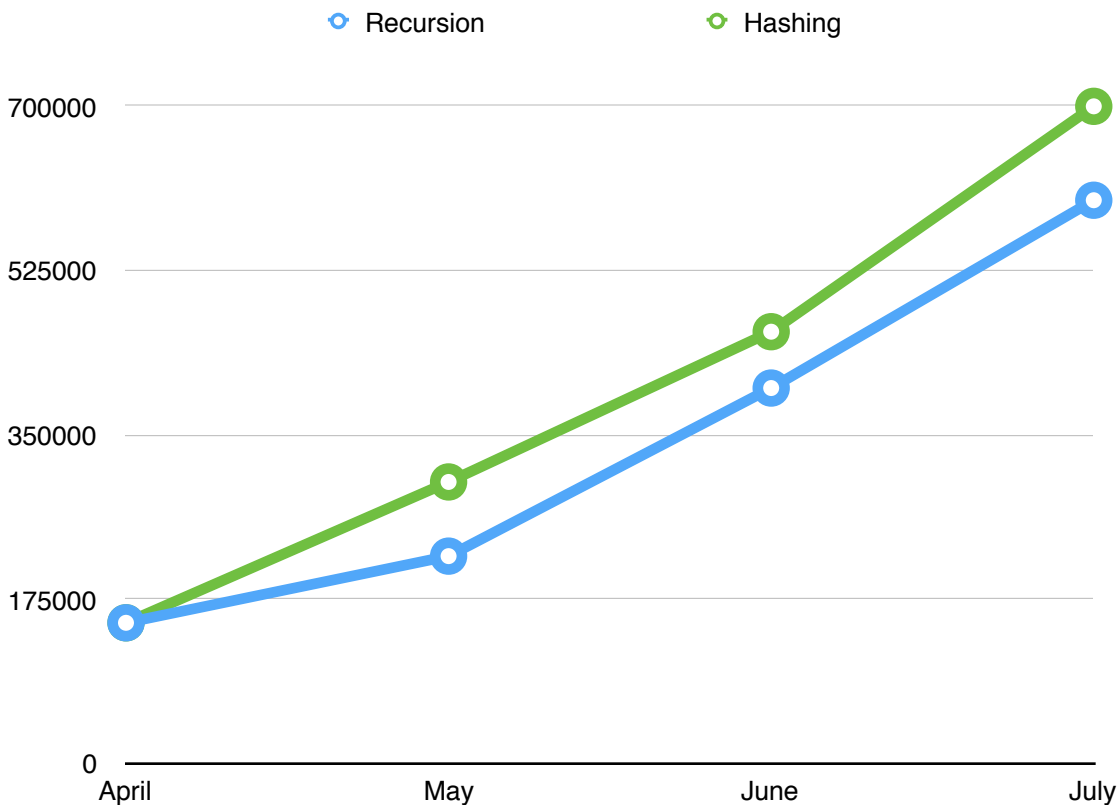


The run time of a solid hash function can be as good as O(n) depending of course on what you are looking at. Not being fully able to test it with a nanosecond counter, made me just look at the worst case scenarios based on what I read in the book, as well as online sources of run times of a hash function. Assuming mine does in fact work, I see the worst case scenario being O(mlgn) run time. As the first for loop runs through all the edges/keys of values that are in the hashmap. It then checks to see whether or not it is actually contained in the graph, by looking at the value of the index, and then adds it to the array. It then checks to see if the location is now contained in the new array list. If it is, then it maps the paired island/vertex into the hashmap.

Looking at the Recursion approach, I see the run time being slightly slower in some cases, and in others faster, depending on how many more edges there are than vertices. I mentioned in the written portion in terms of “b” and “s” that the run time would be O(b+lgs+c), the reason for that, is that the algorithm first looks at all the islands(b) 2 at a time, it would then find a match to the starting index parsed at the beginning of the program, and then look for it’s paired island(b). It would store both islands and it’s edge(b) theoretically, but not actually, as we assume the first two stored are already a matching pari. Once those are stored it looks for that new location that was just stored throughout the islands, and would find another match, and so on. The reason for it being log(s), is that we know for a fact it isn’t s^2 as we never have a double for loop checking the same last twice, yet we are incrementing by 2, and then removing those two islands that are found to be a match. Making me feel confident saying it’s a O(b+lgs +c) run time, with some extra commands in the background.