ANALYSIS OF SUBSET SUM SOLVER USING BACKTRACKING

In doing the exercise for the subset problem solver dynamically, and now the project for it using backtracking, I have realized some things. First of all, the backtracking way of coding for this project is much easier to do than the initial dynamic way of coding it. One of the reasons why is because, the backtracking way of doing this is so similar to other backtracking problems, especially the combination problem using backtracking. Backtracking is also much more complete and faster than the dynamic solution. The backtracking process can print all the solutions of the subset sum problem while the dynamic way can only print one solution at a time. If the subset problem is run by and stopped after one solution like the dynamic solution, it is much faster. Both have exponential speed but backtracking is faster by a fair margin.

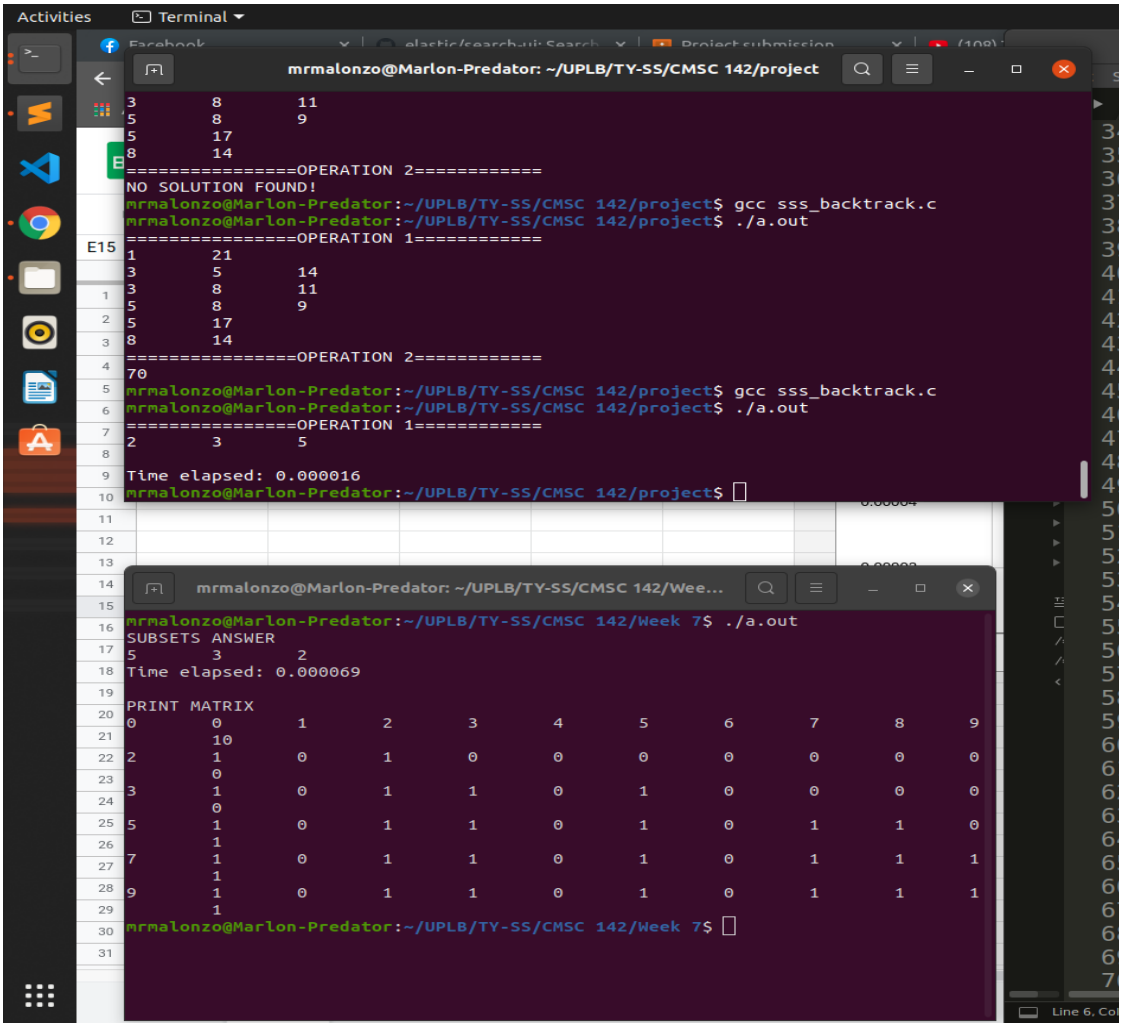

Figure 1.1 Backtracking on Top and Dynamic on bottom

As you can see in Figure 1.1, running both the algorithms with the same set of numbers which is {2,3,5,7,9} and sum which is 10, will result in backtracking being faster by a lot of margin, even though they are both exponential in running times.

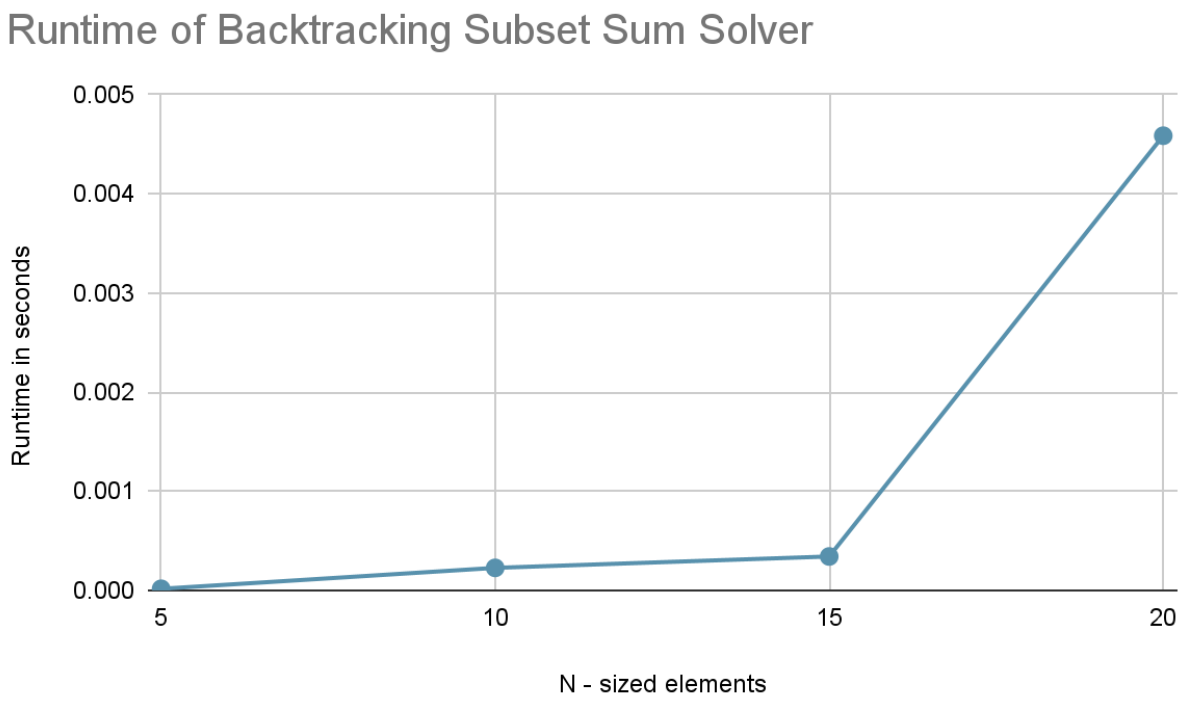

Figure 1.2 Sum is 10, 20, 30 and 40 respectively

As I've run the code, I noticed that not only the number of elements, but also the sum can affect the speed of the whole program. The bigger the sum that you want to subset, the longer time it takes for the program to finish finding all the subsets. Naturally, the distance of the target sum to the elements also plays a big factor in the speed of the algorithm. The harder it is to find a subset of the sum, the longer time it takes for the algorithm to find it. Needless to say, the background application running in your computer also influences the speed of the program, even by a little bit.