DATA

We organized our data in Microsoft Excel and created the following table to be able to construct our plots. Here we have the execution time of each program depending on the number of coins for each input file.

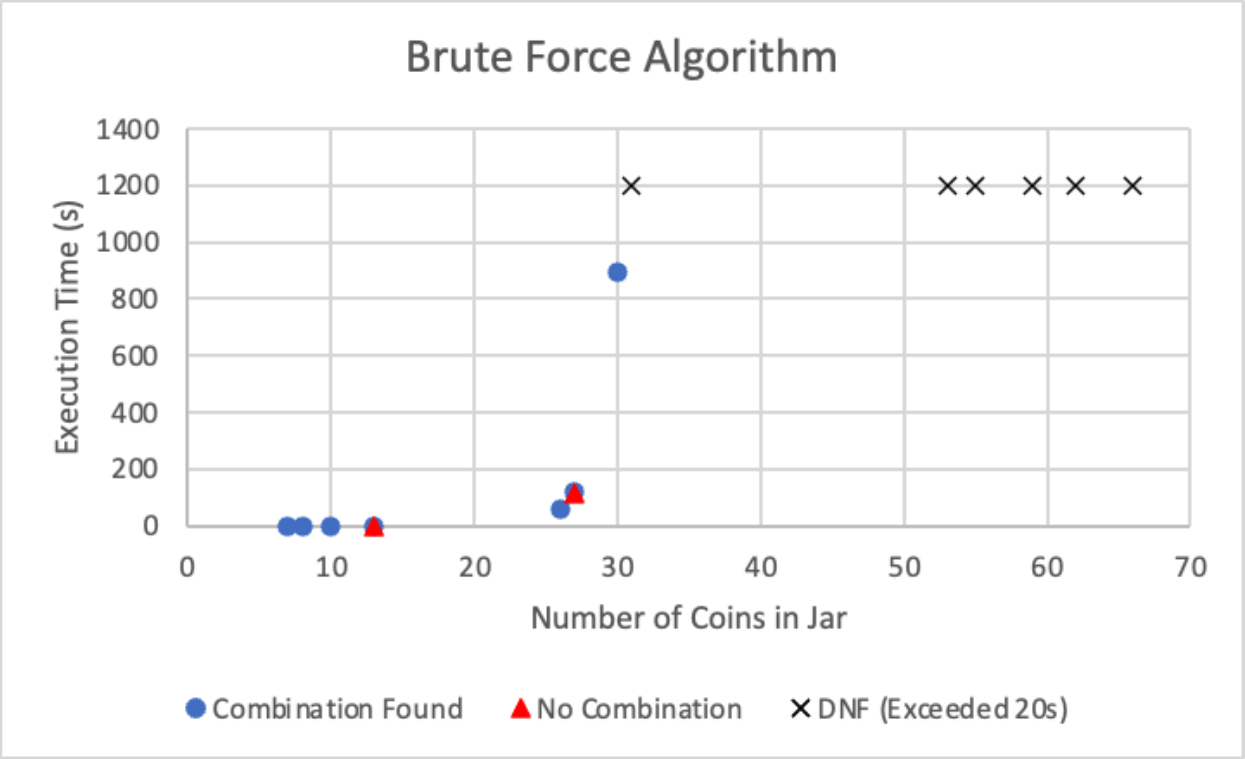
Input	Num Coins	Brute Force	Pruning	Dynamic Programming
small1	7	0.033	0.032	0.031
small2	13	0.035	0.039	0.032
small3	13	0.043	0.031	0.032
small4	8	0.034	0.034	0.032
small5	10	0.031	0.031	0.034
medium1	27	121.589	156.625	0.031
medium2	26	61.543	65.362	0.037
medium3	27	115.128	141.206	0.035
medium4	31	1200	1200	0.035
medium5	30	895.128	431.097	0.031
large1	55	1200	1200	0.038
large2	62	1200	1200	0.034
large3	53	1200	1200	0.038
large4	59	1200	1200	0.032
large5	66	1200	1200	0.034

PLOTS

We made three scatter plots for each algorithm comparing the execution time to the number of coins. The blue dots represent the times for which the algorithm found the most efficient combination of coins. The red plots represent the times for which the algorithm found no valid combination of coins. The X represents the cases which exceeded an execution time of 20 minutes. Here the program was stopped because of efficient run times that would take much longer than 20 minutes.

Brute Force Algorithm Plot

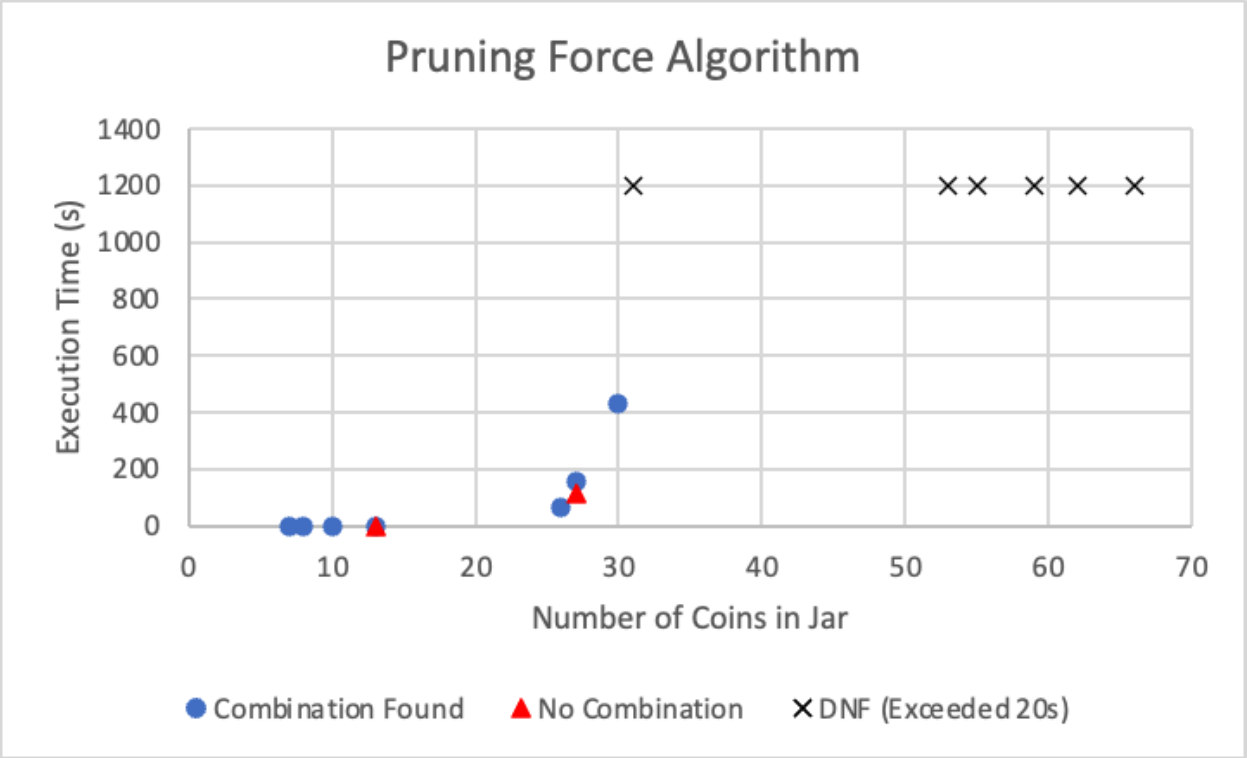
Looking at the plotted data, we can see an exponential trend. As the number of coins increases, the execution time increases by a lot more. For the cases with 13 or less coins we see a very small execution time of around 0.030-0.045. As we increment the number of coins to 26 we get an execution time of 61 seconds which is an enormous increase. By just adding one more coin, in the cases of 27 coins, we see a dramatic increase to 115 and 121 seconds, doubling the time. The exponential trend is further supported as the case with 30 coins increases to 895 seconds and then for 31 coins we exceed 1200 seconds. The time complexity is 2^N, for the middle cases, roughly doubling the time for every coin that is added to the jar. We could expect the execution time for the large cases to be near to infinite as the trend of the exponential growth appears to have a vertical asymptote at a little over 30 coins.



Pruning Algorithm Plot

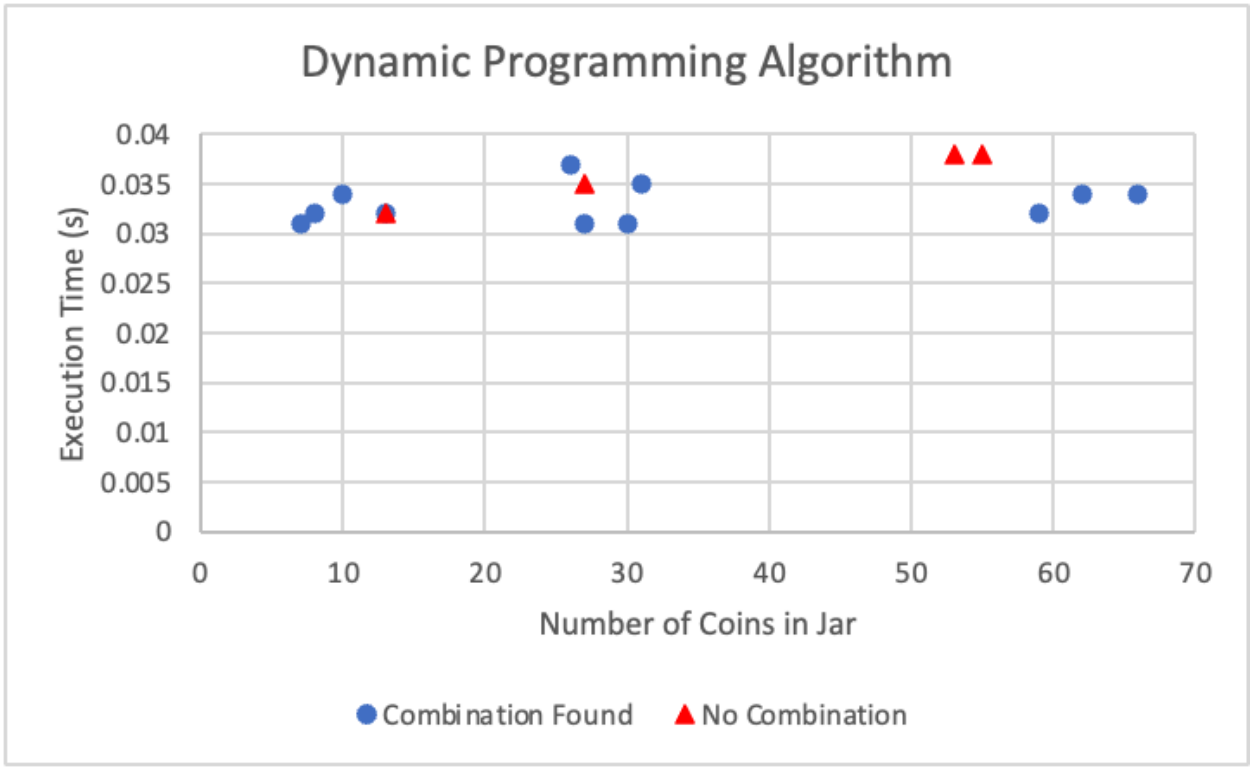
Looking at this algorithm, we implemented pruning on the brute force algorithm to make it more efficient. As we can see, for the smaller test cases it isn't any faster because there are very few combinations of coins to look through. However for larger cases it is much more efficient, proved by test case medium5, which takes 431 seconds, less than half the time for the brute force algorithm at 895 seconds. However, this code is still highly inefficient and the curve is exponential with a vertical asymptote at around 30+ coins. The larger test cases would take an extremely long time.

- same analysis as brute force
- similar to brute force but for 30 coins it halved the time so we can expect the execution time for greater data to be much more efficient



Dynamic Programming Algorithm Plot

This smarter dynamic programming algorithm is extremely more efficient than the brute force. By dividing into subproblems and caching the results, the algorithm saves a lot of time, not having to calculate work multiple times. The algorithm performed under 0.04 seconds for all cases, not noticing much difference between larger and smaller test cases. The trend in the plot is pretty constant, time increasing just very little as coins are increased. However, due to the exponential nature of these problems, we can infer that for much larger test cases, we would see a similar exponential trend where the time increases by a lot.


Dynamic Programming Algorithm
Execution Time (s)
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
0
10
20
30
40
50
60
70
Number of Coins in Jar
Combination Found
No Combination