Homework 4

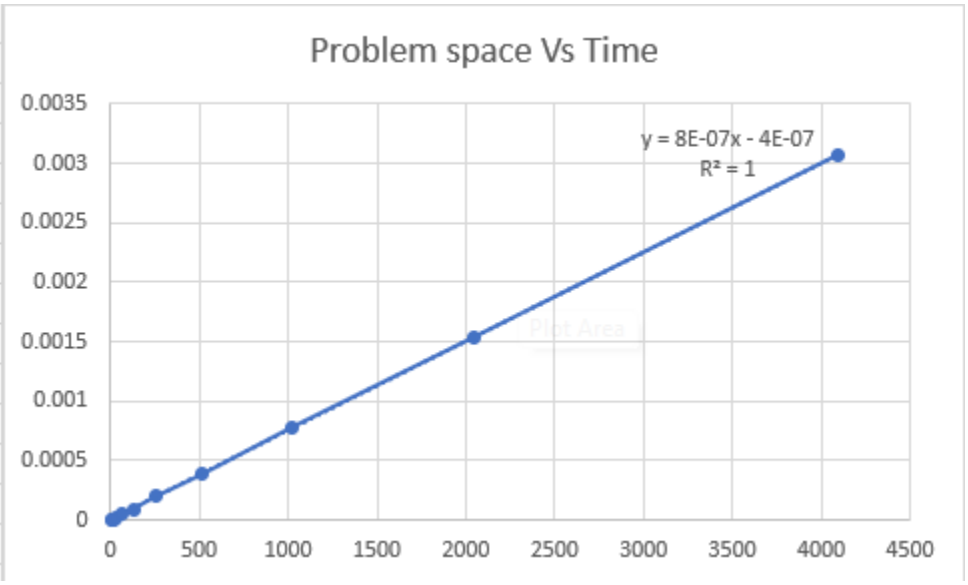
Pete Newman

11/17/23

1c) To allow my algorithm to work with rods longer than the price of the largest piece that we have data for, I did a couple of things. First, I modified the for loop in the auxiliary function. Instead of iterating up through the values starting at 1 and going to the length of the stock rod, I start my loop at the minimum of either n or the size of my price array -1. This allows me to start at either the largest piece that I have price data for or the biggest piece that I can make with the remaining rod and then iterate backwards from there. I also had to add a check that ensures that I am not trying to cut a piece that is bigger than the rod that I have available to cut from. These modifications do not add any time to the computation because the only thing that I have changed is how it moves through the data and the extra check is a simple statement of linear time complexity. In terms of space complexity, the modifications would probably decrease the required space since the space needed for the price array has decreased and the number of recursive calls is reduced since it is working from the largest pieces down.

1d)

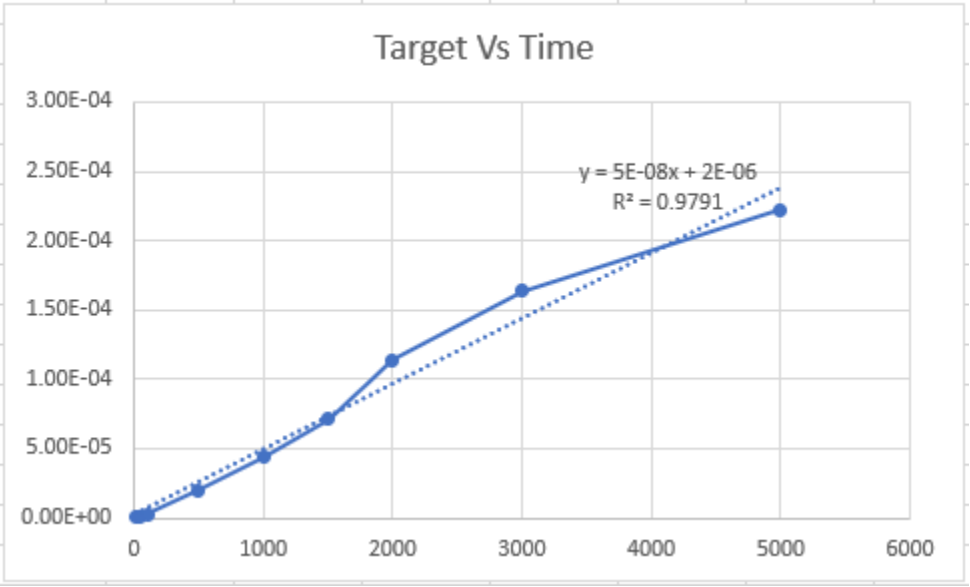
n	time
4	0.00000348
8	0.00000352
16	0.00000834
32	0.00002043
64	0.00004394
128	0.00009335
256	0.00019588
512	0.00038676
1024	0.00077948
2048	0.00153674
4096	0.00307385



Given that this algorithm only has one for loop and I did not add any loops to the code during my modifications, these results make sense because of the fact that the values are memoized. When values are memoized they require constant time after the first calculation. The trendline indicates that it is near linear time.

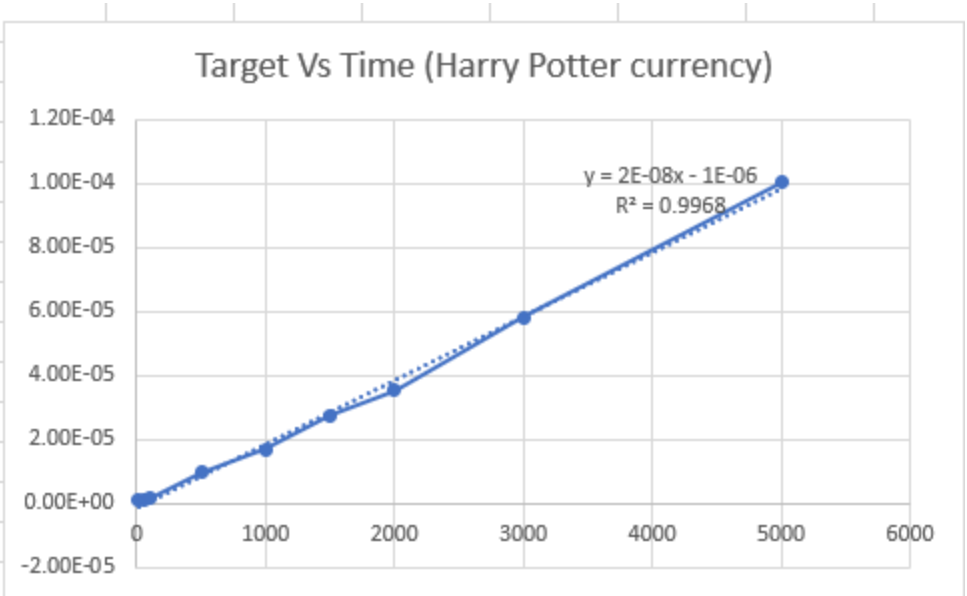
2a)

target	time
10	5.40E-07
50	1.47E-06
100	2.89E-06
500	1.99E-05
1000	4.41E-05
1500	7.16E-05
2000	1.14E-04
3000	1.64E-04
5000	0.00022



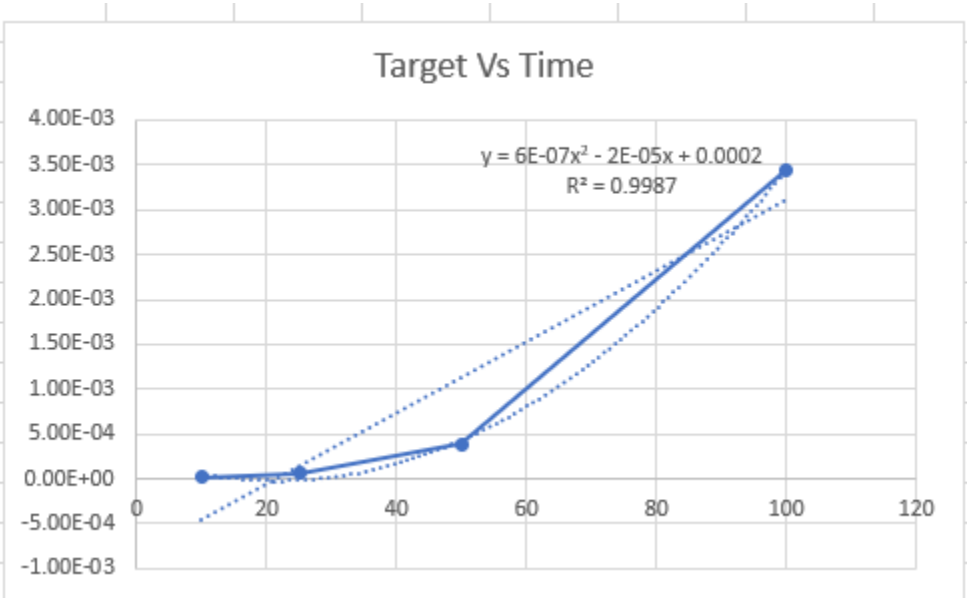
2b)

target	time
10	1.26E-06
50	1.16E-06
100	1.56E-06
500	9.70E-06
1000	1.68E-05
1500	2.76E-05
2000	3.53E-05
3000	5.82E-05
5000	0.0001

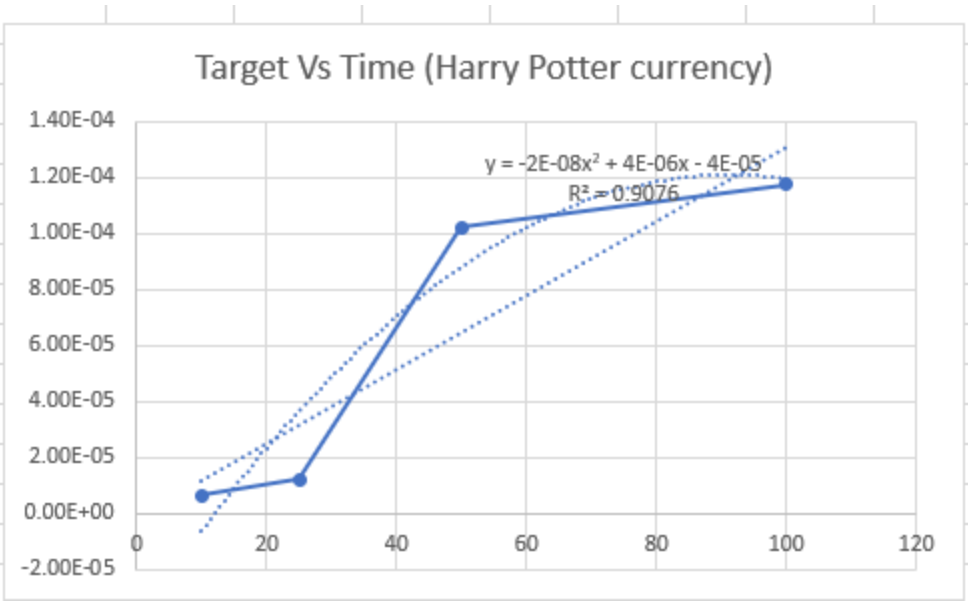


2c)

target	time
10	1.33E-05
25	6.60E-05
50	3.85E-04
100	3.45E-03



target	time
10	6.66E-06
25	1.23E-05
50	1.02E-04
100	1.18E-04



2d) The original algorithm contains 2 for loops although only one of them iterate over the size of the problem space. This would explain why the graph in 2a has more time variance; it has a larger coin array than the coins in the Harry Potter currency. Now looking at the adapted algorithms, we see an increase in the time complexity. This is due to the addition of a for loop in the code. This loop has to iterate over every combination for each value and store it so that it can be recalled later. From looking at the graphs we can draw some conclusions. The first graph is the one for the US currency. Because of the larger number of coins in the currency there are more possible combinations. The time results reflect that by being closer to n^2. The second graph shows the currency from Harry Potter. Because of the smaller number of coins in the currency, there are significantly less possible combinations and the loop does not have to iterate over as many combinations. This leads times worse than would be possible with the original algorithm and better than the times that we got from the adapted algorithm working on the US currency.