Project 2: Coin Change

Group 46

Group Members

David Brouillette
James Fitzwater
James Stallkamp

1. Theoretical Run-time Analysis and Pseudo-Code

Brute Force

Pseudo-Code

```
on changeslow(listOfCoinDenominations, desiredChangeAmount)
	Repeat i over listOfCoinDenominations
		set change to array with same length of listOfCoinDenominations to all
		0's
		if coin is equal to desiredChangeAmount
			set change[i] to 1
			return change
	set responseList to {}
	Repeat j over desiredChangeAmount
		if desiredChangeAmount minus coin is greater than 0
			set response to changeslow(listOfCoinDenominations,
			desiredChangeAmount - coin)
			Increment response[j] by 1
			add response to responseList
	return the lowest value in responseList
```

Asymptotic Runtime

The bounded runtime is O(n+W), a result from changeslow's disjoint pair of loops (denomination type and desired amount of change), which are iterated idependently.

Greedy Algorithm

Pseudo-Code

```
on changegreedy(listOfCoinDenominations, desiredChangeAmount)
    set usedCoinsTallyList to {}
    repeat length of listOfCoinDenominations times
        (end of usedCoinsTallyList) = 0

    i = length of listOfCoinDenominations
    totalUsedCoins = 0
    currentChangeAmount = 0

    while currentChangeAmount is less than desiredChangeAmount
        if item i of listOfCoinDenominations <= (desiredChangeAmount -
currentChangeAmount)

            currentChangeAmount = currentChangeAmount + (item i of
listOfCoinDenominations)
            totalUsedCoins = totalUsedCoins + 1
            currentCurrencyTallyValue = (item i of usedCoinsTallyList)
            (item i of usedCoinsTallyList) = (currentCurrencyTallyValue+ 1)
        else
            i = (i - 1)

    return {usedCoinsTallyList, totalUsedCoins}
```

Asymptotic Runtime

The bounded runtime O(nW) depends on the greater of the 2, the values of currency vs the size of the change amount W.

Dynamic Programming

Pseudo-Code

```
on changedp(listOfCoinDenominations, desiredChangeAmount)
       table[0] = (an array with all 0's and length of listOfCoinDenominations)
       table from [1] to [desiredChangeAmount + 1] = (arrays with all
desiredChangeAmount
            + 1 and length of listOfCoinDenominations)
       repeat i over length of listOfCoinDenominations
              repeat j over length of table
                     if j - listOfCoinDenominations[i] < 0
                            comp = j
                     else
                            comp = tablevalue - listOfCoinDenominations[i]
                     if sum of table[j] > sum of table[a] + 1
                            table[j] = table[a]
                            Table[j][i] = table[j][i] + 1
       return table[desiredChangeAmount]
```

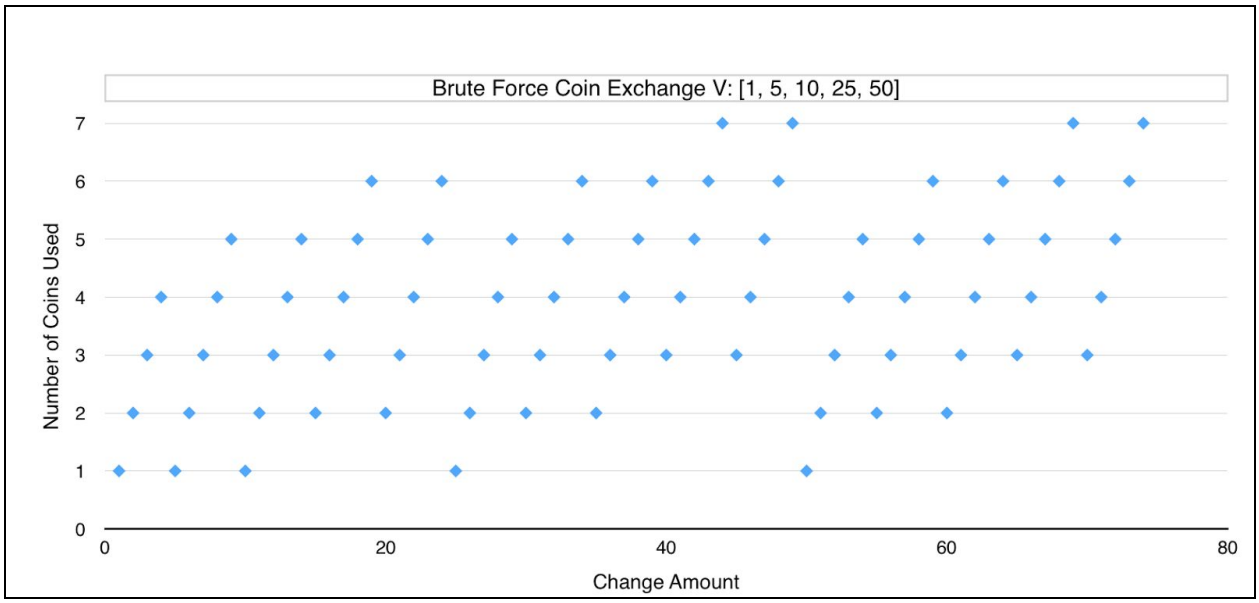
Asymptotic Runtime

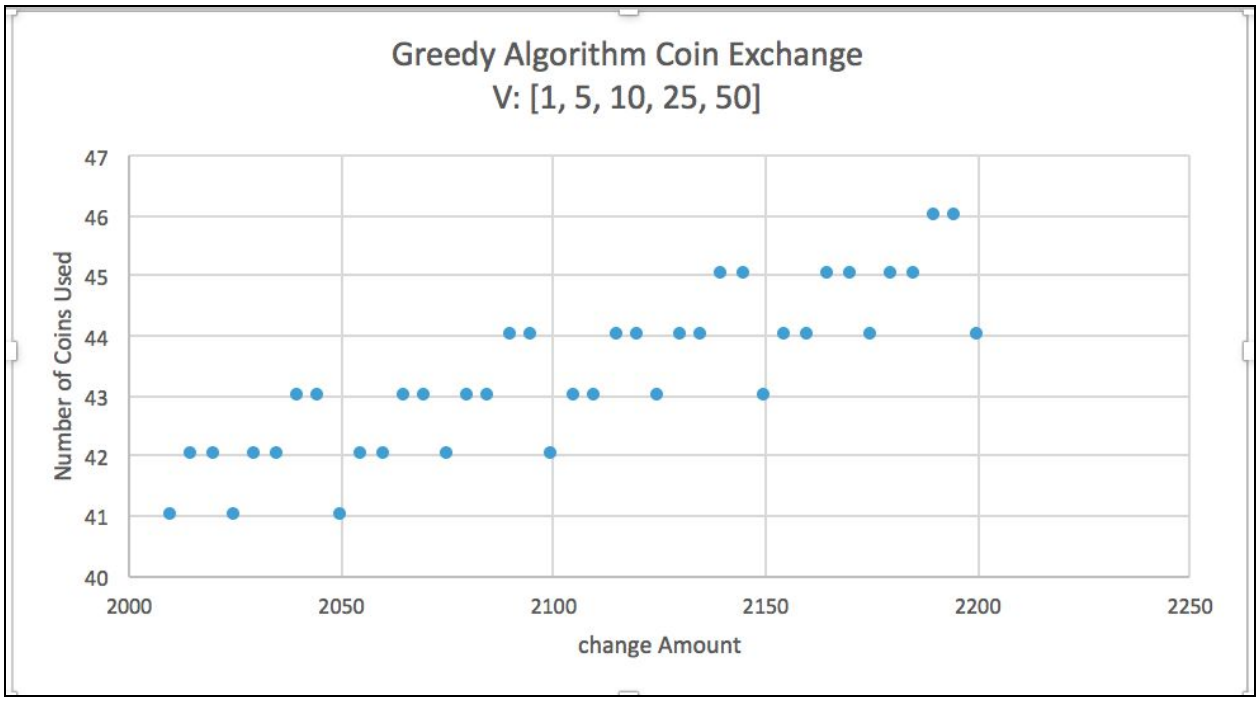
The runtime is contingent upon the function's nested loop, that is time iterating through the denomination list and through the change table. This results in a bounded runtime of O(nW).

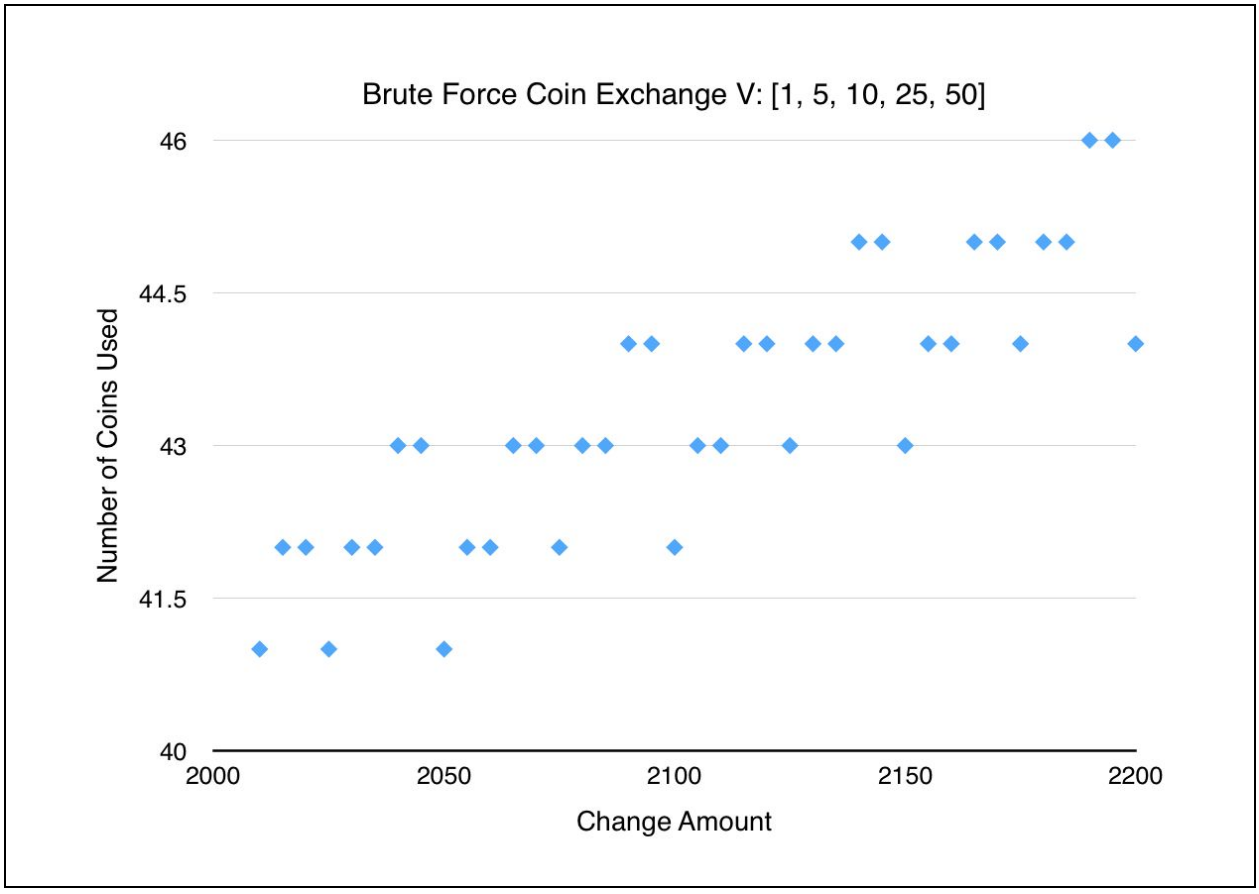
2.

The dynamic programming method utilizes a table that is looped over once for every coin provided. Each table row is set to a high value and is then compared to the row that is the coins value before it. If the value of that row plus one is less than what is currently in that row then the row is replaced with a copy of the other row while the place representing the current coin is incremented. The last row remaining is the answer. This method is valid due to only calculating small amounts once while also using them to determine the change of larger values.

3.

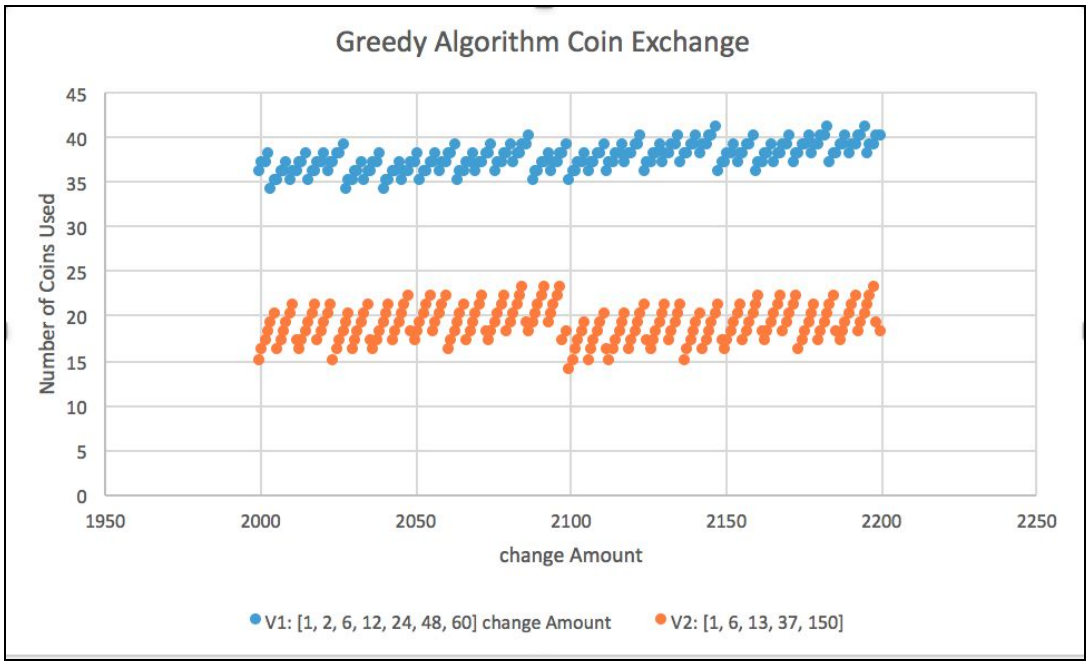

Brute Force Coin Exchange V: [1, 5, 10, 25, 50]
Number of Coins Used
0
1
2
3
4
5
6
7
0
20
40
60
80
Change Amount

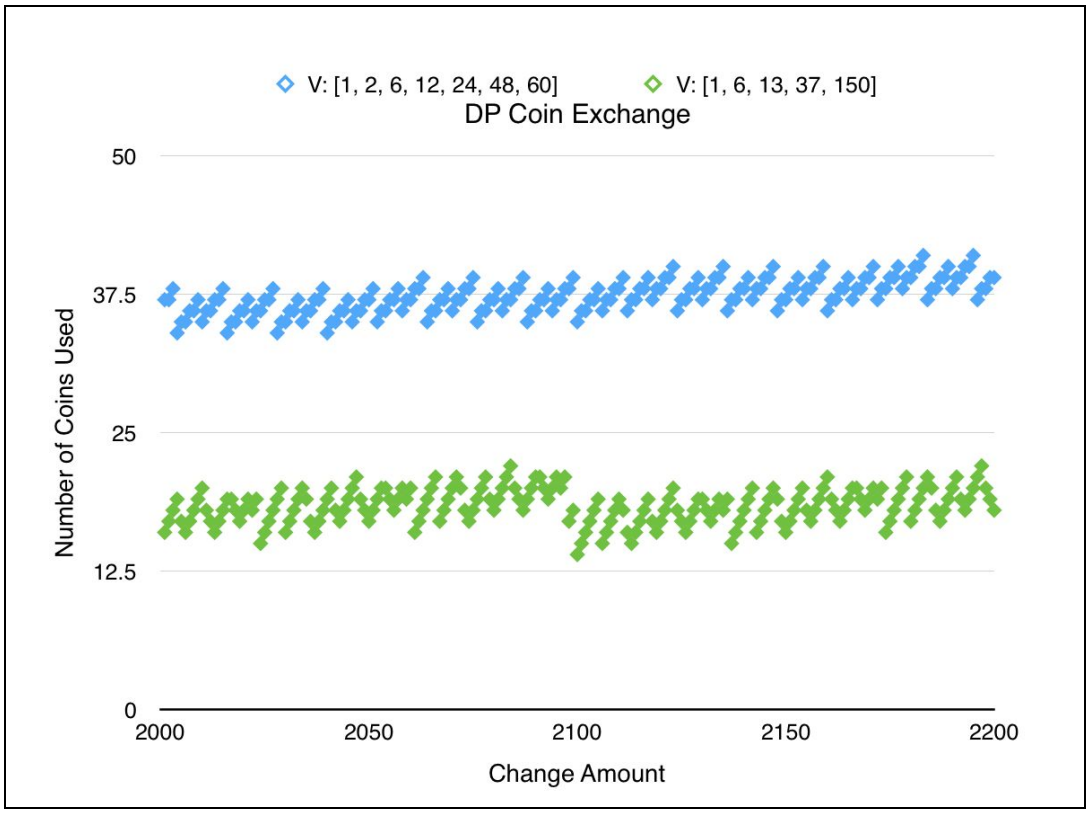

Greedy Algorithm Coin Exchange
V: [1, 5, 10, 25, 50]
Number of Coins Used
40
41
42
43
44
45
46
47
2000
2050
2100
2150
2200
2250
change Amount



Each of these algorithms arrive at the same answer. While this may not seem apparent with the brute force algorithm, this is only due to the use of different inputs. The brute force algorithm uses a smaller range of numbers due to it running dramatically slower than the other algorithms. Despite this, by analyzing the graph, one can see that the pattern created matches the growth pattern of the other algorithms.

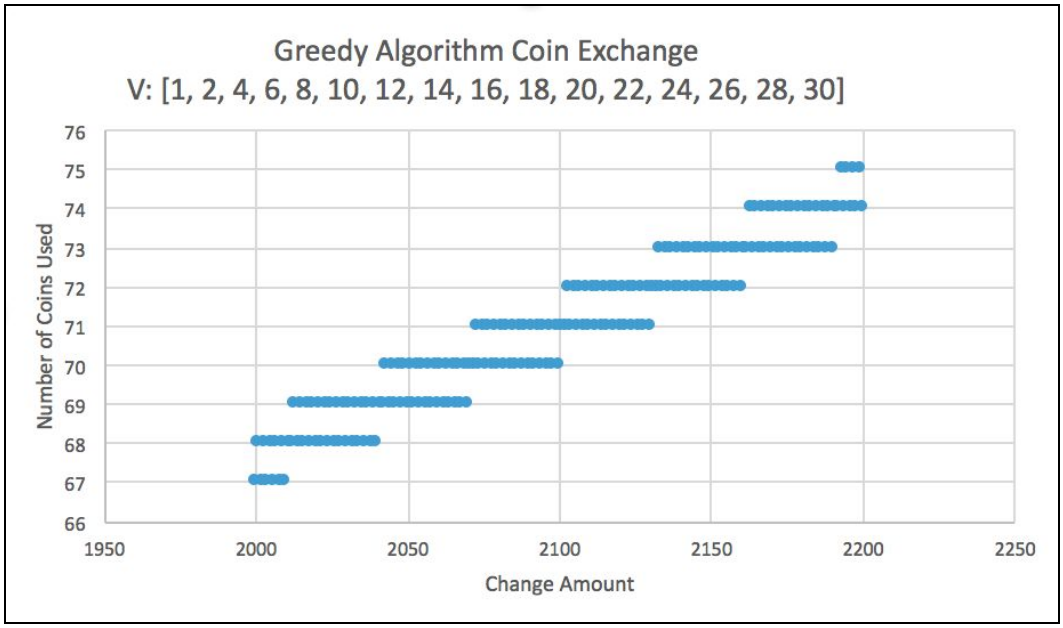
4.

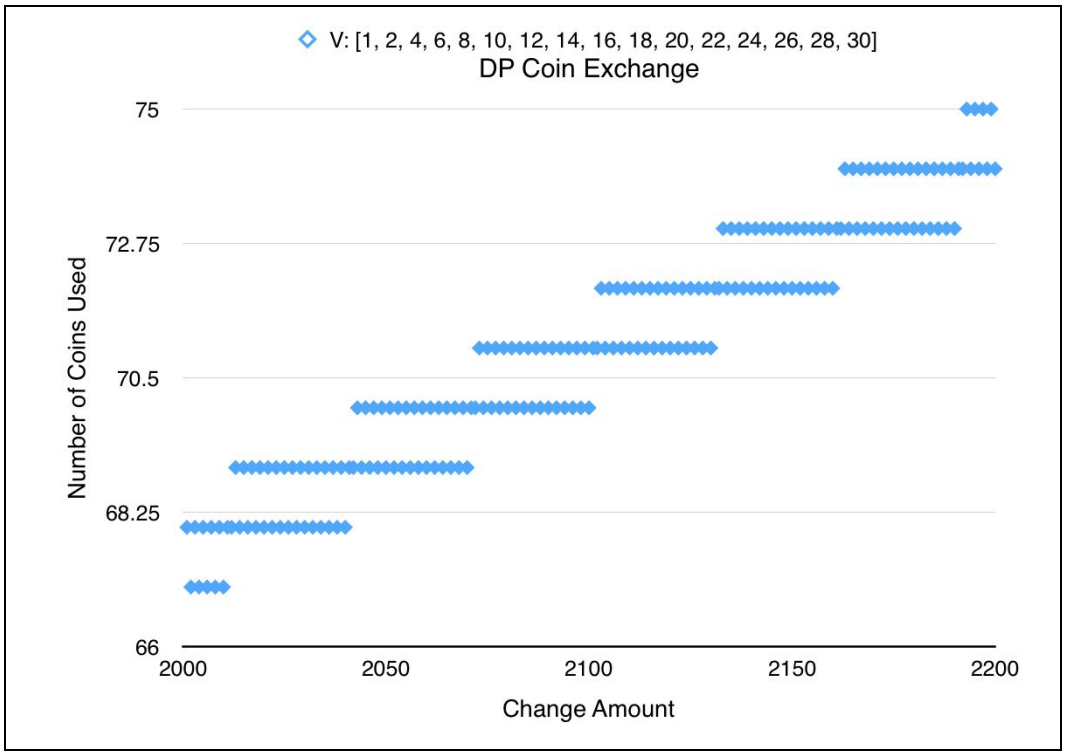

Greedy Algorithm Coin Exchange
Number of Coins Used
45
40
35
30
25
20
15
10
5
0
1950
2000
2050
2100
2150
2200
2250
change Amount
V1: [1, 2, 6, 12, 24, 48, 60] change Amount
V2: [1, 6, 13, 37, 150]


V: [1, 2, 6, 12, 24, 48, 60]
V: [1, 6, 13, 37, 150]
DP Coin Exchange
Number of Coins Used
50
37.5
25
12.5
0
2000
2050
2100
2150
2200
Change Amount

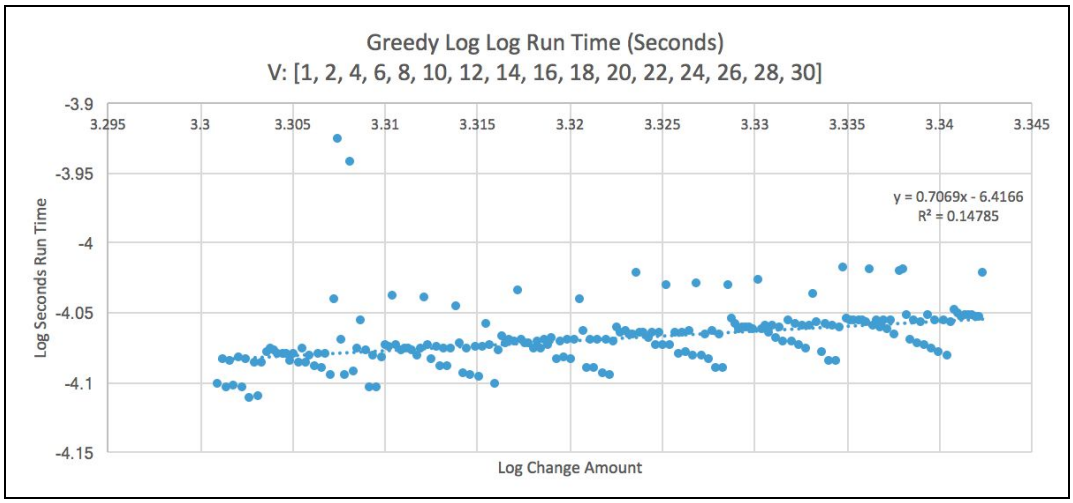
Based on analyzing the graphs above, the greedy algorithm yields results similar to the dynamic programming algorithm when comparing the number of coins used. However, the dynamic programming algorithm has a lower average number of coins used. It's this property that shows the dynamic programming algorithm as more superior.

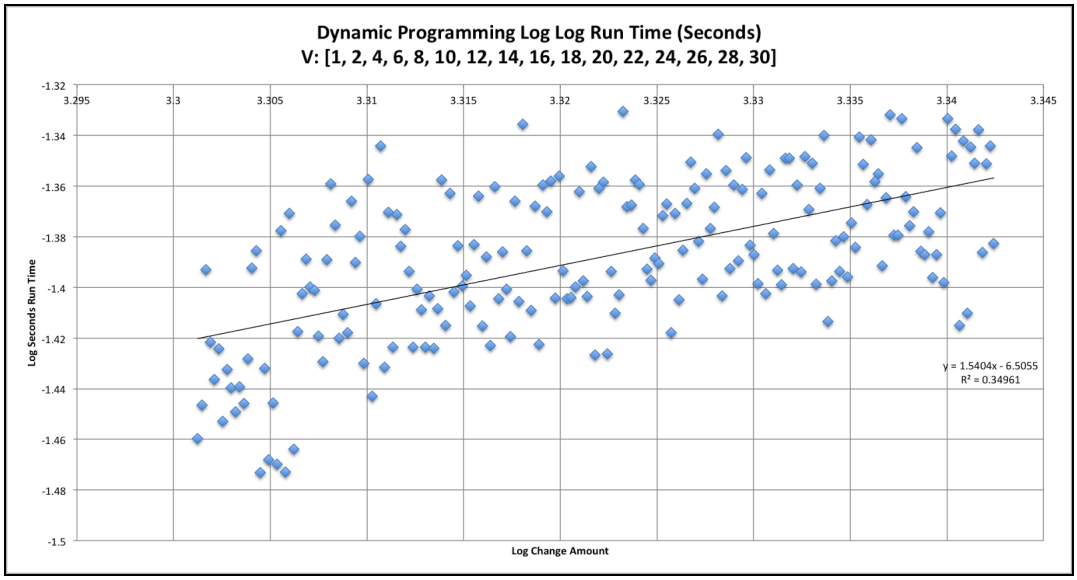
5.





6.

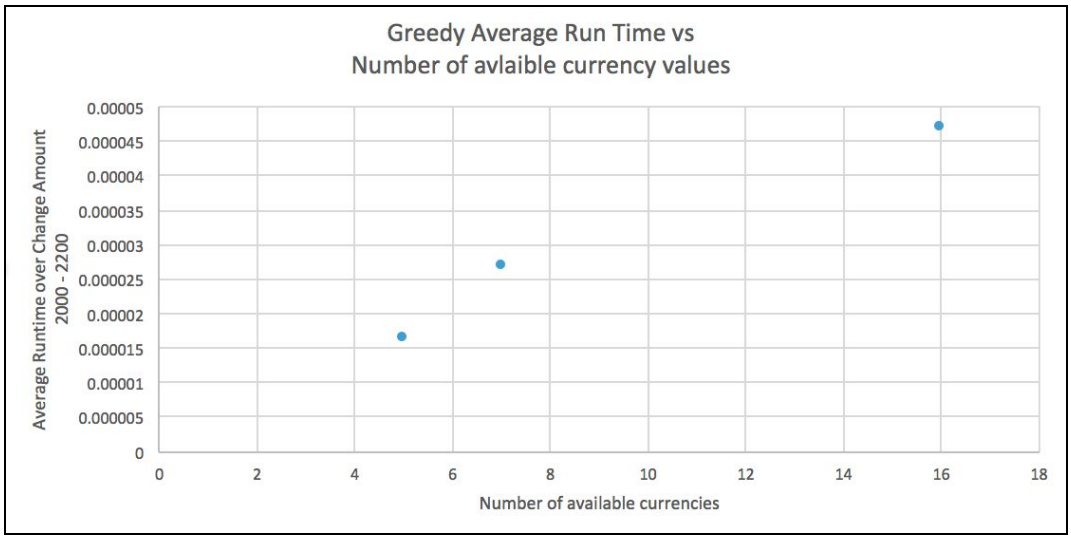


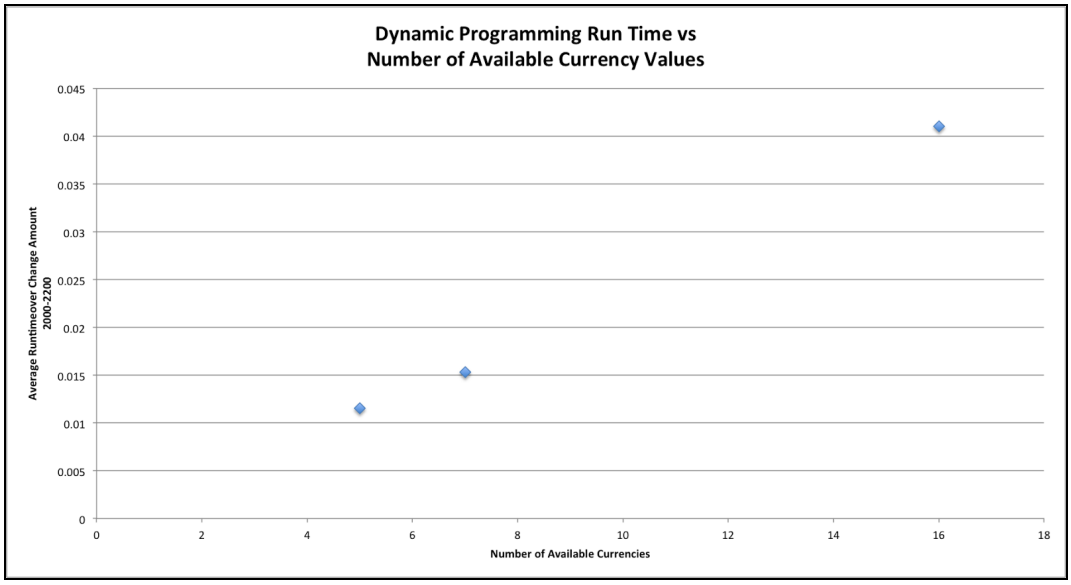


In the greedy algorithm, the runtime shows a weak R-squared value. This is due to the nature of selecting from a group of possible currency values by starting with the greatest possible. The log log plot suggests a near linear runtime in the above situation.

In the dynamic programming algorithm, the runtime shows a similar weak R-squared value to the greedy algorithm The log log plot suggests either a linear and exponential runtime.

7.

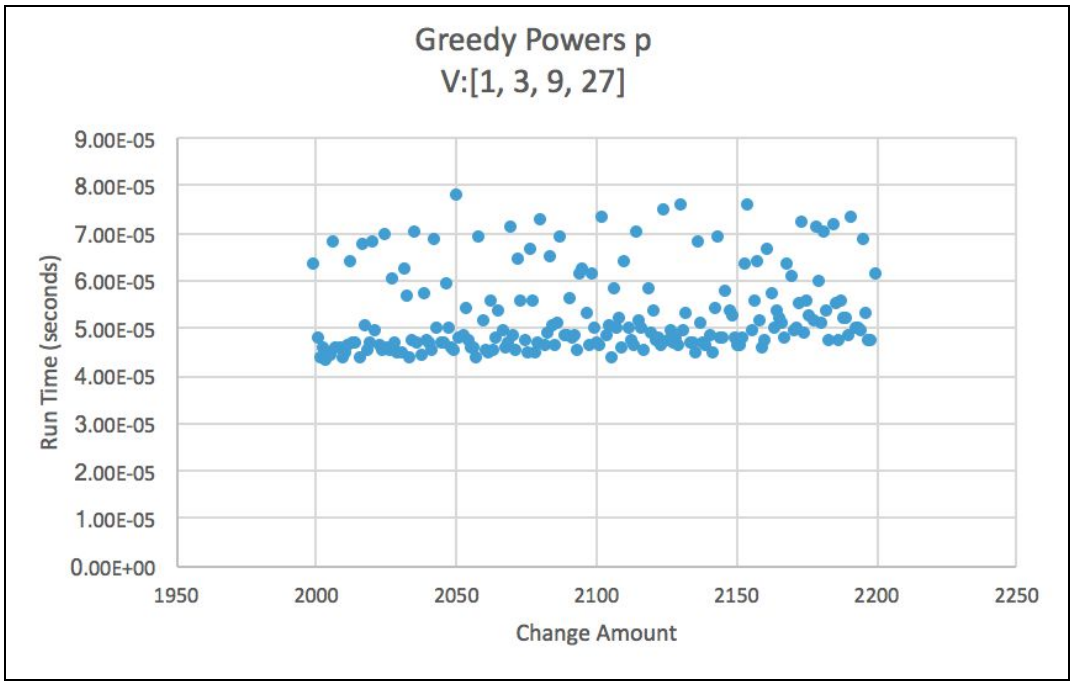


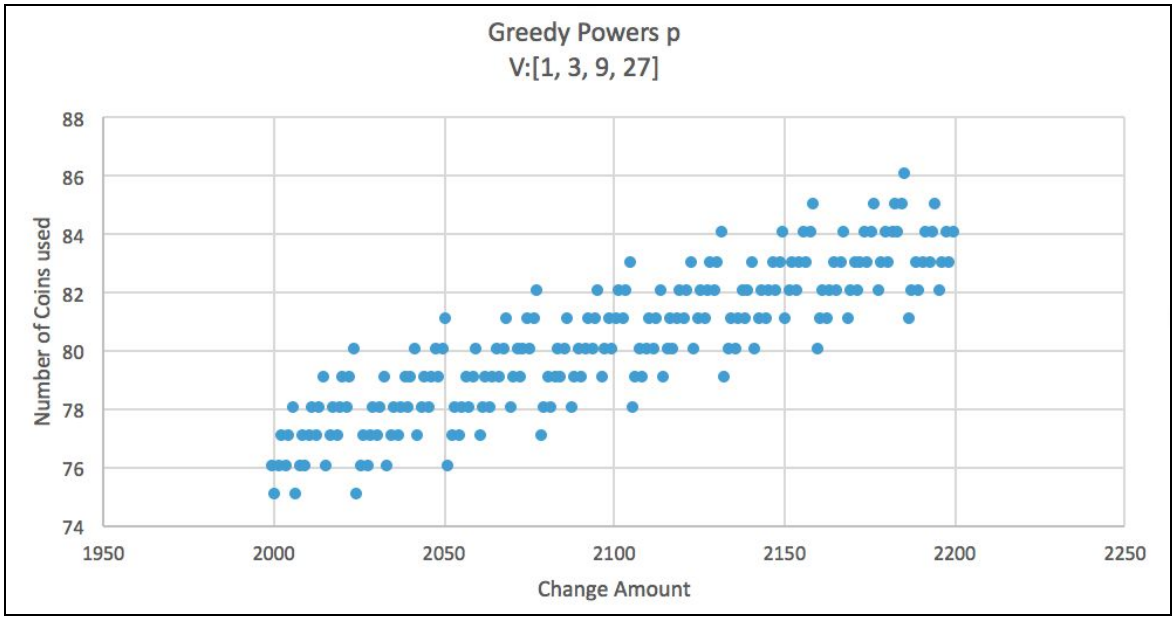


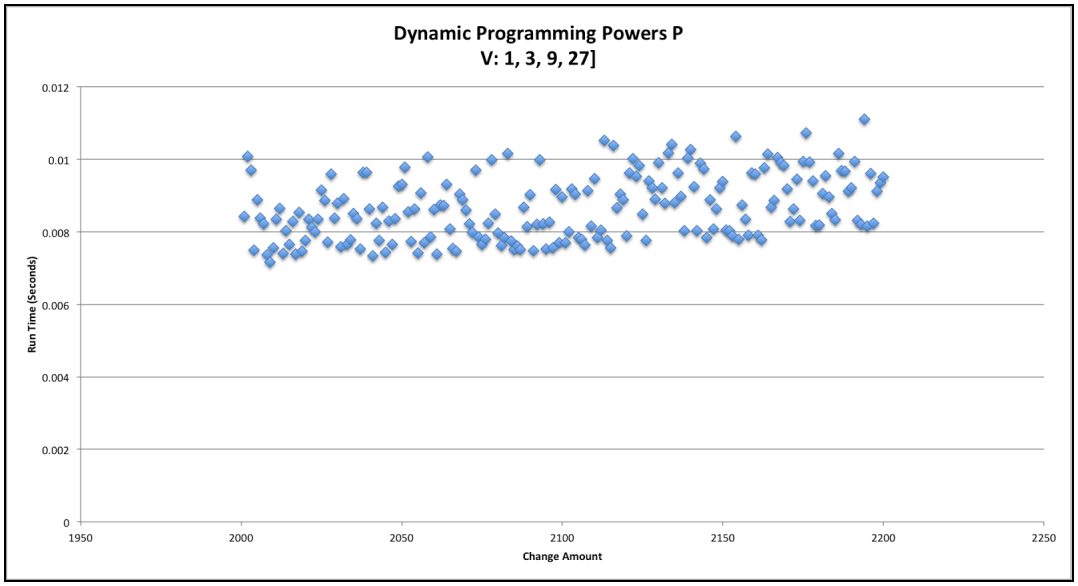
When utilizing the greedy algorithm, the average run time increases as larger numbers of available currencies increases. This in line with the theoretical runtime analysis for greedy based algorithms.

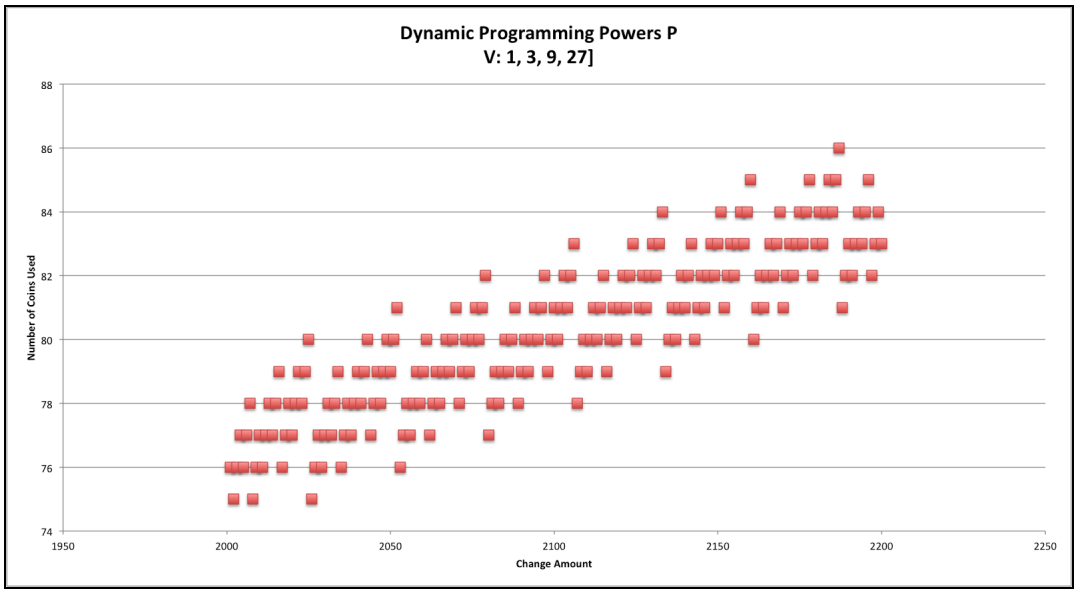
The dynamic programming algorithm functions in a similar pattern to the greedy algorithm; the average runtime increases with the number of available currencies.

8.


Greedy Powers p
V:[1, 3, 9, 27]
Run Time (seconds)
9.00E-05
8.00E-05
7.00E-05
6.00E-05
5.00E-05
4.00E-05
3.00E-05
2.00E-05
1.00E-05
0.00E+00
1950
2000
2050
2100
2150
2200
2250
Change Amount


Greedy Powers p
V:[1, 3, 9, 27]
Number of Coins used
88
86
84
82
80
78
76
74
1950
2000
2050
2100
2150
2200
2250
Change Amount





The greedy algorithm yields a scattered plot when values of currency are powers p [1, 3, 9, 27]. From this, one sees that the number of coins used ranges widely in a small range of desired change amount.

By analyzing the above graphs, the greedy and dynamic programming algorithms run approximately the same in the given circumstances.

9.

Any denomination set with one or more large disparities (cliffs) between denomination values benefit most from a "greedy" solution. Analogous to identifying when best to implement weighted edges, *e.g.,* Dijkstra's Shortest Path, when a comparative "shorter path" emerges, don't waste cycles: *take that path*. Similarly, where it is clear that a substantial number of cycles can be saved—and here our "edges," *i.e.,* denominations come to us already defined—then save the weighing of "best approaches" & implement "efficiency-first."