Group 7 - Project 2

Team Members

Justin Kruse, Nicholas Koller, Jon Larsen

Algorithm 1 - Change Slow (Divide And Conquer)

<u>Pseudocode</u>

```
Helper Function
  1.   The change helper function is the primary recursion function
  2.   Base case if the sum of coins used so far matches the target
       2.1.   Return that list
  3.   If sum is greater than the target
       3.1.   Pass because it's gone too far.
  4.   If denomination list is empty
       4.1.   Also pass this is also necessary for finding different coin combinations
  5.   Else
       5.1.   Run first recursion on all adding the first coin available to the passed coin list
              5.1.1.   Allows multiples of the same coin
       5.2.   Second recursion we remove the first available coin
              5.2.1.   Allows a variety of coin combinations
Driver Function
       1.   Create a list of solutions using the helper function
       2.   Pick the solution from the list that has the smallest length
       3.   Create an array with same length as the parameter array of coin values
       4.   For each element of the coin denomination array,
            4.1.   assigning the count of each coin from the optimal solution.
```

Analysis:

The main function iterates through the solution array of n coin values once, and the helper function has up to two recursive calls per solution, each doing n-1 amount of the work.

Asymptotic Running Time:

$T(n) = n * 2T(n-1) + c$

Using the Muster Theorem: $a= 2$, $b = 1$, $f(n) = c \Rightarrow d = 0$

$a>1 \Rightarrow O(n^d a^{n/b}) = n^0 2^{n/1}$

$= n * O(2^n) = O(n2^n)$

Algorithm 2 - Change Greedy

<u>Pseudocode</u>

```
  1. Create an coin used array the same length as the array of coin denominations
  2. Set the number of coins used to zero
```

```
3. Iterate through the indices of the array of coin denominations in reverse order to remove
   largest coin first (assumes denominations are sorted smallest to largest)
    a. While the total change minus the denomination at that index is greater than or
       equal to zero
        i.   Decrement the total change by the denomination
        ii.  Increment the total number of coins used by one
        iii. Increment the value at corresponding index of the to show how many of a
             certain denomination coin is used
4. Return the coin used array and the number of coins used
```

Analysis:
Assuming the denominations are sorted, the function runs through the entire contents of the denomination array, of size n, once. The inner while-loop will run at most n times.

Asymptotic Running Time:
$T(n) = T(n) * T(n) + c$
$= O(n^2)$

Algorithm 3 - Change DP (Dynamic Programming)

Pseudocode

```
1.  Create 2 arrays size of the target + 1 populated with largest possible value (large array)
    the other is populated with -1 (small array).
2.  Then Setting the first position to zero in the large array and the corresponding position to
    -1 in the small array
3.  Loop through the coin array
    3.1.  Loop through the target table
          3.1.1.  We want to compare the minimum between large array position value
                  (current number of coins to calculate) and the lowest number of coins
                  needed to calculate the target value minus the current coin value. Plus 1
                  for the current coin
                  3.1.1.1.  We will update the Large array when we discover a lower
                            combination of previous values plus 1
                  3.1.1.2.  The small array is used to track the last position of the coin what
                            used for that target.
4.  To find the optimal solution we looped bottom up through our table
5.  Setting start equal to the length of the small table - 1
6.  When start is not 0
    6.1.  Set j to that small array value at the start position.
    6.2.  Append that coin value to optimal solution
    6.3.  Decrement start by that coin value
7.  For each element of the coin denomination array,
```

    7.1. We assign the count of each coin from the optimal solution.
8. We return those coin counts and optimal solution number

Analysis:
The program iterates through a loop the size of the coin array, n. Within that loop there is an inner loop the size of the large array, which is the target value for the change +1, which we will shorten to “m”.

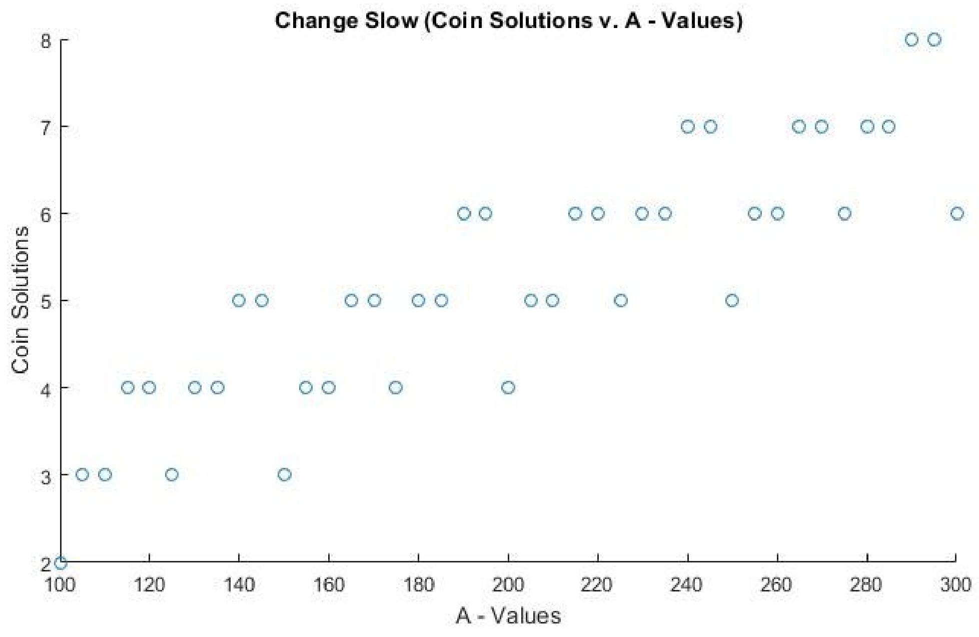
Asymptotic Running Time:
O(nm)

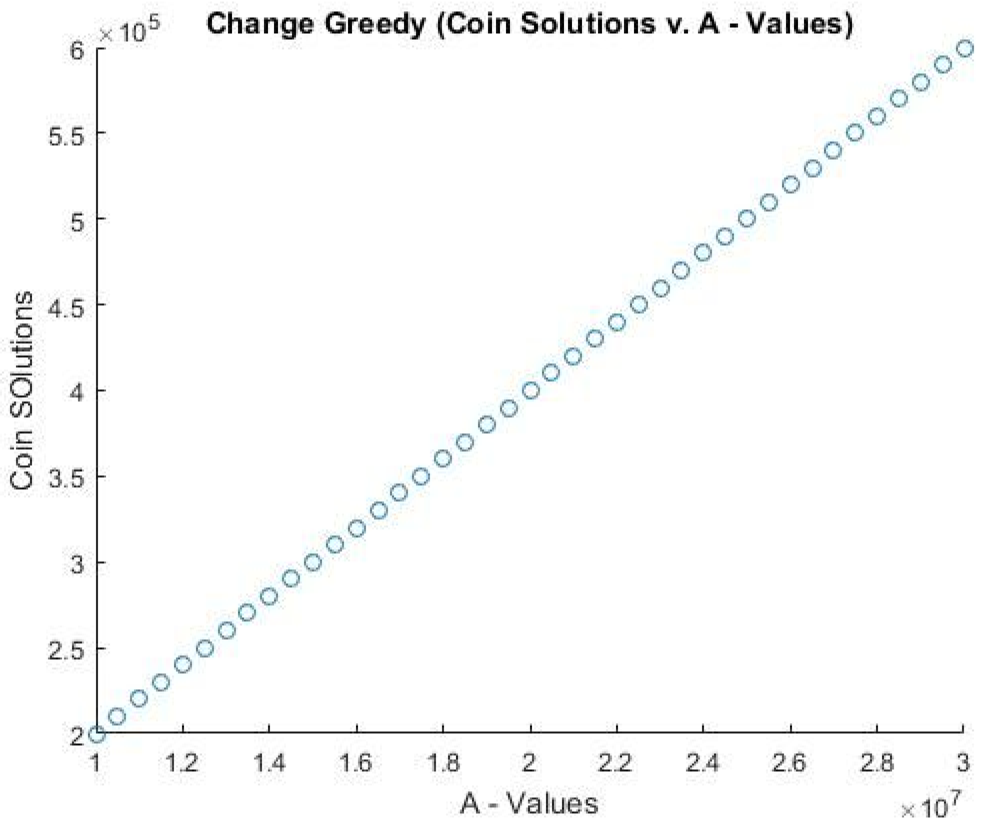
2. To fill our dynamic programming change table, we created 2 arrays size of our target + 1 and then populated the first with the largest possible value (large array). The second is populated with -1 which will track our coin positions (small array). We set the first position to zero in the large array and the corresponding position to -1 in the small array since no coin will have the value zero and thus those positions will always be zero and no corresponding coin. We use two loops the first goes through the coin array and the second goes through the large array. Our first check insures that our current target i is greater than the coin we are checking. We then compare the minimum between large array position value (current number of coins needed to calculate that target) and the lowest number of coins needed to calculate the target value minus the current coin value. We have to add 1 for the current coin and will update the large array position when we discover a lower combination of previous values plus 1. The small array is used to track the last position of the coin what used for that target.

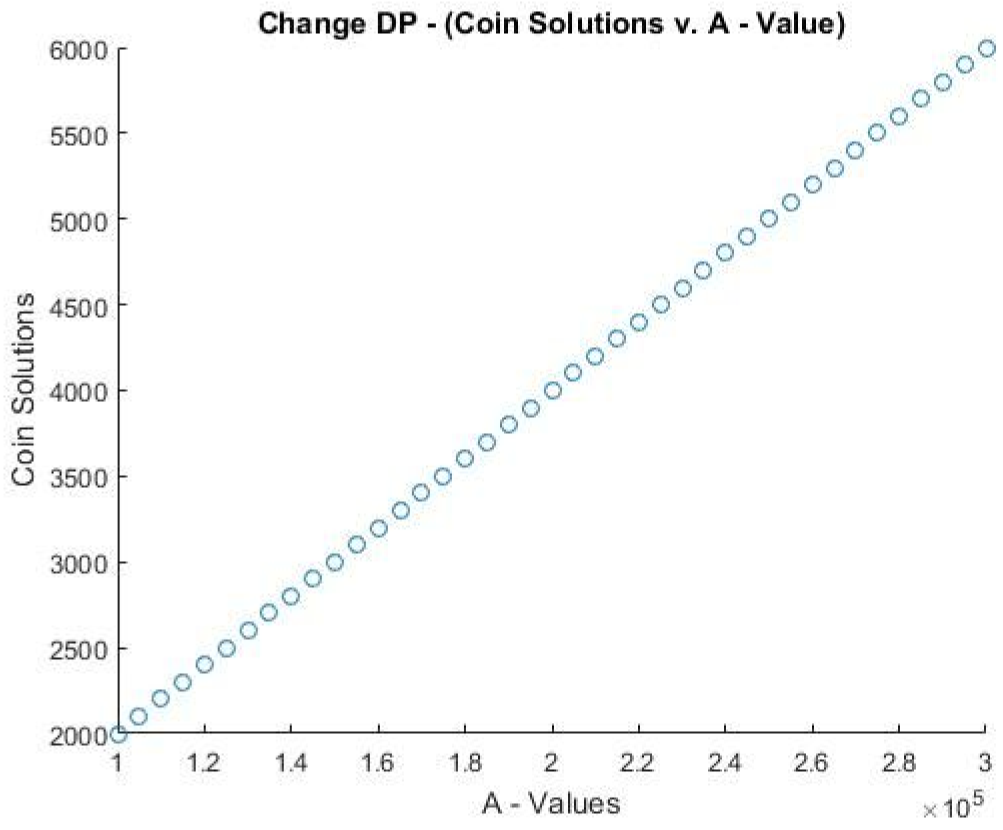
This is valid way to fill the table as it ensures that these solutions are the lowest since it compares the largest possible number against the amount of coins used to produce that. This guarantees that there will be at least 1 solution. Because it is a dynamic, it uses previous calculated values to determine the current one and it only updates the values when a smaller combination has been discovered. By checking each coin against table values we can update and calculate the lowest combination of coins to create the target.

3. The greedy and dynamic solutions have steady increasing amounts of coins used, whereas the brute-force approach has more noticeable variation (though this is more likely because it had to be run at smaller values for A) - the number of coins dips when it reaches a point where it can use multiples of the highest-denomination coins. More importantly, the greedy algorithm closely matches the optimal.

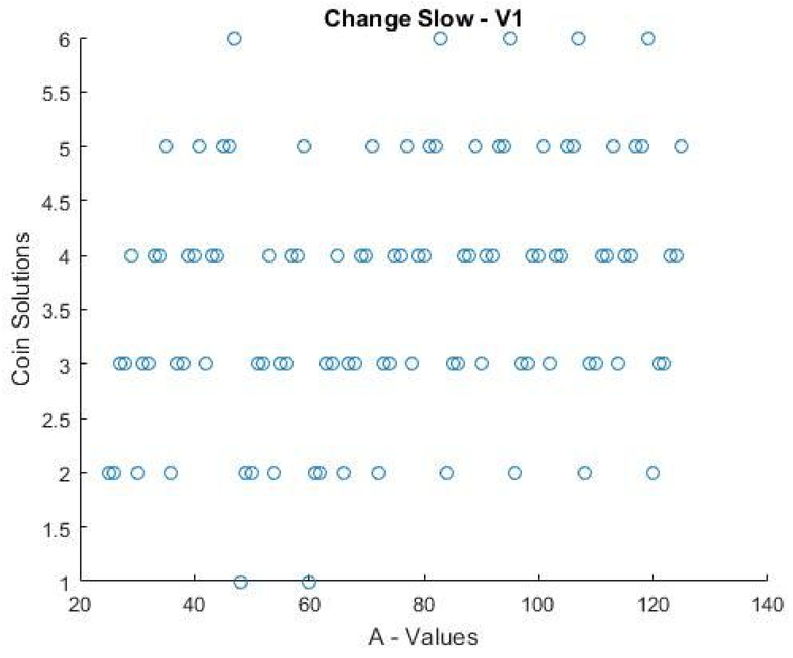
Group 7 - Project 2



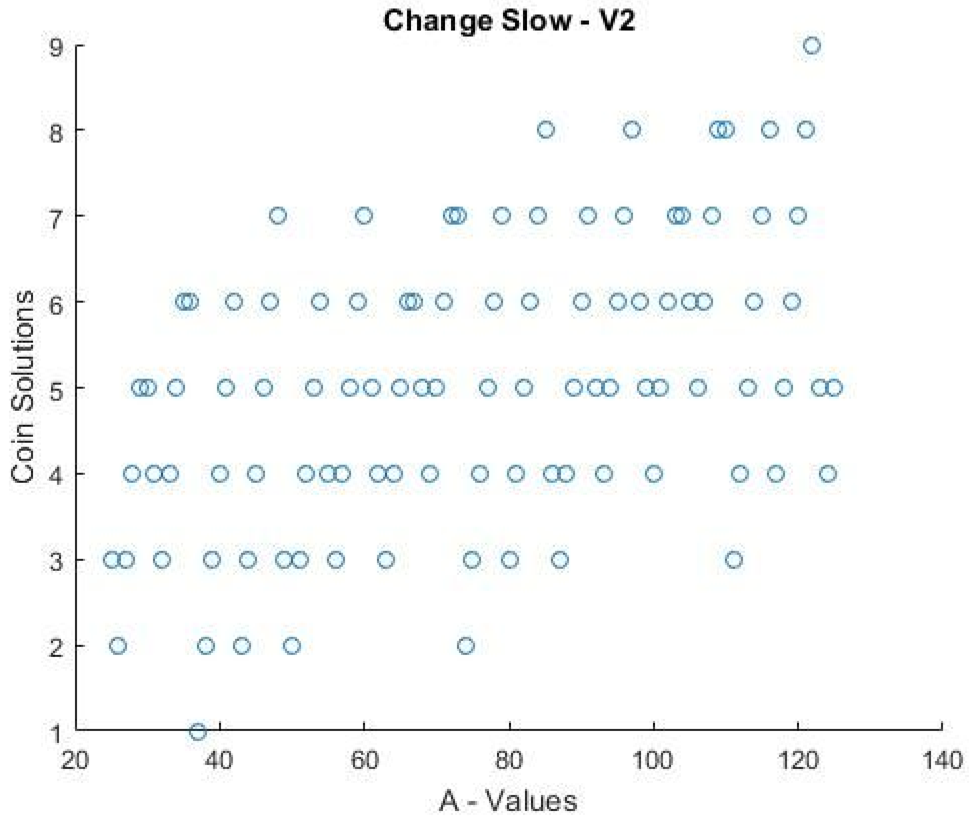


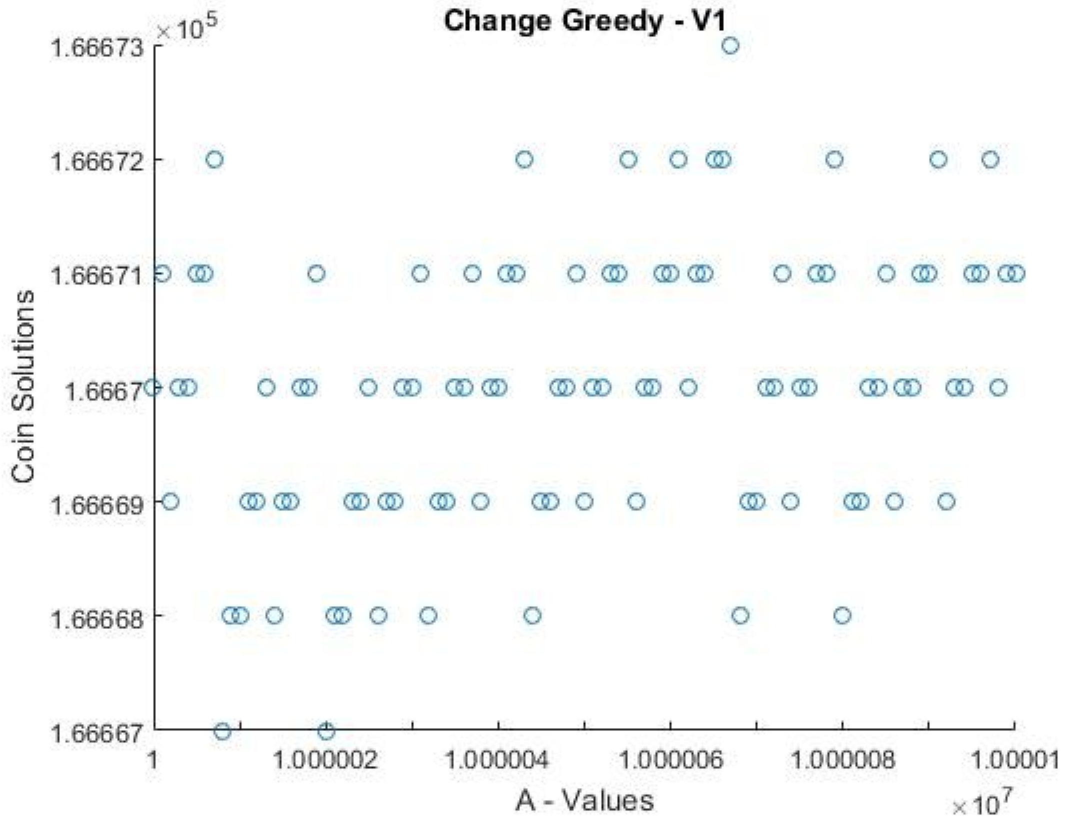


4. In both versions the greedy algorithm is very close to the dynamic one, though the greedy version has more variation

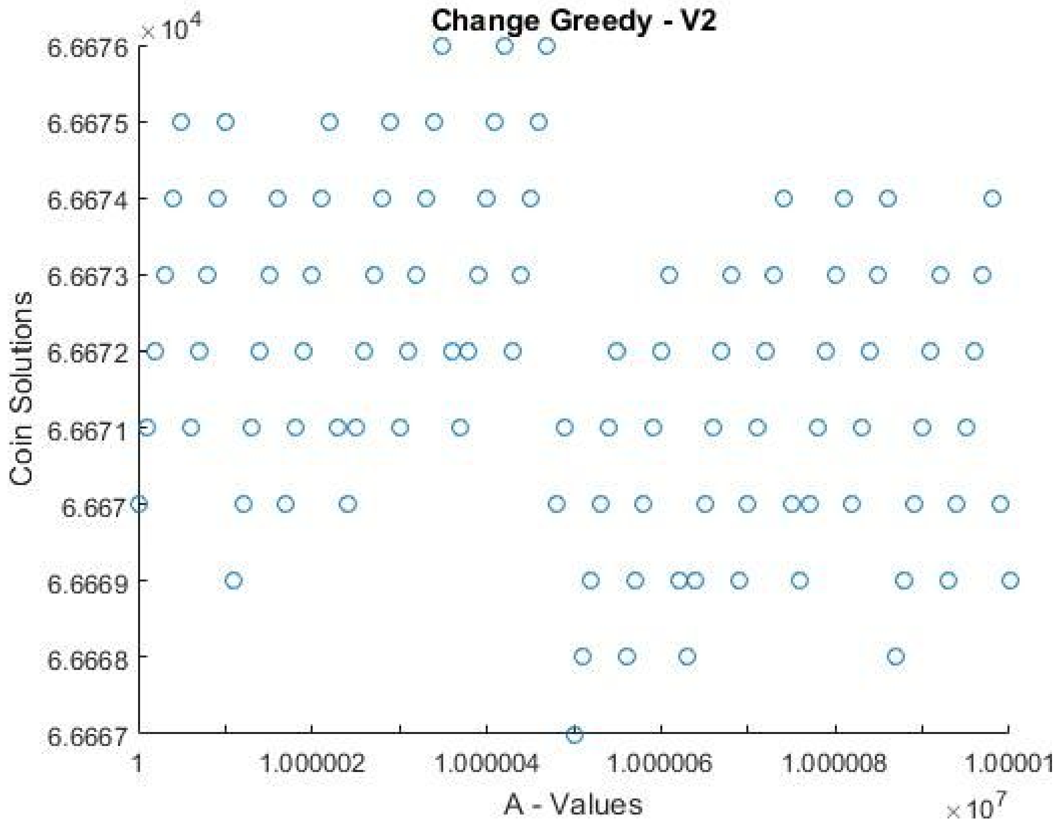


Group 7 - Project 2

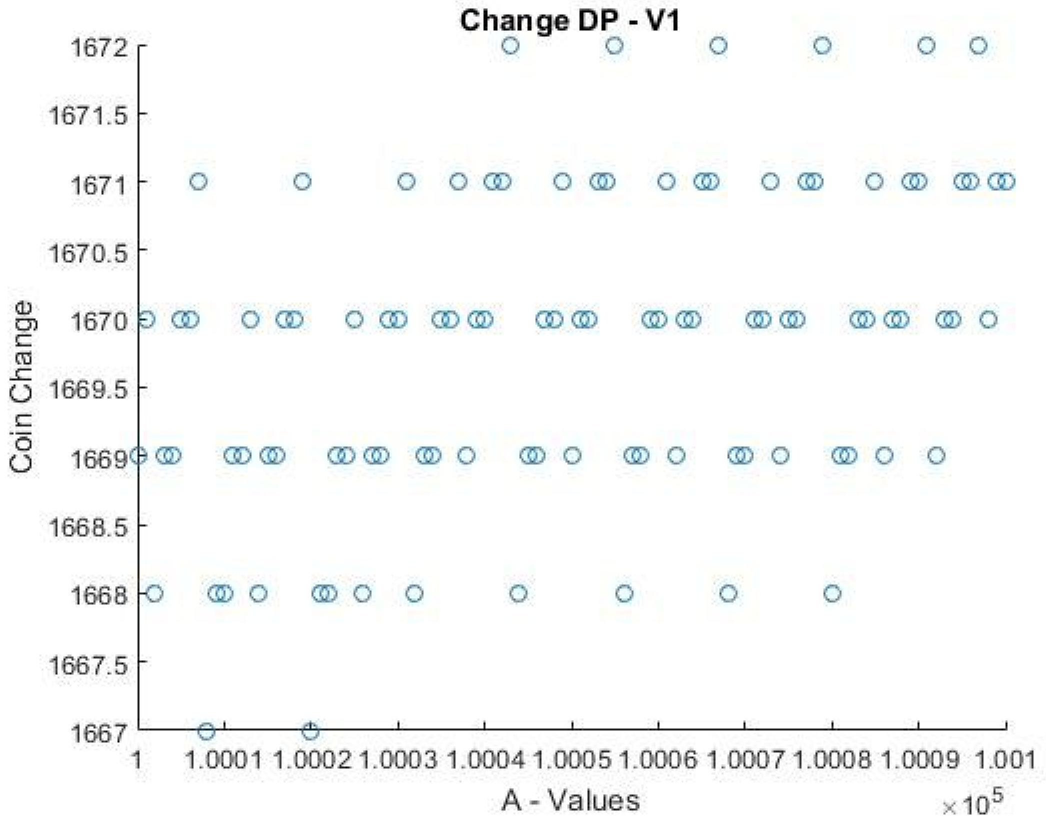


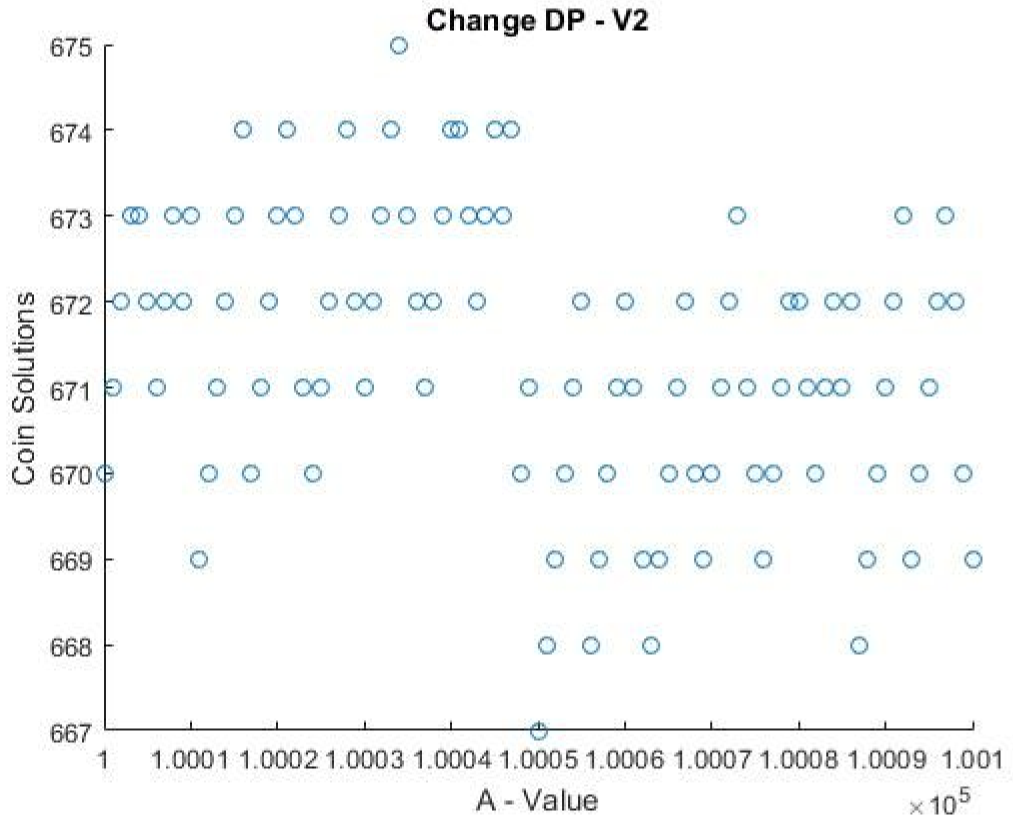


Group 7 - Project 2



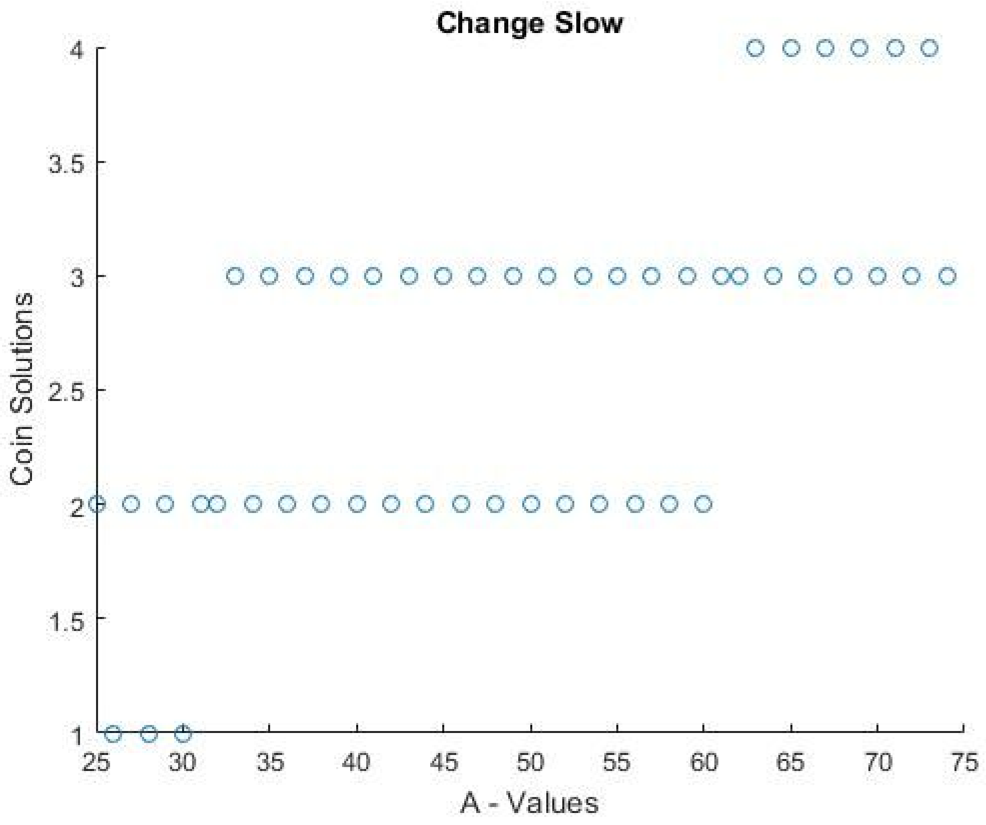
Group 7 - Project 2



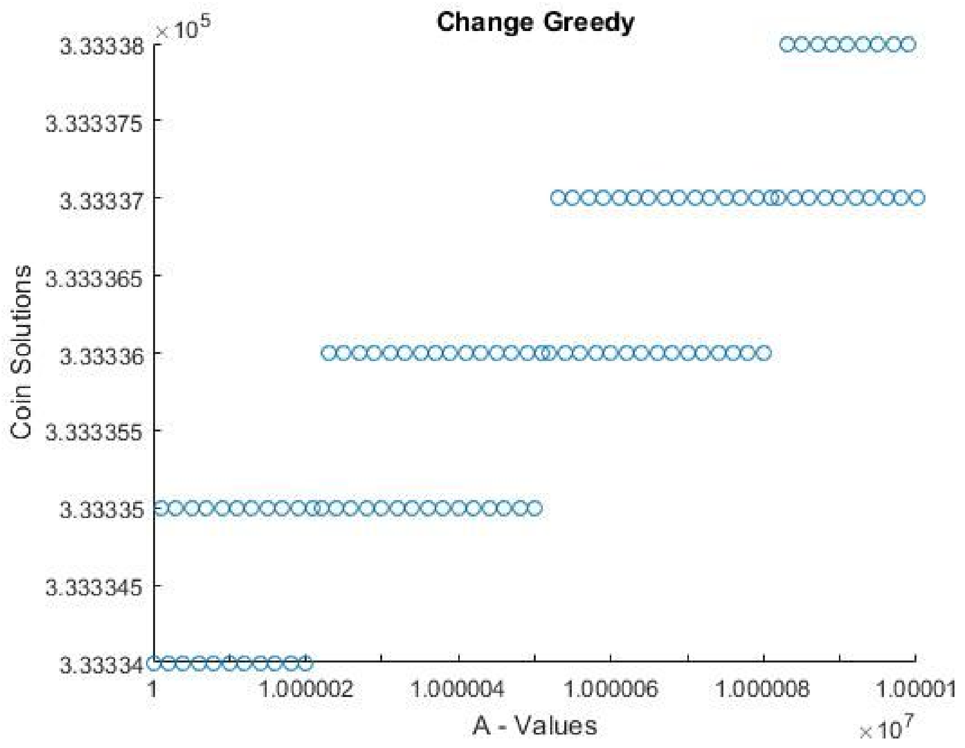


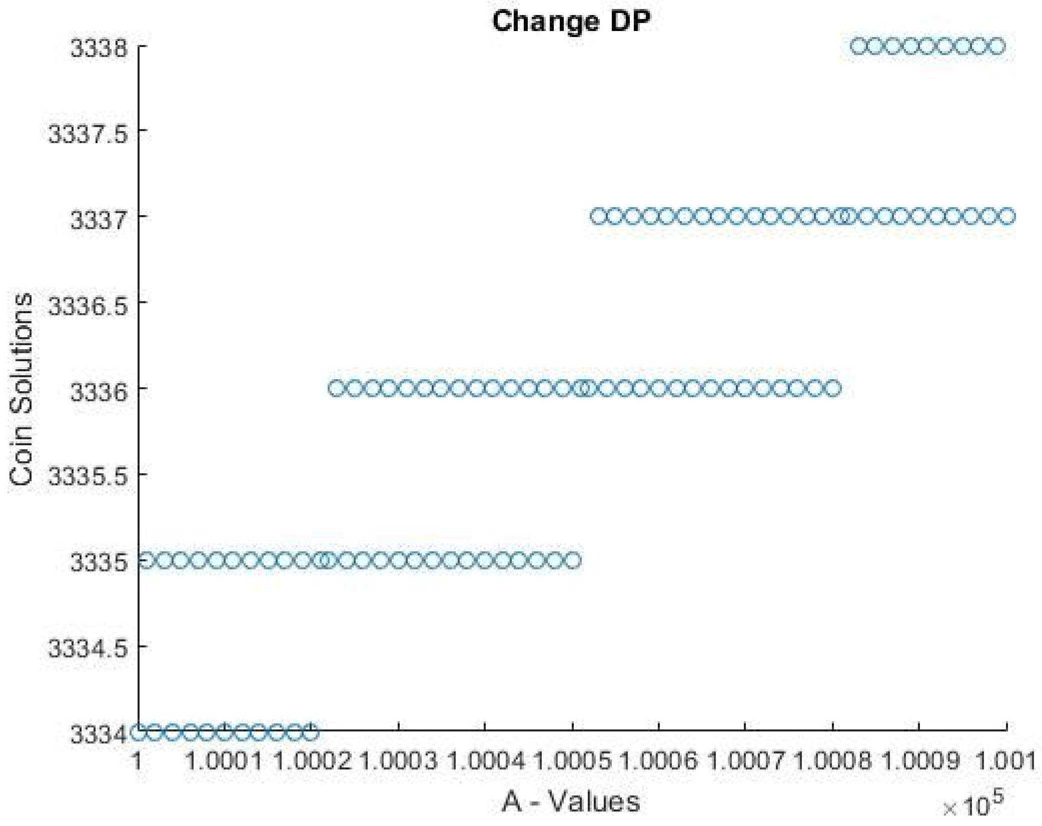
Group 7 - Project 2

5. Again we the greedy and dynamic approaches are close, but there is a tendency for the greedy version to require more coins than the dynamic one.



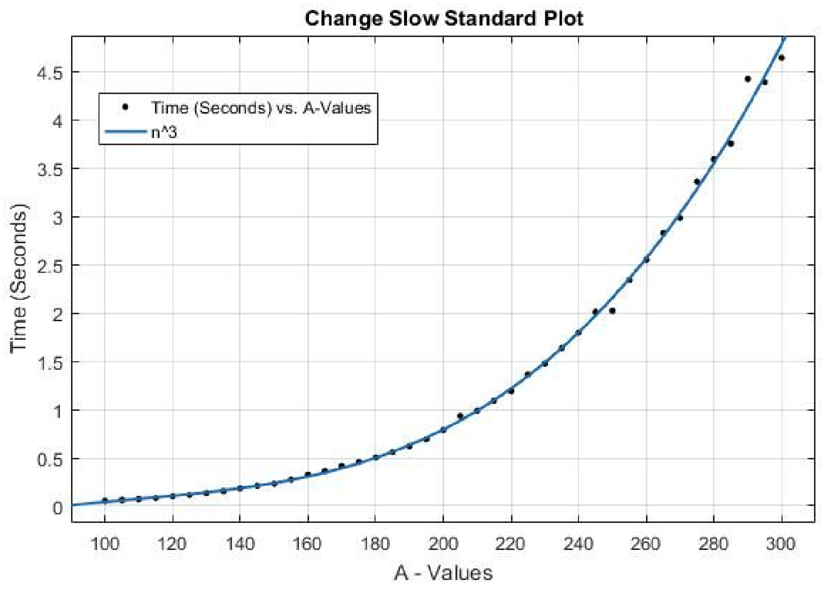
Group 7 - Project 2

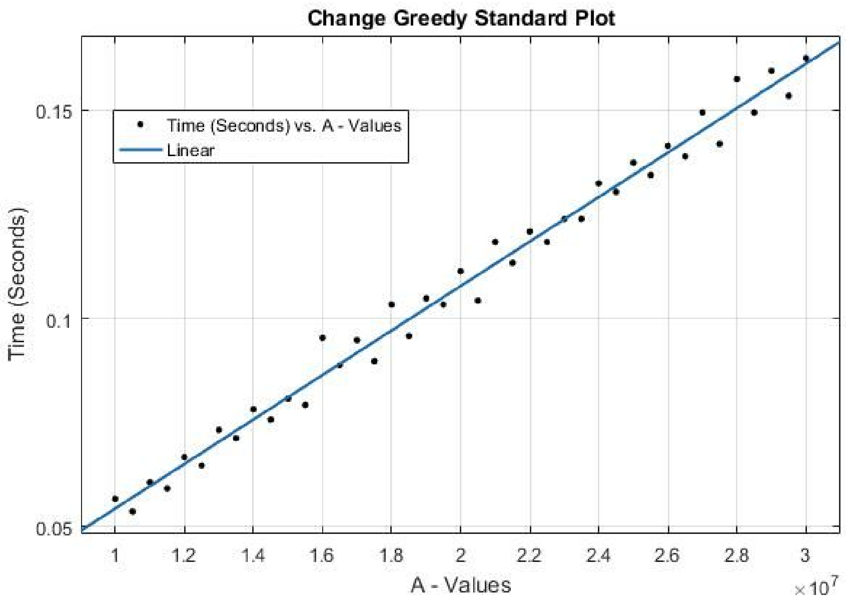




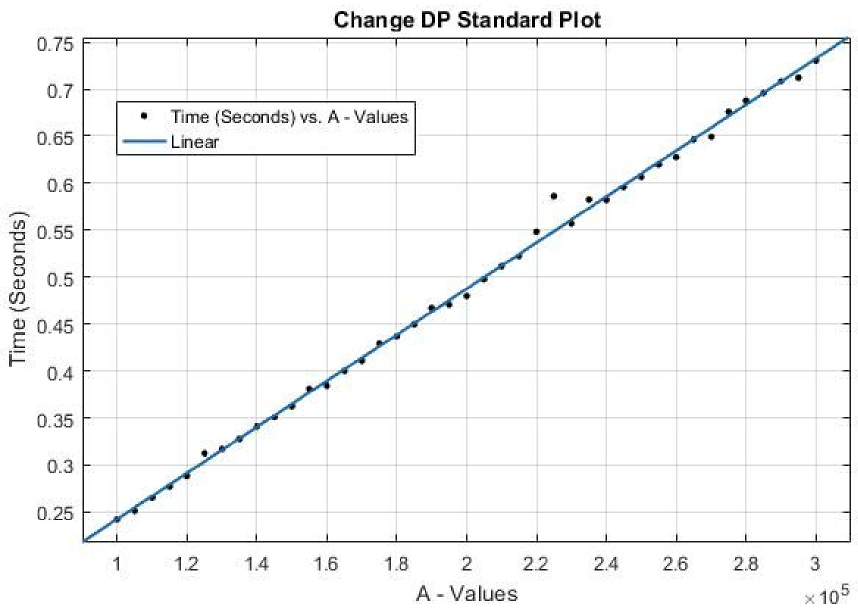
6. The brute-force method, as expected, increases exponentially very early on. The greedy approach and dynamic approach increase in a seemingly more linear fashion, but with the line

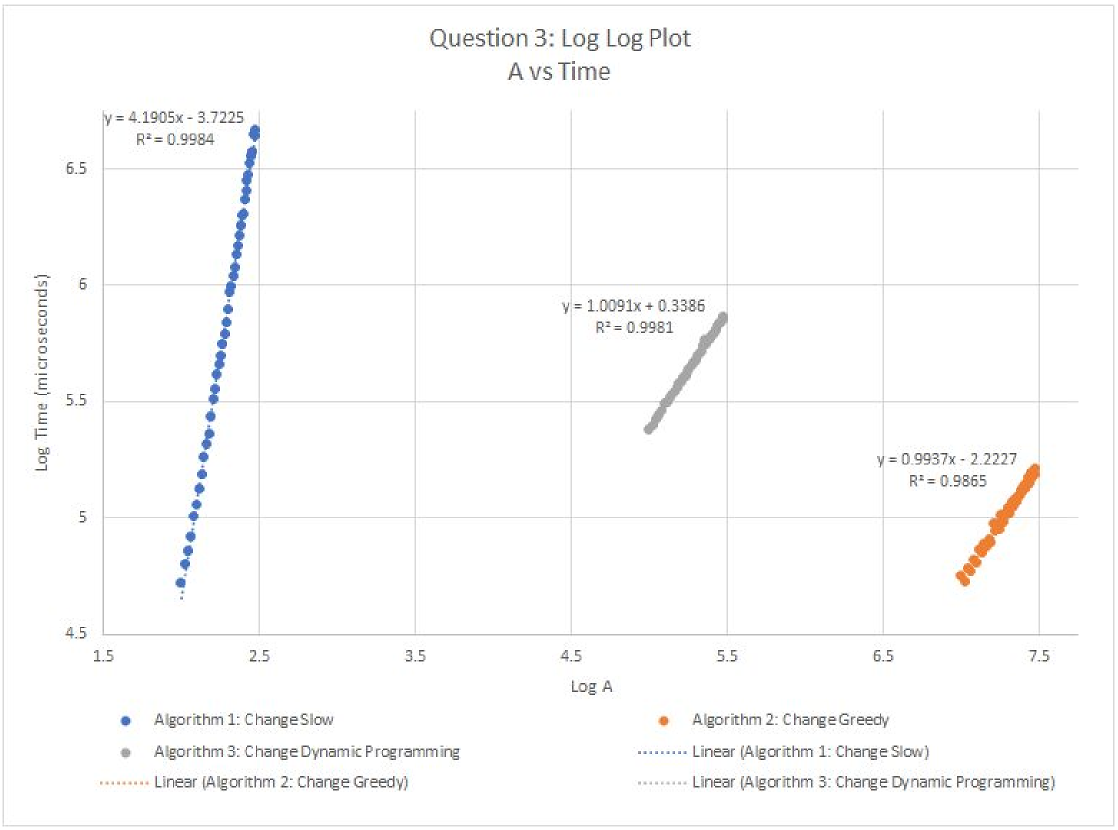
for the greedy version being far to the right of the dynamic one, indicating a tradeoff of accuracy in favor of speed for the greedy algorithm.



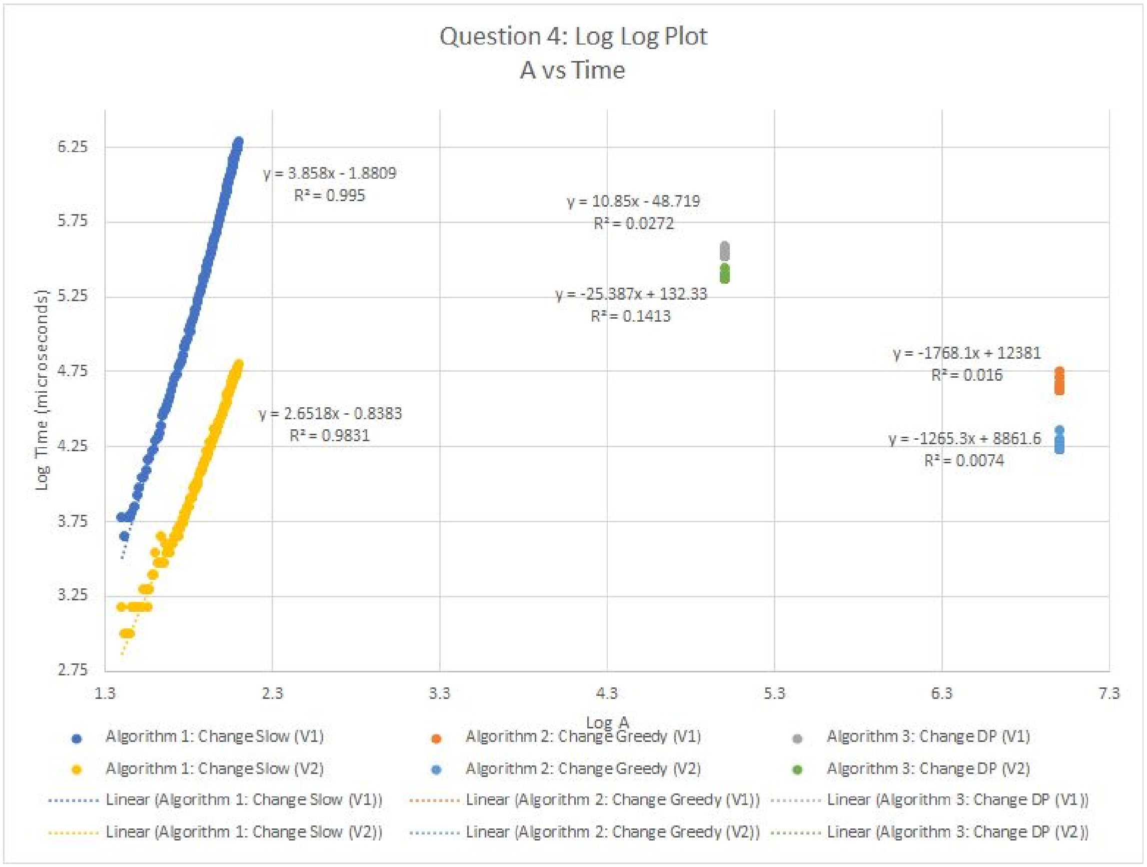


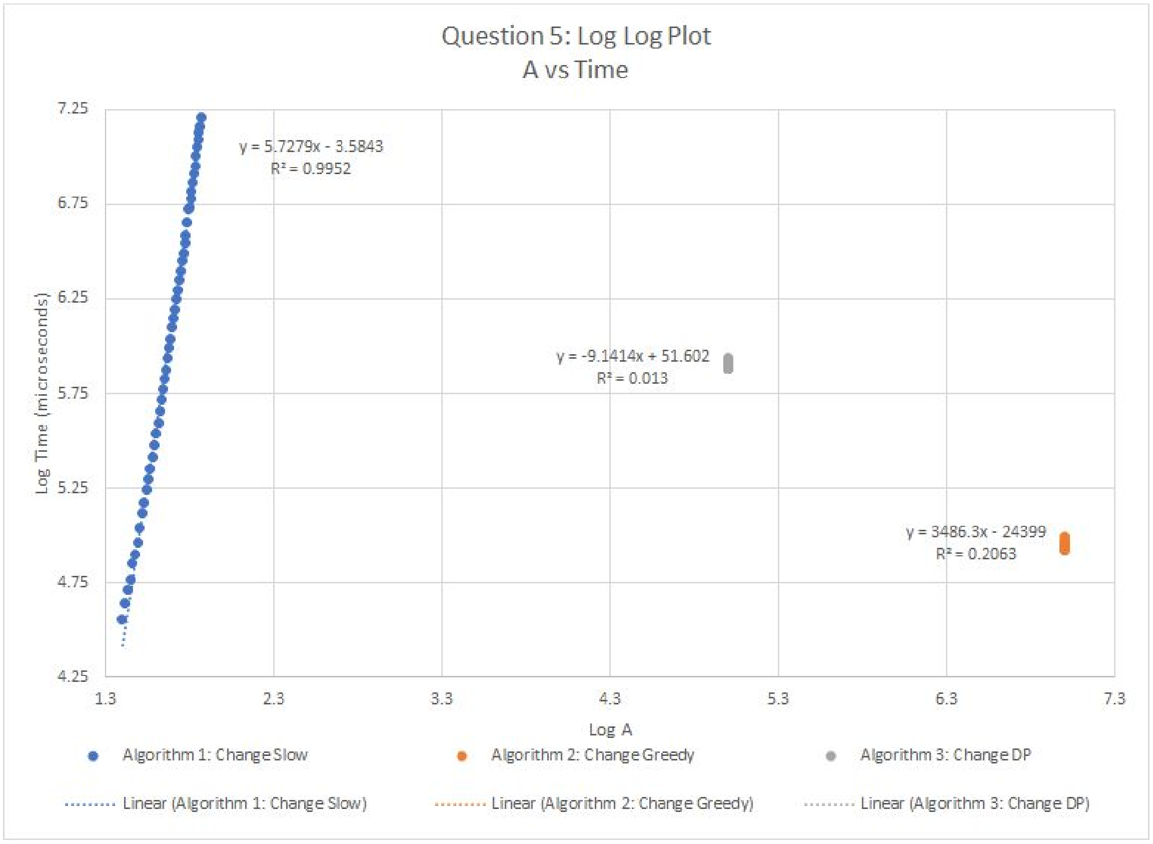
Group 7 - Project 2





Group 7 - Project 2





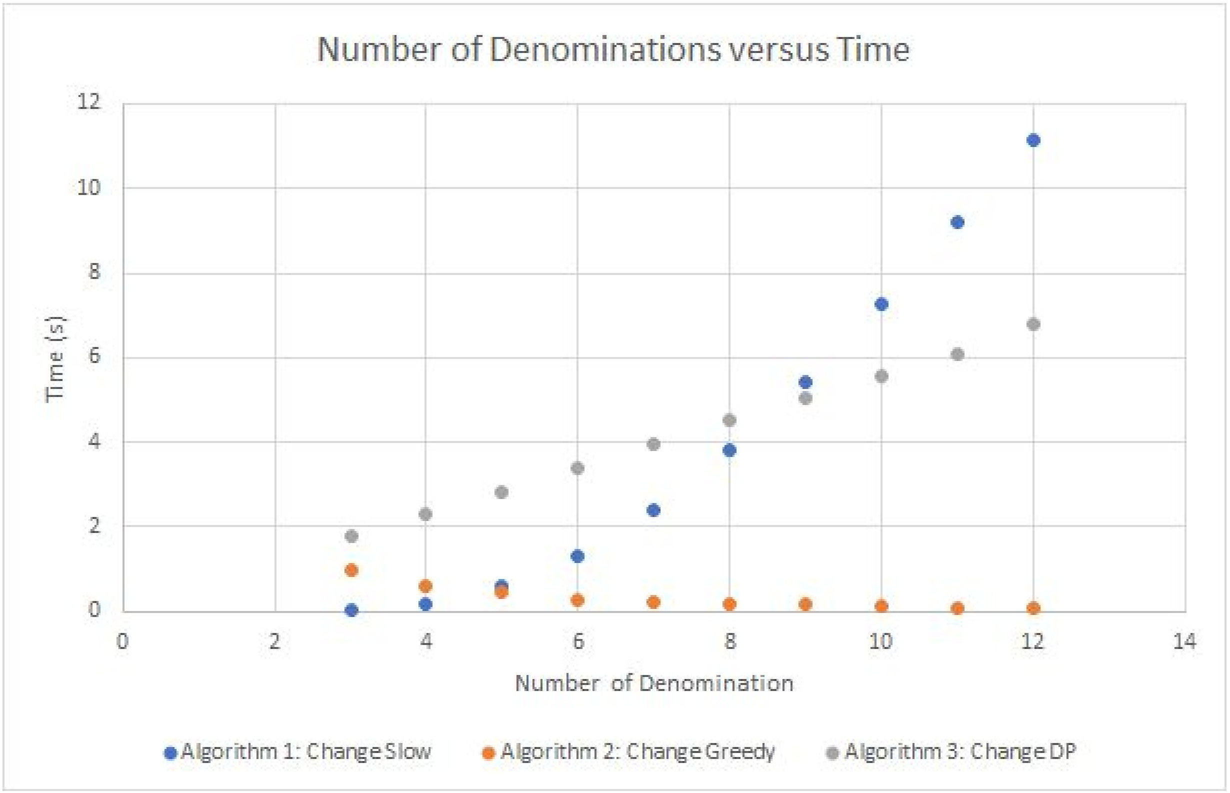
7. To determine if the size of coin options influence the running times of the algorithms, we devised a new experiment where using an array of prime numbers as the various denominations. Specifically, the array contained:

[1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41]

Then, for each algorithm, we chose a constant size for the target value, but used subarrays from size 3 to size 12 from the prime numbers array. For example:

[1, 2, 3]
[1, 2, 3, 5]
[1, 2, 3, 5, 7]
[1, 2, 3, 5, 7, 11]
[1, 2, 3, 5, 7, 11, 13]
etc.

In total, each algorithm ran 10 times to determine the solution for some constant target value. The plot for the resulting data is shown below. The time axis is a relative display to show the trending.



With respect to the behavior of each algorithm as the number of denomination options grow, the results are indicative of the following:

- The time to execute the *Change Slow* algorithm increases exponentially.
    - Since this is a brute force algorithm, it finds every single solution and chooses the the most optimal. When the number of denominations increase, the number of solutions increases exponentially.
- The time to execute the *Change Greedy* algorithm decreases.
    - As more options are added to the denominations, the value of the option tends to grow. Larger denomination values will allow the algorithm to run faster because it results in less loops that subtract from the target amount.
- The time to execute the *Change Dynamic Programming* algorithm increases at a linear rate.
    - The run time for this algorithm is M*N, where M is the number of denominations and N is the target size. As more options are added to the denominations array, the algorithm needs to run an extra iteration from 0 to the target size.

8. The greedy algorithm will produce the optimal solution in faster time than the dynamic programming solution. Because each denomination is a power of 3, 3^n,there will never be a case where twice a denomination will be more than the value of the next higher denomination - the greedy approach will never be wrong in choosing the highest possible value.

9. A system, such as the United States' (1, 5, 10, 25), where each denomination is twice or more the size of the next lowest, as any instance that requires two of the same coin can be replaced by one of the next greatest denomination. Another would be a system based on prime numbers (1, 3, 5, 7, 11), as double of each denomination (other than the highest) can also satisfied by the next higher coin and a lower coin.