Group Info:
- Names: Isaac Perez, Charlie Tuong, Elijah David
- Emails: isaacdan@csu.fullerton.edu, cturne@csu.fullerton.edu, ejdavid122@csu.fullerton.edu

Github Link: Isaacdan333/CPSC335-Project-2 (github.com)

Exhaustive Programming analysis:

Time complexity: O(2^N * N), where N is the number of stocks.

The exhaustive approach involves going through every single possible stock combination and seeing which combination fits under the fund limit, recomputing the combinations each time, resulting in a thorough but also costly solution. In the stock_maximzation function with parameters N, stocks_and_values, and amount, there is a for loop that iterates through all the possible number of stock combinations. There is then a for loop nested within, and this for loop iterates through each stock in the combination. In this for loop, current_stock, current_value, and combinations are updated with the total number and value of the stocks, with combinations storing the indices of the stocks in the combination (they are then reset to 0 at the beginning of a new execution of the outer for loop). Next is a check to see if a combination is valid and maximizes the number of stocks, and if it does, assigns current_stock to max_stocks. The function ends with returning max_stocks.

Dynamic programming analysis:

Time Complexity: O(N * amount)

N would be the number of stocks given and Amount is the money available to spend on the stocks.

- For example: the sample input would be O(4 * 12) because there are 4 subarrays within it.

What this dynamic problem does is break it down into sub problems where it stores the results in the created array to look over. This helps avoid repetitive calculations and results.The function, max_stocks, takes in N (number of subsets with each containing the # of available stocks and the value of them), stocks_and_values, and Amount. In the function, we create a 2d array to help store all of the possible answers from the subproblems. The array dimensions are based on the amount given and the number of subsets in the given stocks_and_values. The outer loop runs for each stock, N times and the inner loop runs for each possible amount from 0 to the given sum. The statements inside that run for constant time. In the if-else statement we check to see if the value of the current stock is greater than the amount available, j. If it is greater than the available amount it can't be included so the value from the previous set is taken. If it isn't greater then the current stock is included and the code takes the maximum value from whether we would include or exclude the current stock. It adds the current stock based on the value of it. It then updates the possible_stock array to place the max. After this is done, the result of possible_stock[N][Amount] is returned meaning the maximum value achieved from the given restraint.

The better approach:

The dynamic approach is better because its time complexity is much better than the exhaustive approach. The dynamic search's time complexity being O(N * amount) simplifies to being linear, as "amount" can be seen as just a constant multiplier, as "amount" doesn't change as N increases. Exhaustive on the other hand has exponential growth time complexity multiplied by linear growth, O(2^N * N), which simplifies to exponential growth due to the exponential part having greater weight than the linear part. Exponential growth is massively more inefficient than linear growth, therefore, dynamic is the better and more efficient approach in this problem.