Group Info:
- Names: Isaac Perez, Charlie Tuong, Elijah David
- Emails: isaacdan@csu.fullerton.edu, charlestuong150@gmail.com, ejdavid122@csu.fullerton.edu

Exhaustive Programming analysis:

Time complexity: O(N^2 * N)

N is the number of companies

Exhaustive method generates all the possible combinations and compares them all to see which has the highest possible value to be generated within the given amount. There are 2^N possibilities for the combination of stocks. For each combination we iterate through N stocks to find greatest value. Finally we check to see if the current value is less then or equal to the Amount and the current stock is greater than the max stock. This takes a great amount of time to compute for larger values.

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 dynamic approach is better because we are solving the problem as we iterate through the stocks and values by using the 2d arrays to store the results as we go along it, so there is no need to constantly re-solve the values to see if they add and match up. As we iterate through the list we check to see if the current value is greater than the previous value rather than finding all of the possible combinations first.This makes the process greatly faster in O(N*Amount), especially when it comes to bigger data sets.