Project 2

CPSC 335 - Algorithm Engineering

Douglas Yu, Shadi Hirbawi, & Taylor Livingston

3 December 2023

Github link: https://github.com/Douglasmyu/DouglasY_TaylorL_ShandiH_projet2

Time Complexity and Logic:

Part A: Exhaustive Search

Time complexity → O(2^n)

Logic:

```
#stock maximization
def exhaustive(N, stocks_and_values, amount):
    print("Using exhaustive method")
    #initialize the max set which is already set in our input
    max_amount = 0
    #loop through all sets
    for i in range(1 << N):
        #initialize the number of sets that can be packed to our max
        current_stock = 0
        current_value = 0
        #loop through the stock&values
        for j in range(N):
            if (i >> j) & 1:
                #check each comboination
                current_stock += stocks_and_values[j][0]
                current_value += stocks_and_values[j][1]
        if current_value <= amount:
            max_amount = max(max_amount, current_stock)
    return max_amount
```

Part B: Dynamic Programming

Time complexity → O(n)

Logic:

```
#stock maximization
```

```
def dynamic(N, stocks_and_values, amount):
    print("Using dynamic method")
    #initailize
    A = [[0]* (amount +1 ) for _ in range(N+1)]

    for i in range(N+1):
        for j in range(amount+1):
            if stocks_and_values[i-1][1]<=j:
                A[i][j] = max(A[i - 1][j], A[i - 1][j - stocks_and_values[i -
1][1]] + stocks_and_values[i - 1][0]) #0case
            else:
                A[i][j] = A[i - 1][j] #1case
    return A[N][amount]
```

Which approach is better?

Both approaches work well with the program; however, for better performance and efficiency dynamic programming is the best choice. Moreover, working with large sets of data, dynamic programming allows for overlapping of various problems that happen simultaneously. Granted, it may not work all the time, but for this scenario, dynamic programming is the way to go.