10/30/2023 Ryan Stapf

Project 1 Report

a. The recursive call is made on line 50 within the main function robot_stack_memo where the arguments b and n are incremented as follows:
    - robot_stack_memo(b – i, n – 1, comboDict)

   The comboDict array is manipulated within the function and remains unchanged during the recursive call, but must be referenced by the function.

   The function checks if it has already computed the current n and b combination results. If it has, it retrieves the result from the memoization table (memo) and returns it. This is essential for avoiding redundant calculations and improving efficiency.

   If the function has not computed the result yet, it calculates it. It does this by iteratively distributing robots into the current stack (from 0 to min(b, k) robots) with the for loop containing the recursion, and making recursive calls with reduced values of n and b which are incremented as described above. The results of these recursive calls are added together to calculate the total number of ways to distribute the robots.

b. There are two base cases:
    - n = 0 and b = 0
        - There is only one combination for 0 bots and 0 spaces
    - n = 0 or b < 0
        - There cannot be a negative number of bots, therefore 0 is the sentinel value

c. The worst-case scenario in this algorithm is when all possible combinations of each (b, n) pair must be calculated and is thus bound by n*b. Within each of these subproblems, the algorithm performs a loop referencing min(b, k) + 1. Therefore, the overall worst case time complexity is O(n*b*(min(b, k) + 1)).

   The overall space complexity is primarily determined by the memoization dictionary O(n*b) and the function call stack O(n) d.

Inputs: number of bots b, number of stack spaces n, stack height limit k
Output: Number of combinations

```
Algorithm robot_stack_iter(b, n, k):

    Initialize a 2D array called ComboArr

    # Base Case to describe 0 bots and 0 stacks
    ComboArr[0][0] = 1

    # Fill in ComboArr
    for i in range of 1 to n+1:
        for j in range of b + 1:
            for x in range of (minimum of j or k) + 1:
                ComboArr[i][j] += ComboArr[i - 1][j - x]

    return ComboArr[n][b]
```

e. The worst-case time complexity is O(n*b*k), where n is the number of stacks, b is the total number of robots, and k is the maximum number of robots per stack.

The worst-case space complexity is primarily determined by the 2D ComboArr, resulting in O(n*b).