Name: Poojitha Bagam
UID: U35394589

Intro to Theory of Algorithms
PROJECT-1 QUESTIONS

In this given project we are given,

b -> Number of Robots
n -> Number of stacks
k -> Maximum number of Robots that can go in each stack

We need to find how many ways the given ‘b’ number of Robots can distribute themselves into the ‘n’ number of stacks where each stack can have only ‘k’ number of robots.

a. The recurrence used in this project is memoized dynamic programming which follows as:
For the basecases :
if b = 0, it's a valid arrangement, and there's only one way to arrange 0 robots. T(b, n, k) = 1
If 'b' is negative or 'n' is less than or equal to 0, it's not possible to arrange the robots, and there are 0 ways. T(b, n, k) = 0

For recursive case :
For all other cases where 'b' is positive and 'n' is positive, the number of ways to arrange the robots depends on the sum of the number of ways to arrange 'b' robots into 'n-1' stacks (i.e., with one less stack) and the number of ways to arrange the remaining robots in the current stack (limited by 'k' robots per stack).
This is expressed as:
T(b, n, k) = Σ T(b - i, n - 1, k), for i in range(0, min(b, k) + 1)
And the relation would be

$$T(n) = \begin{cases} \Theta(1) & \text{if } n <= 0 \text{ or } b <= 0 \\ T(n-1) + \Theta(k) & \text{if } n > 0 \end{cases}$$

This means that for each possible value of 'i' (from 0 up to the minimum of 'b' and 'k' to respect the stack limit), we calculate the number of ways to arrange 'b - i' robots into 'n - 1' stacks and sum these results to get the total number of ways to arrange 'b' robots into 'n' stacks(limited by 'k' robots per stack).

b. The base cases of this recurrence are:

1. If b = 0 then return 1 as there will be only 1 way to distribute 0 robots into ‘n’ number of stacks. Where all the stacks are empty is the only possibility.

2. If b < 0 or n <= 0 the return 0 as there will be no way to distribute a negative number of robots or stacks.

c. The time and space complexities of this recursive algorithm using memoized dynamic programming are:

The recursive function makes a recursive call for each possible value of i in the recurrence. The number of possible values of i is O(b). Therefore, the number of recursive calls is O(b).

The memoization table stores the results of all subproblems that have already been solved. This ensures that each subproblem is only solved once, and the number of accesses to the memoization table is O(n).

Therefore, time complexity is **O(n * b)**

Space complexity is **O(n * b)** because the memoization table stores the results of all subproblems.

d. Pseudo-code for an iterative approach for this algorithm is:

```
Given  Input : b n k
          Output : number of ways ‘b’ robots are arranged in ‘n’ stacks with a
                   maximum of ‘k’ robots per stack
Algorithm: IterativeRoboStack
# Initialize memo with -1
for i = 1 to n do
   for j =1 to b do
      if j = 0 then
         memo[ i ][ j ] = 1;
      else if i = 0 then
         memo[ i ][ j ] = 0;
      else
         for x in range(max(0, j - k, j + 1):
            memo[ i ][ j ] += memo[i-1][x];
return memo[n][b];
```

This pseudo-code iteratively calculates the number of ways to arrange ‘b’ robots into ‘n’ stacks with a maximum of ‘k’ robots per stack, storing the results in a memoization table. The memoization table ensures that each subproblem is only solved once, which significantly improves the performance of the algorithm.

e. The time and space complexities for this iterative approach are:
In this iterative approach ,
The outer loop runs from i = 1 to n, which is O(n) iterations.
The middle loop runs from j = 1 to b, which is O(b) iterations.
The inner loop runs from x = max(0, j - k) to x < j + 1, which can have up to k iterations.
Therefore, the time complexity is **O(n * b * k)** because it needs to iterate over all possible values of i in the recurrence and Space complexity is **O(n * b)** because of the memoization table.