Project Questions

a. The recurrence used for this problem is based on dynamic programming. For a given set of robots to distribute (`b`), the number of stacks available (`n`), and the maximum number of robots that can be in each stack (`k`), the recurrence is as follows:

```
O(min(b,k)) + T(n-1) + θ(1)
```

This recurrence expresses the number of ways to distribute `b` robots into `n` stacks, considering the constraint of a maximum of `k` robots in each stack.

b. The base cases of the recurrence are as follows:

- If `b` is 0 (no robots to distribute), there is only one way to do it (return 1).

$T(n) = 1$

- If `n` is less than or equal to 0 (no stacks available) or `k` is less than or equal to 0 (maximum robots per stack is 0), there are no valid ways to distribute the robots (return 0).

$T(n) = 0$ for all $n<=0$ and $k<=0$

c. Time and Space Complexities:

- Time Complexity:

1) The function is called recursively for each possible value of i in the range from 0 to min(k, b), which is O(min(k, b)).

2) In each recursive call, the function explores n stacks.

if we consider both factors, the time complexity is O(n * min(k, b)).

- Space Complexity:

The space complexity is primarily influenced by the memoization dictionary named memo, which stores previously computed results for specific combinations of b, n, and k. In this case, the space complexity is O(b * n) because the dictionary may need to store results for each combination of b and n.

d. Pseudo-Code for Iterative Approach:

```python
Algorithm distribute_robots(b, n, k):
  Create a 3D array dp[b+1][n+1][k+1]

  for i from 0 to b:
    for j from 0 to n:
```

```
      for x from 0 to k:
        if i == 0:
          dp[i][j][x] = 1
        else if j <= 0 or x <= 0:
          dp[i][j][x] = 0
        else:
          dp[i][j][x] = 0
          for y from 0 to min(x, i):
            dp[i][j][x] += dp[i-y][j-1][x]

  return dp[b][n][k]
```

This pseudo-code describes an iterative approach to calculate the number of ways to distribute robots into stacks using a 3D array `dp` to store intermediate results.

e. Time and Space Complexities for the Iterative Approach:

- **Time Complexity**: O(n * b * k) due to the three nested loops.

- **Space Complexity**: O(n * b * k) for the 3D array `dp`.