Dynamic Programming:
a-What is the recurrence you are using for this problem?
Response:I'm using top down approach
b-What are the base cases of your recurrence?
Response:When we passed through all stacks if there is a remaining robots return 0 and don't count this a way
c-What are the time and space complexities of your algorithm?
Response:time complexity is:
number_of_stacks x number_of_robots x min(number_of_robots,k)=n*b*min(b,k)
space complexity is:
number_of_stacks x number_of_robots=n*b
d-Iterative approach:
first of all initialize our dp array with 0 for each possible pair(i,j) where 0<=i<=n && 0<=j<=min(b,k)
define the base case :dp[0][0]=1
loop through stacks
for every stack i loop through possible number of robots let's call it j:
for the current state (i,j) the number of ways is the sum of all ways in the previous state the stack i-1 with all possible number of robots in previous state (j-idx_k) where 0<=idx_k<=k
space complexity is :n*b
time complexity is:n*b*min(k,b)
here is the implementation in python

```python
def iterative(n,b,k):
    #initialize dp array with 0 value
    dp = [[0] * (b + 1) for _ in range(n + 1)]
    #base case
    dp[0][0]=1
    #loop through stacks
    for i in range(1,n+1):
        #loop through robots
        for j in range(0,b+1):
            #loop through number of robots to put in current stack i
            for idx_k in range(min(j+1,k+1)):
                #number of ways in current state i is sum of number of ways of previous states
                # with all different remaining number of robots(j-idx_k)
                dp[i][j]=dp[i][j]+dp[i-1][j-idx_k]
    return dp[n][b]
```