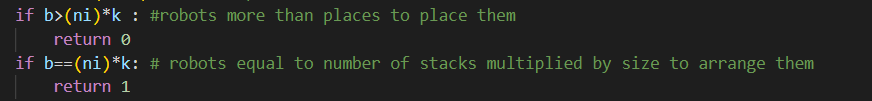
1. The recurrence I am using for this problem is   
   

Where max is min(b,k) and mini is calculated using  
Text

Description automatically generated

The problem solves the A(b,n,k) to where j value varies from max to mini in the above code snippet, where A is the number of possibilities for the given b,n,k respectively

1. The base cases for my approach are :  
     
   The first case returns 0 as it considers the cases where there are more robots than places available  
   The second case returns 1 as it considers the case where robots number is equal to the places available which can only be placed in 1 way
2. Time complexity for worst case scenario occurs when recurrence calls min(b,k) (lets just say **min** to be the value)  
   We get recurrence relation as

= **O(n\*b)**

For space complexity we are using a 2d array(memo) whose size is (n+1)\*(b+1) which is our space complexity **O(n\*b)**

1. **Iterative approach:**For Iterative approach I’ll be using the same code as in memorization but with slight modifications to the way we call the function ways  
   **Algorithm**

def totalPossibilities(b,n,k) :

memo = []

for i in range(b+1) :

memo.append([-]\*(n+1))

def instance(b,ni)

if b>(ni)\*k :

return 0

if b==(ni)\*k:

return 1

if memo[b][ni] != -2 :

return memo[b][ni]

z = ni-1

mini=0

if (z\*k)<b :

mini = b-(z\*k)

max = min(b,k)

ct = 0

for j in range(b-max,b-mini+1,1) :

ct+=instance(j,ni-1)

memo[b][ni] = ct

return(memo[b][ni])

for numi in range(1,b+1,1) :

for numj in range(1,n+1,1) :

instance(numi,numj)

print(b,n,k, ” =” ,memo[b+1],[n+1])

1. Time case complexity for iterative approach is same as Memoized method’s worst-case complexity as it’ll be calculating all possibilities

Therefore, Time complexity is **n\*b**

For space time complexity we are using a memo same as Memoized method thus  
Space time complexity is (**n\*b)**