Q1) Consider the following recursive algorithm

```
K(n){
        IF (n = 1)
                RETURN 1
        ELSE
                SUM = 0
                FOR( i = 1 to n − 1 )
                        SUM = SUM + (K(i) + K(n − i))/3 + n/2
        RETURN SUM
}
```

Convert the recursive code given above into bottom up iterative dynamic programming algorithm.

Solution:

Let's dry run the above code for better understanding.

n= 1, K(1) = 1

n=2, K(2) = (K(1) +K(1))/3 + 2/2 = 2/3+1 = 1.66

n=3, K(3) = [(K(1) +K(2))/3 + 2/2] + [(K(2) +K(1))/3 + 3/2] = [(1+1.66)/3+1] + [(1.66+1)/3+1.5] = [1.88] + [2.38] = 3.26

We can see that we need value of K(2) multiple times. If we can use memoization to store K[2] in an array then we can save repeated computations. The above recursive code is calculating K(2) again and again.

We can create an array DP[i] which saves solution for i. We will need two for loops, one for calculating smaller subproblems and other for calculating answer of one subproblem.

```
Iterative(n){

                DP[1] = 1  // smallest subproblem
                FOR( j = 2 to n )   // j represents size of subproblems
                        SUM = 0
                        FOR( i = 1 to j − 1 )
                                SUM = SUM + (DP[i] + DP[j − i])/3 + j/2
                        DP[j] = SUM
                RETURN DP[n]
}
```

Q2) Consider the following recursive algorithm

```
P(n, k){
        IF (k > n)
                RETURN 0
        IF (k == n || k == 0)
                RETURN 1

        RETURN   P(n-1, k-1) + P(n-1,k)
}
```

Convert the recursive code given above into bottom up iterative dynamic programming algorithm.

Solution

```
P(n, k){
     C[0] = 1

     FOR( i = 1 to n )
         FOR( j = 0 to min(i,k) )
             IF(j == 0 || j == i)
                    C[i][j] = 1
             ELSE
                   C[i][j] = C[-1][j] + C[i-1][j-1]

      RETURN   C[n][k]
}
```