Dice Throw

Given n dice each with m faces, numbered from 1 to m, find the number of ways to get sum S by adding the values appearing on the faces when all the dice are thrown.

Example

If we have 2 dice each with 2 faces, there are only 2 ways to get 3.

Question 1: Write a function, DICE, using a top-down approach to compute the number of ways to get sum S given n dice each with m faces

Subproblem: $Dice(i, m, X)$ the total number of ways to get the sum $0 \leq X \leq S$ using $1 \leq i \leq n$ dice each with m faces

Guess: the value on the face of the last dice (m possibilities)

Recursive Relation:

$$Dice(n, m, S) = \begin{cases} 0, & \text{if } n.m < S \text{ or } S < 0 \\ 1, & \text{if } n = 1 \text{ and } 1 \leq S \leq m \\ \sum_{i=1}^{m} Dice(n-1, m, S-i), & \text{otherwise}. \end{cases}$$

Original Problem: $Dice(n, m, S)$

```python
def Dice(n, m, S, memo=None):
    if memo is None:
        memo = {}
    if (n, S) not in memo:
        if S > n*m or S<0:
            ret = 0
        elif n==1 and S >= 1 and S <= m:
            ret = 1
        else:
            ret = sum([Dice(n-1, m, S-i, memo) for i in range(1, m+1)])
        memo[(n, S)] = ret
    return memo[(n, S)]
```

Question 2: Analyze the running time of your code.

Running Time Analysis: Number of subproblems: $O(n \cdot S)$
Computation time per subproblem: $O(m)$
Total running time: $O(n \cdot m \cdot S)$

Question 3: Write a function, DICEBOTTOMUP, to solve the same problem using a bottom-up approach.

```
def DiceBottomUp(n, m, S):
    if S > n*m or S <= 0:
        return 0
    table = [[0 for j in range(S+1)] for i in range(n+1)]
    for i in range(1, n+1):
        for j in range(S+1):
            if i==1 and j >= 1 and j <= m:
                table[i][j] = 1
            else:
                table[i][j] = sum([table[i-1][j-k] for k in range(1, min(m+1,j))])
    return table[n][S]
```

Question 4: Analyze the running time of your code.

Each loop index (i and j) takes on at most n and S values. Inside the loops we compute the sum of m elements of the array. Thus, the total running time of this algorithm is $O(n \cdot m \cdot S)$.