1.

		n					
		1	**2**	**3**	**4**	**5**	**6**
k	**1**	B/I	B/I	B/I	B	B	B
	2		B/I	I	D		
	3			B/I	D	*	

2. To evaluate the elements of this data structure, you can iterate through the rows and, for each row, iterate through each of the columns. For each cell processed, the result will be placed in the data structure before continuing. In this way, the first row of base cases (k = 1) will be evaluated and, for every row after, the base case where k = n will be evaluated prior to elements that are directly or indirectly dependent on it. Continuing this process, the last element to be evaluated will be the solution (*).

3. **Input:** *n*: number of 'objects'
Input: *k*: number of 'piles'
Restriction: n >= k
Output: S(n, k)
Algorithm: IterativeStirling

```
1  int arr[k][n] // 2-D array to store dependencies (k rows, n
   columns)
2  for (int i = 0; i < k; i++)          // Iterate over rows
3  {
4     for (int j = 0; j < n; j++)       // Iterate over columns of row
5     {
6        if (i == 0 || i == j)          // Base case (0 for indexing)
7           arr[i][j] = 1               // Result is 1
8        else if (i > j)                // Undefined case
9           arr[i][j] = 0               // Result is 0
10       else
11          arr[i][j] =(i + 1) * arr[i][j - 1] + arr[i - 1][j - 1]
12    }
13 }
14 return arr[k - 1][n - 1]
```