Offline 6 on DP

1) The Longest Increasing Subsequence (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order.

   For example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 (red colored ones).

   Process: we will use a **dptable** to store the Longest Increasing Subsequence value upto each array element.

array	10	22	9	33	21	50	41	60	80
dptable	1								?
	0	1	2	3	4	5	6	7	8

Base case:
LIS for the first element is itself.
So, LIS(0)=1

Recursive step:
LIS upto array size n, **LIS(n) = 1 + max(LIS(p), LIS(q), LIS(r), … etc.)**

here, p, q, r are the indices of those array elements before the element at n that are smaller than arr[n] .

So, search for smaller elements from 0 to (n-1) and find out the maximum value of LIS() and add 1 to calculate the value of LIS(n).

Sample Input	Sample Output
{10, 22, 9, 33, 21, 50, 41, 60, 80}	6

2) Given a value N, if we want to make change for N cents, and we have infinite supply of each of S = { S1, S2, .. , Sm} valued coins, how many ways can we make the change? The order of coins doesn't matter.

For example, for N = 10 and S = {2,5,3,6}, there are five solutions: {2,2,2,2,2}, {2,2,3,3}, {2,2,6}, {2,3,5} and {5,5}. So, the output should be 5.

Process: we will use a **2D dptable**, where one dimension will track the coins and other dimension will consider the change value.

			Change, N										
			0	1	2	3	4	5	6	7	8	9	10
	empty	0	1	0	0	0	0	0	0	0	0	0	0
	2	1	1										
	5	2	1										
	3	3	1										
coins	6	4	1										?

Base Case 1:
When the change value N is 0 then we can make 0$ change using any sized coin array, cause we don't need any coin for 0$ change.

Base Case 2:
When the coin array is empty then we can create any change from 1$ to N$ using an empty array.

Recursive step:
For any other cell containing coin array of size sz and change amount n

we have two options:

option 1: ignore the last sz^{th} coin and let your friend to find a solution for you using (sz-1) sized coin array and change amount of n.

option 2: consider the last sz^{th} coin and let your friend to find a solution for you using the same sz sized coin array (cause we can consider same coin multiple times) and remaining change amount of (n-sz^{th} coin amount). If the remaining change amount is negative then ignore this step.

CC(coin_arr[], sz, n)
= CC(coin_arr[], sz-1, n)
+ CC(coin_arr[], sz, n-coin_arr[sz])

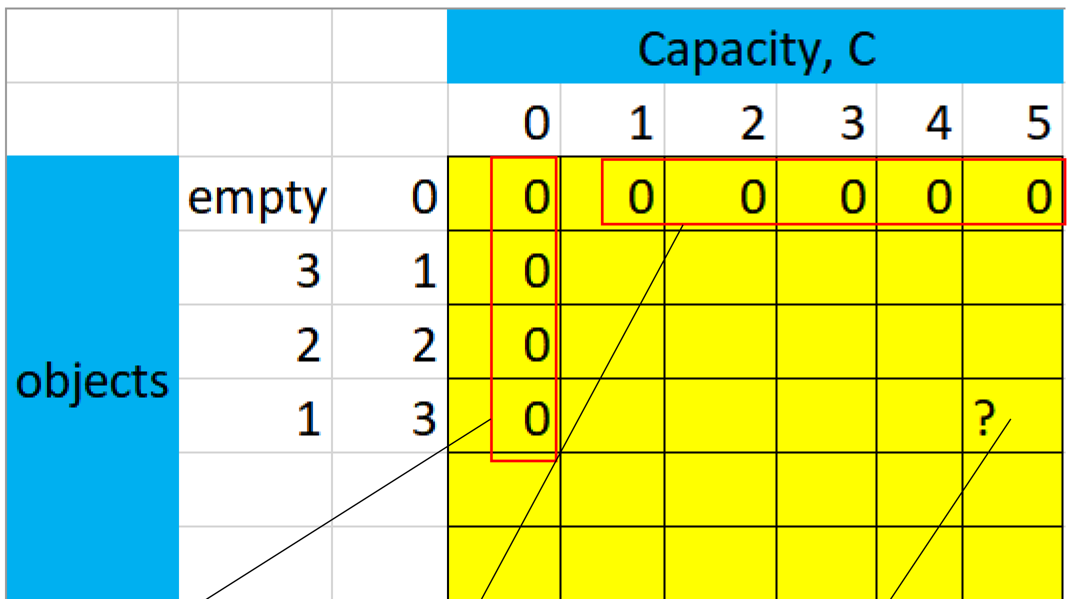
Sample Input	Sample Output
{2, 5, 3, 6} N=10	5

3) Given two integer arrays V[0..n-1] and W[0..n-1] which represent values and weights associated with n items respectively. Also given an integer C which represents knapsack capacity, find out the maximum value subset of V[] such that sum of the weights of this subset is smaller than or equal to C. You cannot break an item, either pick the complete item or don't pick it (0-1 property).

Sample Input	Sample Output

{12,10,6} //value array
{3,2,1} //weight array
5 //capacity
	22

Process: we will use a **2D dptable**, where one dimension will track the objects and other dimension will consider the capacity.



Base Case 1:
When the capacity is 0 then our gain will be zero.

Base Case 2:
When there exist no objects then the gain will also be 0.

Recursive step:
For any other cell containing n objects and capacity c

we have two options and we will choose the best option:

option 1: ignore the last n^{th} object and let your friend to find a solution for you using rest of the (n-1) sized objects and capacity of c.

option 2: consider the last n^{th} object with gain V[n] and let your friend to find a solution for you using the rest (n-1) objects and capacity of (c – W[n]). Ignore this step if the remaining capacity (c-W[n]) is negative.

KS(W[], V[], sz, C)
= max(KS(W[], V[], sz-1, C) ,
V[n] + KS(W[], V[], sz-1, C-W[sz])
)