Convert the given Brute force solutions into bottom-up (iterative/tabular) DP solution. Assume that all the arrays are globally available.

Q#1:
Coin Change(N, Amt)
{
  if(N == 1)
  return C[N]
  if(Amt == 0)
  return 0
  if(C[N] > Amt)
  return Coin Change (N-1, Amt)
  else
  return min(Coin Change(N-1, Amt), 1+Coin Change(N-1,Amt-C[N]))
}
	Problem:
	Return the given amount with minimum number of coins.
	
	// (N) number of coins
	// (Amt) //amount to be returned.

Q#2:
Coin Change (N, Amt){
  if(N == 0)
  return C[N]
  if(Amt == 0)
  return 0
  if(C[N] > Amt)
  return Coin Change(N-1, Amt)
  else
  return Coin Change(N-1, Amt) + Coin Change(N-1,Amt-C[N]))
}
	Problem:
	Count all the possible ways to return the given amount.
	
	// (N) number of coins
	// (Amt) //amount to be returned.

Q#3:
KSP(N,M){
  if(N == 0)
  return 0
  if(M == 0)
  return 0
  if(W[N] > M)
  return KSP(N-1, M)
  else
  return max(KSP(N-1, M), V[N] + KSP(N-1,M-W[N]))
}
	Problem:
	Maximize the profit by including those items in the knapsack having maximum value.
	
	// (N) number of items
	// (M) capacity of bag

Q#4:
LCS(N,M){
  if(N == 0)
  return 0
  if(M == 0)
  return 0
  if(s1[N] == s2[M])
  return 1 + LCS(N-1, M-1)
  else
  return min(LCS(N-1, M),LCS(N,M-1))
}
	Problem:
	Return the length of longest common subsequence.
	
	// (N and M) represent the length of strings

Q#5:
ED(N,M){
  if(N == 0)
  return M
  if(M == 0)
  return N
  if(s1[N] == s2[M])
  return ED(N-1, M-1)
  else if(s1[N-1]==s2[M] AND s1[N]==s2[M-1])
  return ED(N-2, M-2)
  else
  return 1+ min(ED(N, M-1), ED(N-1,M), ED(N-1,M-1))
}
	Problem:
	Convert one string into the other using minimum number of operations.
	
	// (N and M) represent the length of strings.

Q#6:
SSP(N,K)
{
  if(N == 0)
  return 0
  if(K == 0)
  return 1
  if(A[N] > K)
  return SSP(N-1, K)
  else
  return SSP(N-1, K) || SSP(N-1, K-A[N])
}
	Problem:
	Determine whether there exists any subset of array having sum equals to the target value.
	
	// (N) length of array
	// (K) target value

Q#7:
RC(N)
{
  if(N==0)
  return 0
  q = 0
  for (i=1 to N)
  {
  q = max(q, RC(N-i) + Pr[i])
  }
  return q
}
	Problem:
	Rod of length N can be cut into different segments. Return the maximum profit that can be achieved.