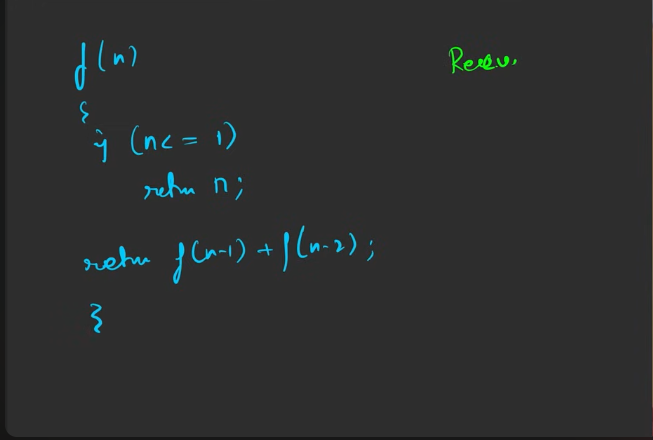
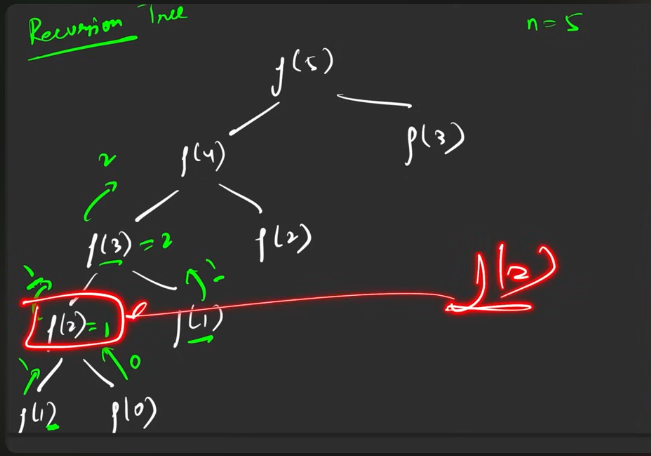
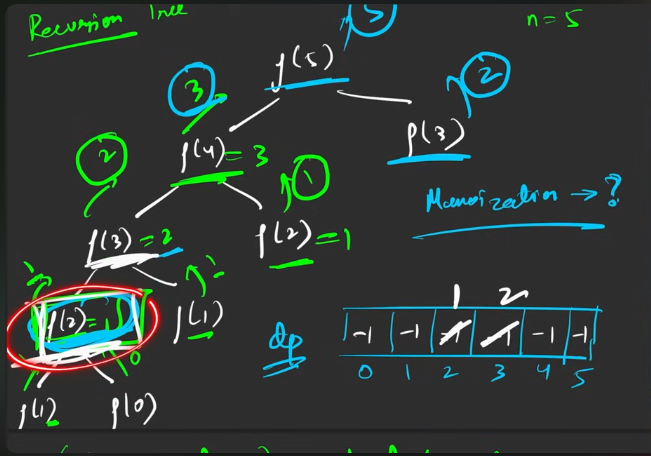
**Fibbinocci Number**



Overlapping sub problems:   
The computation of certain kind of the functions is already done, doing it again will lead to time waste. So we can store the result of thse kind of results and later we are giung to use it.

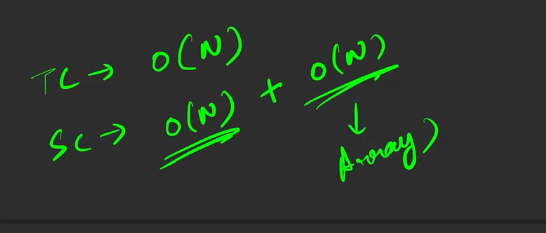


This is where the memoization kicks in. e are going to store the result of this.

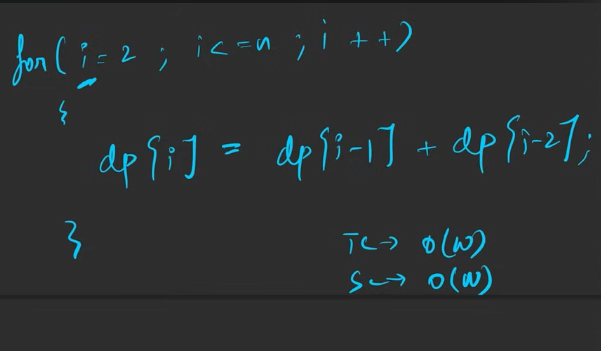
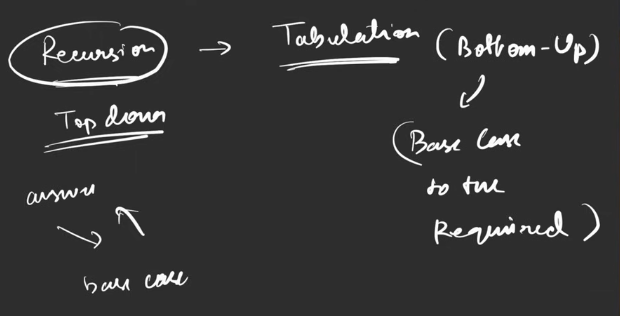


**Steps to convert from reccursion to DP**:

1. Declare the array based on the number of elements to store the intermediate results.
2. The result generated from the reccurssion to be stored in the particulare place, based o the above declaration.
3. Before calling for the reccursion after the base case, check whether the contnt is available in the array declared or not, if availble then return the answer there itself , else return the reccurssion call.



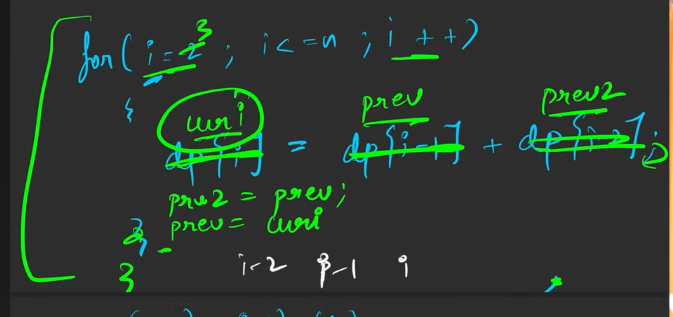
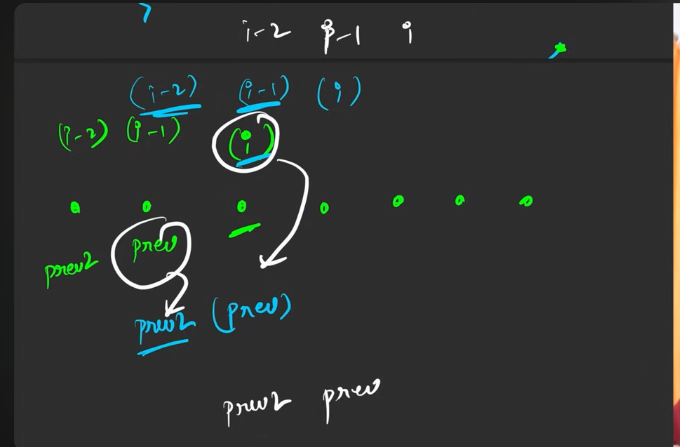
How to convert from reccursion into tabulation method?



To convert from the reccurssion to tabulation:

1. The base cases has to be precomputed ,like the above assignment.
2. The one which comes after the precomputation should be kept in the loop, based on the loop, the answer will be computed:
   1. Convert the reccursiin to storage .
   2. The range that should be followed depends based on the followed base case.

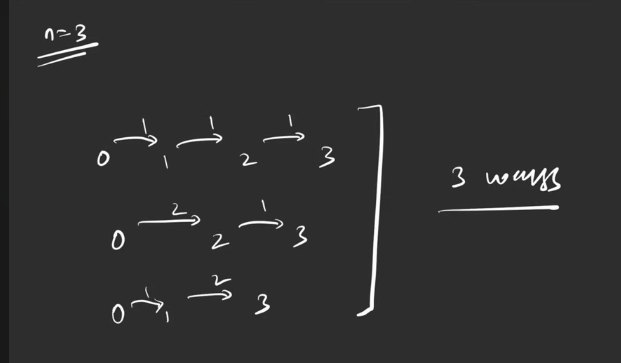
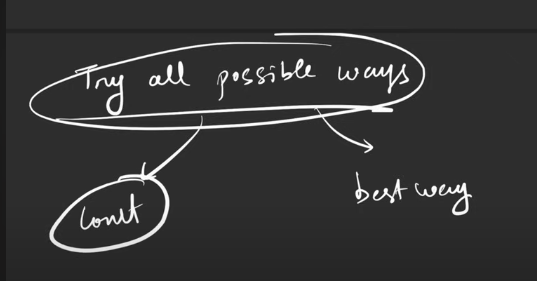
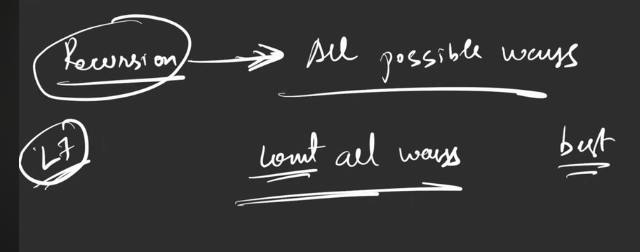
The above way is better we are remove the reccurssion.



**COUNT WAYS TO REACH NTH STAIRS**

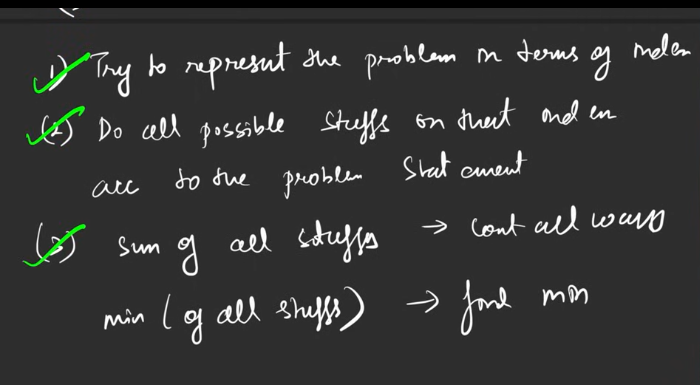
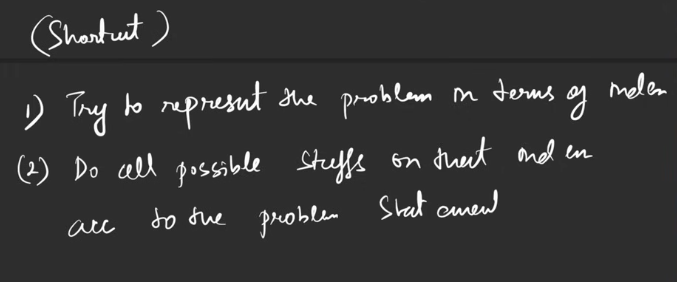


When do we get to know that we need o do reccursion?



**This is the shortcut way to find the solution for all the problems in the DP.**

If you master recurssion, you can master DP.

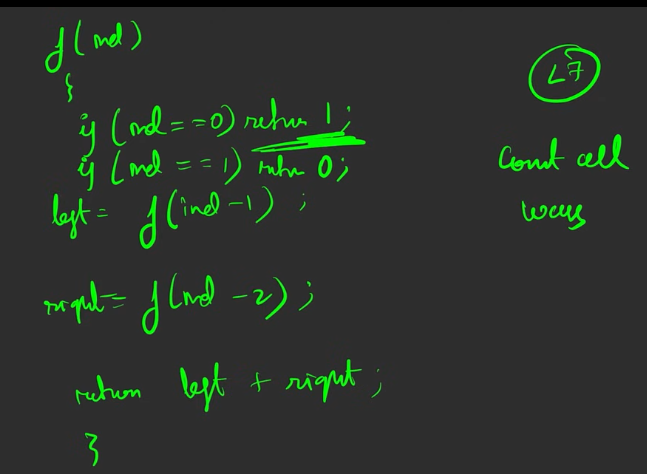


In this a person can make the jump of either 1 or 2 from the current step to reach Nth step.

So according to the reccursion, we are moving in top down approach, we can make, step-2, or step - 1 from the current.

Asked for all possible step, we are going to return the sum of the 2 above.

The l = f(n-1), r = f(n-2), l+r

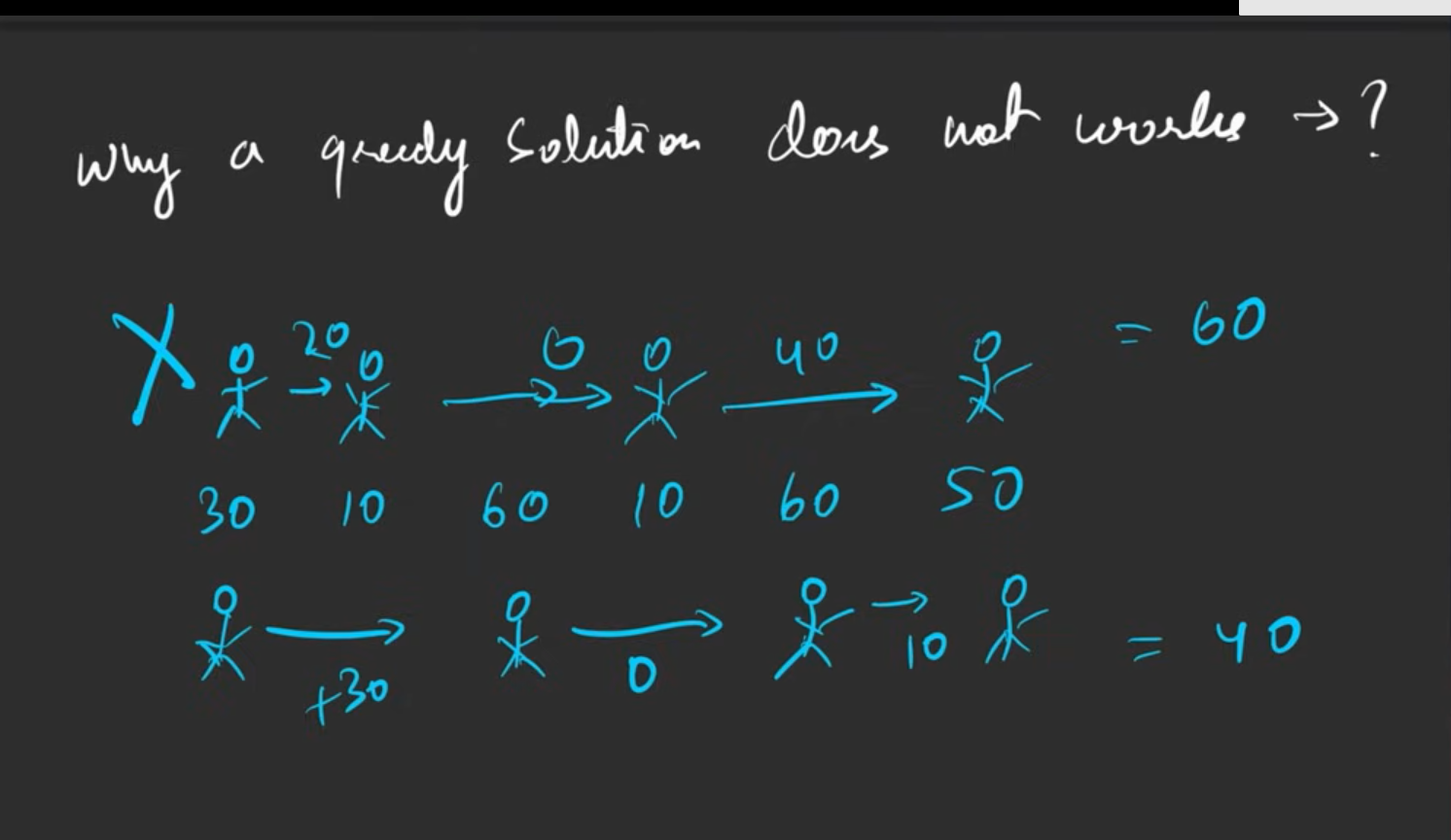
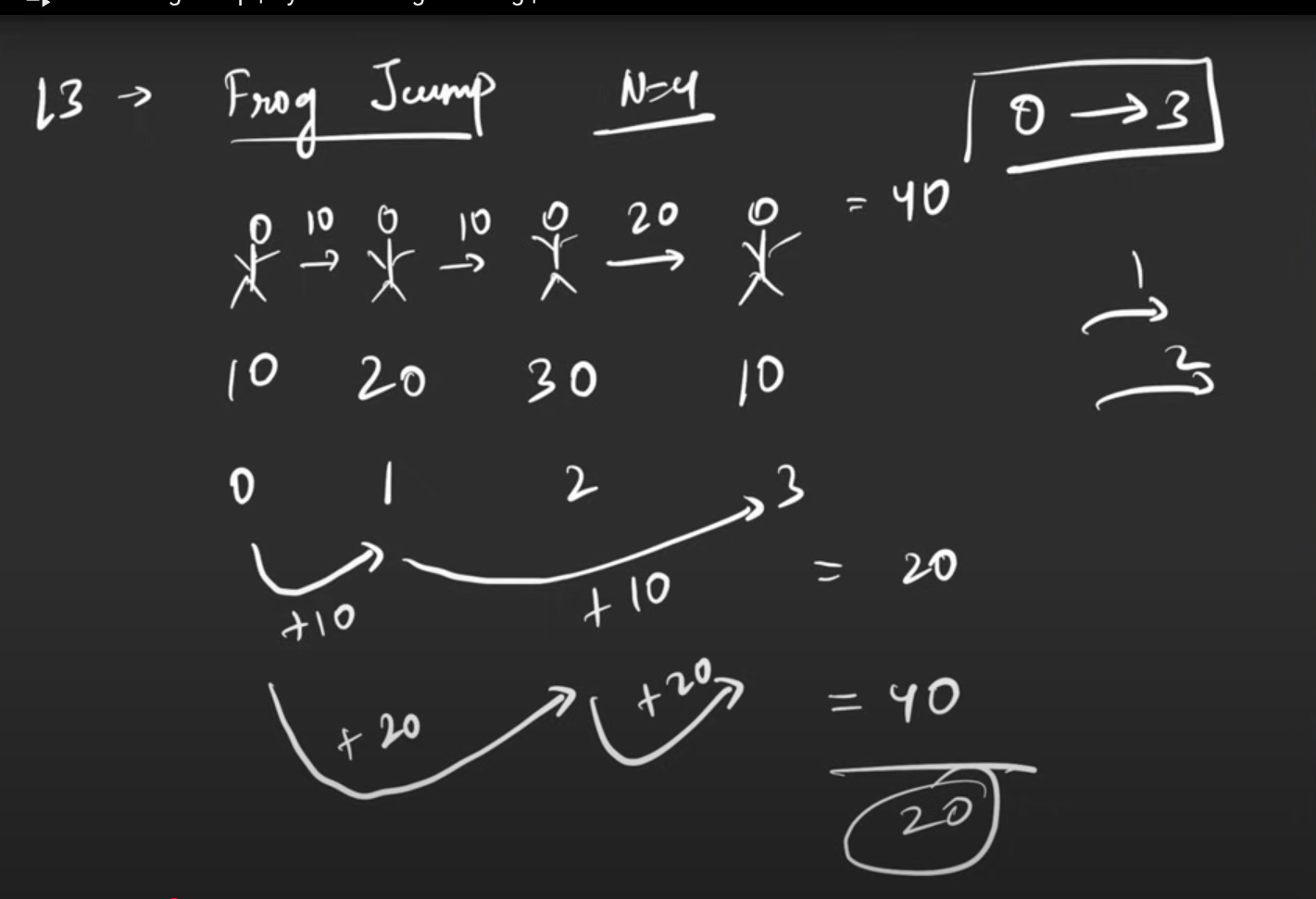


The second return is 1, not 0.   
From here, just go for the  
1. Fibbinocci tabulations( bottom to top ).

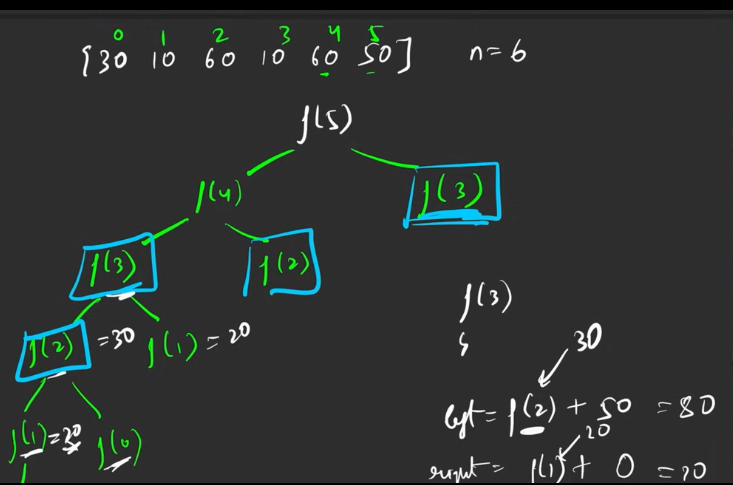
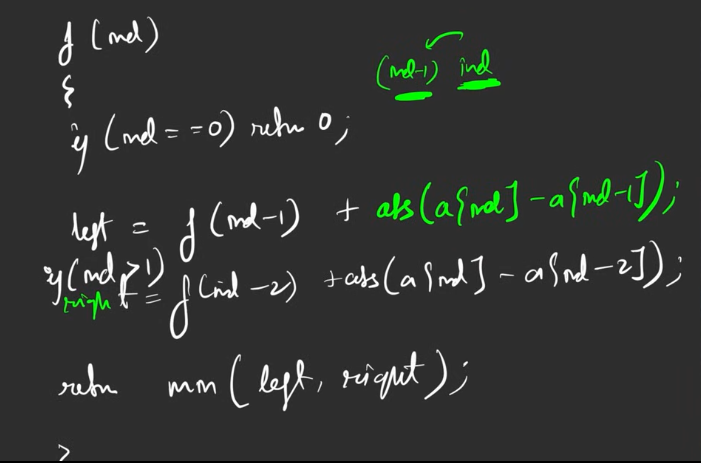
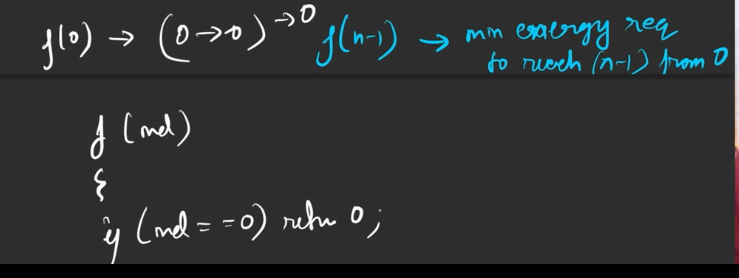
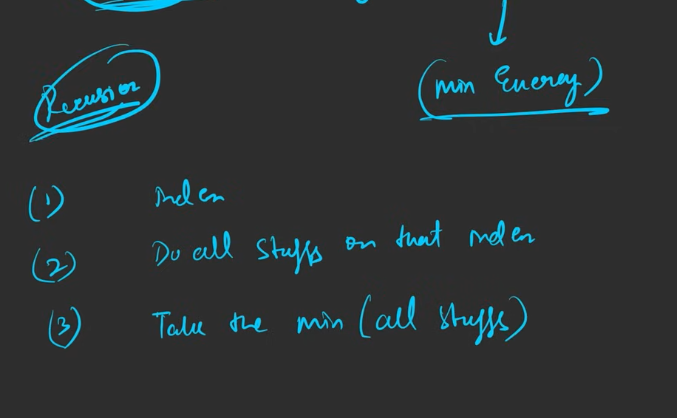
2. Then you can use the top to bottom – memoization.

3. Then make the space optimisation.

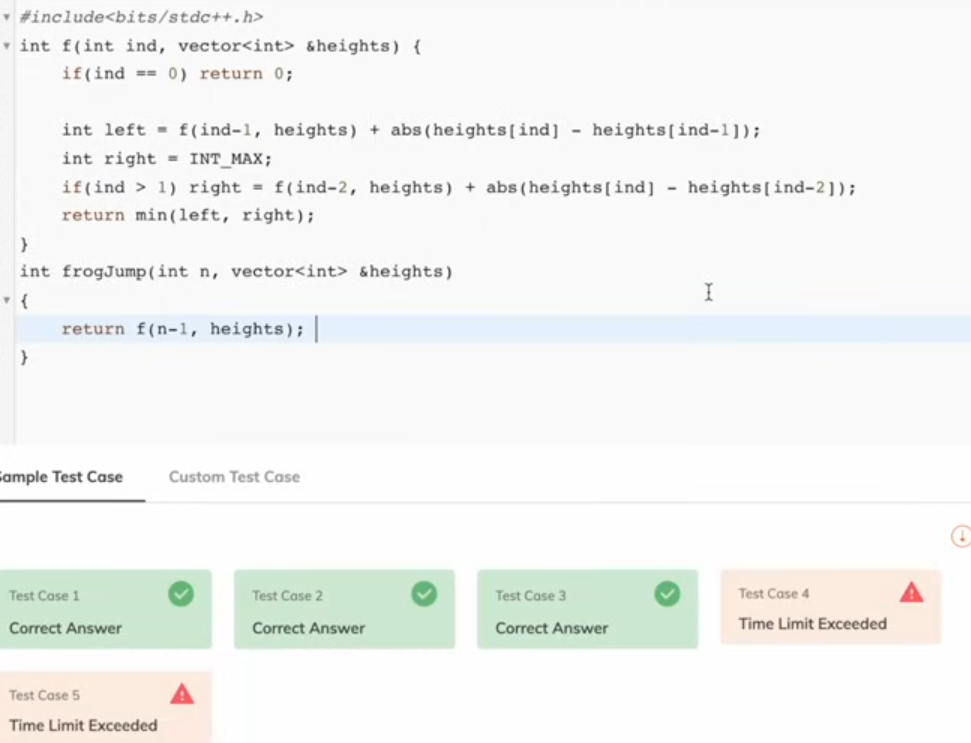
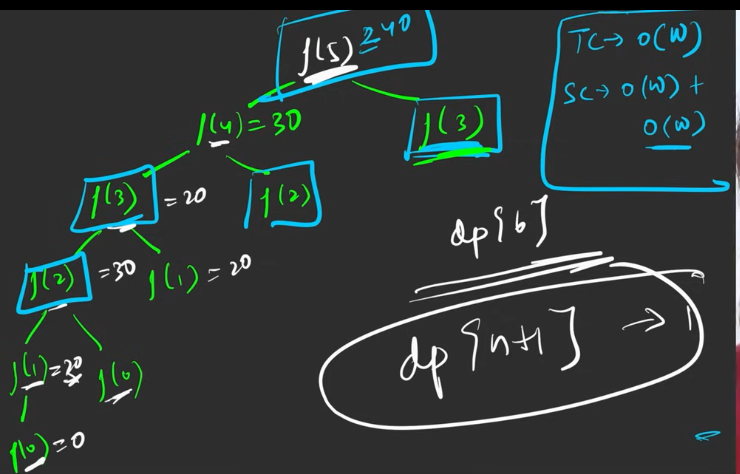
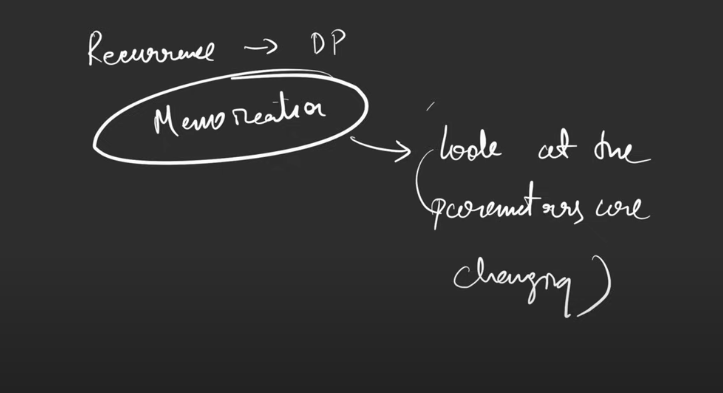
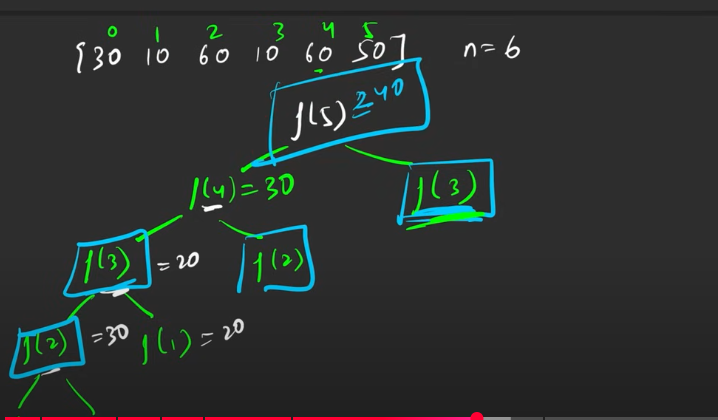
**FROG JUMP**



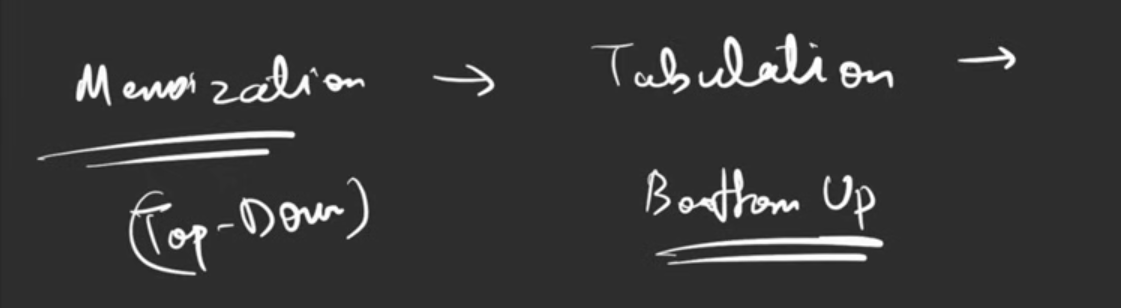
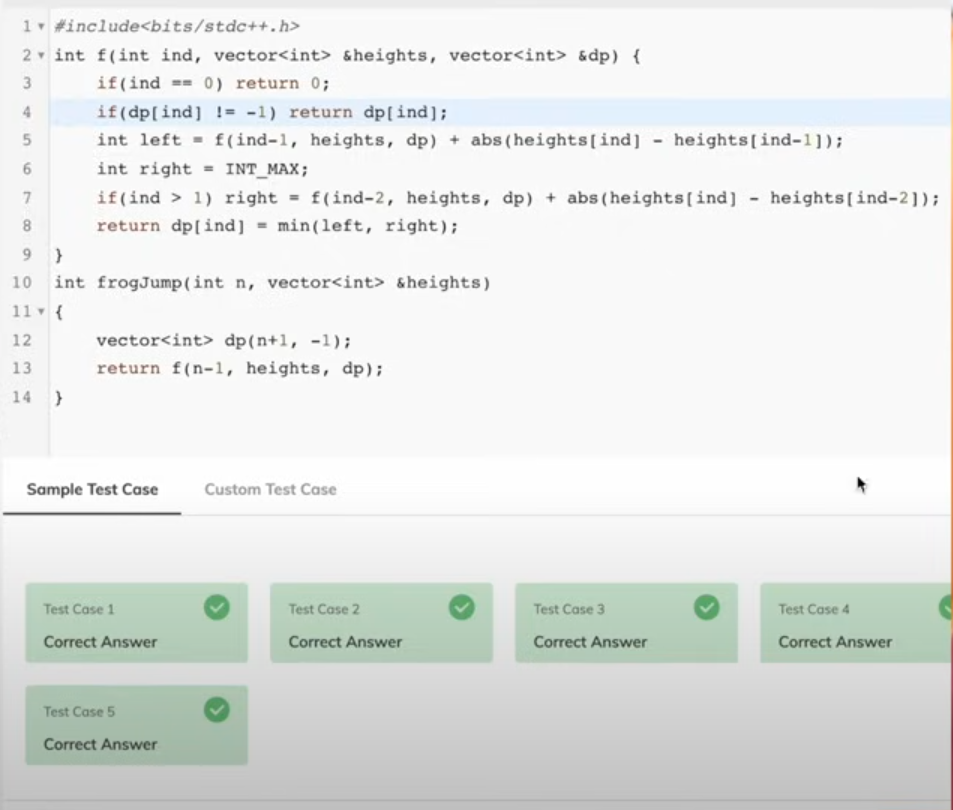
The first half was created using the greedy, the greedy will not make the correct answer always there might be the case where the greedy will fail.



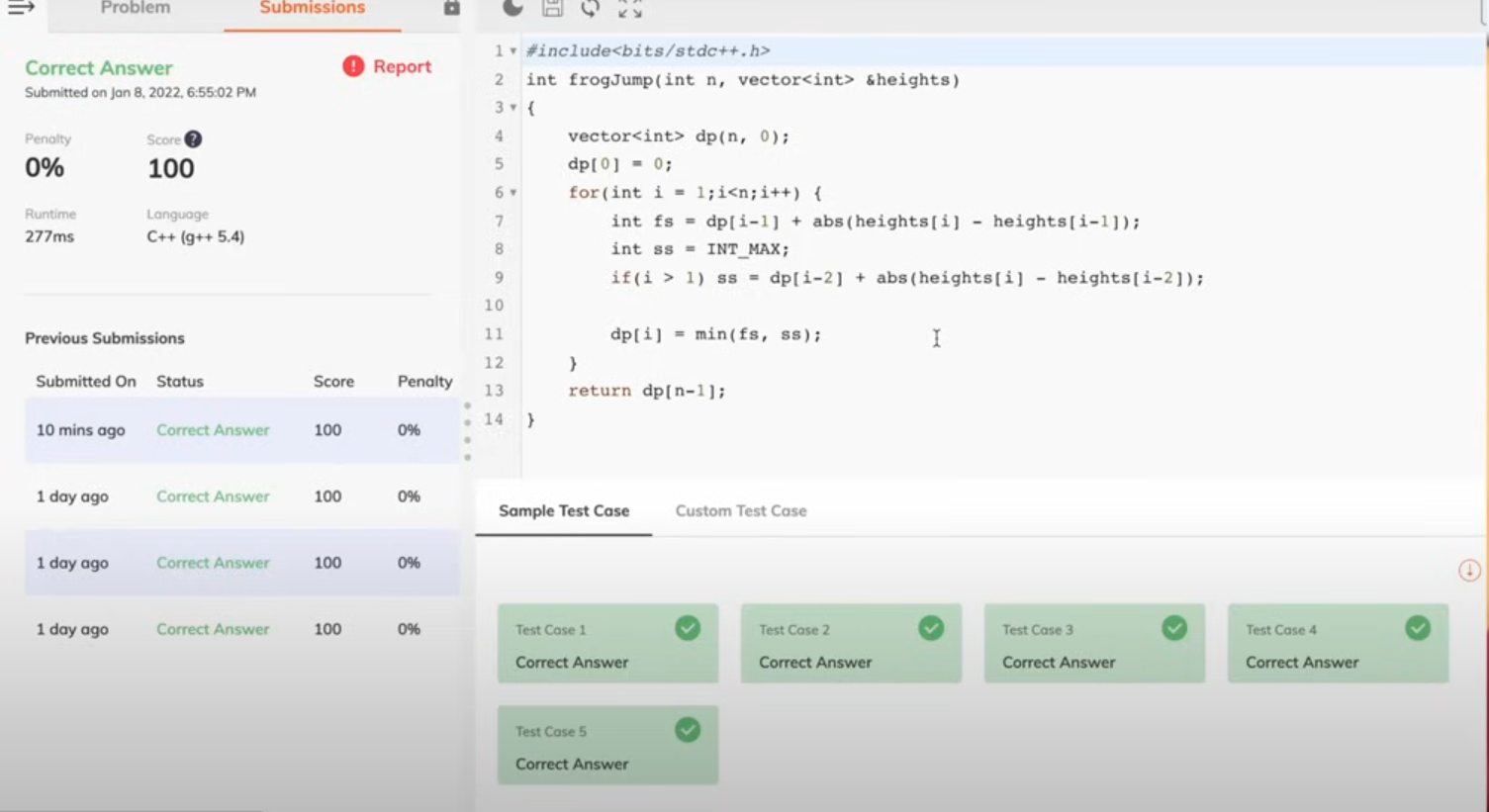
We are avoiding the overlapping sub problems, using the dp array.



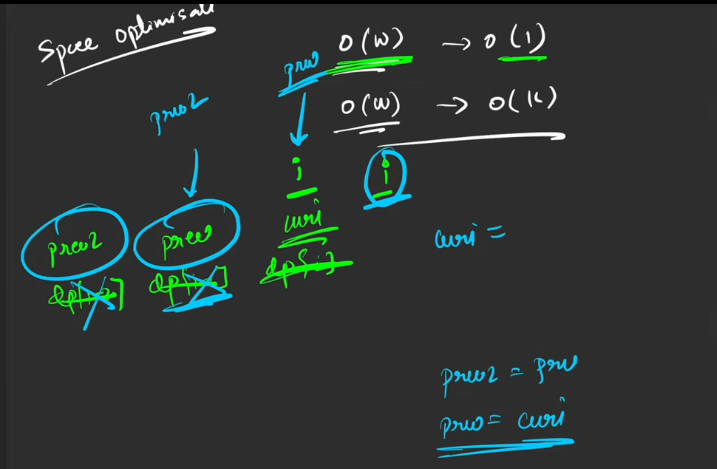
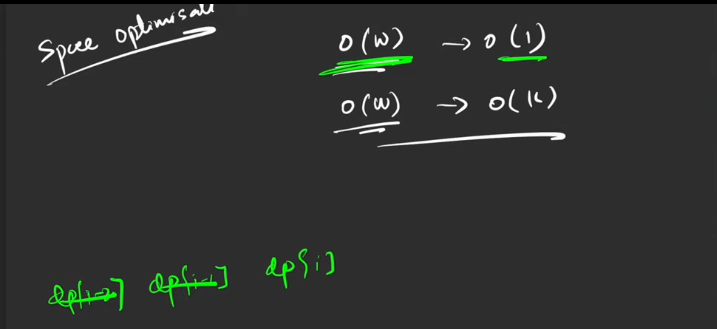
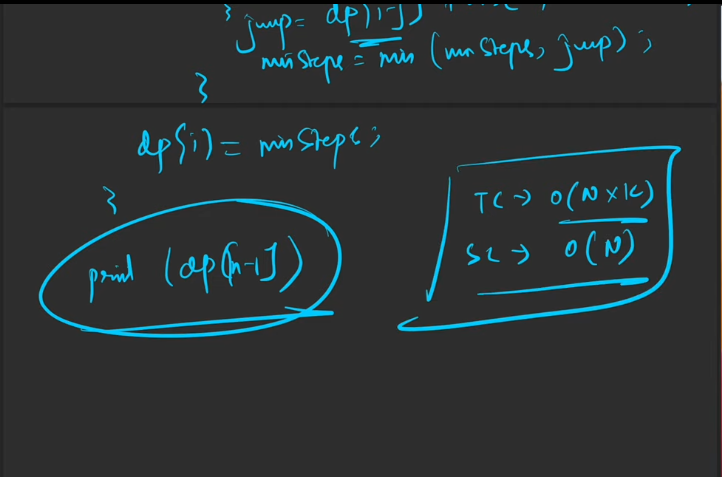
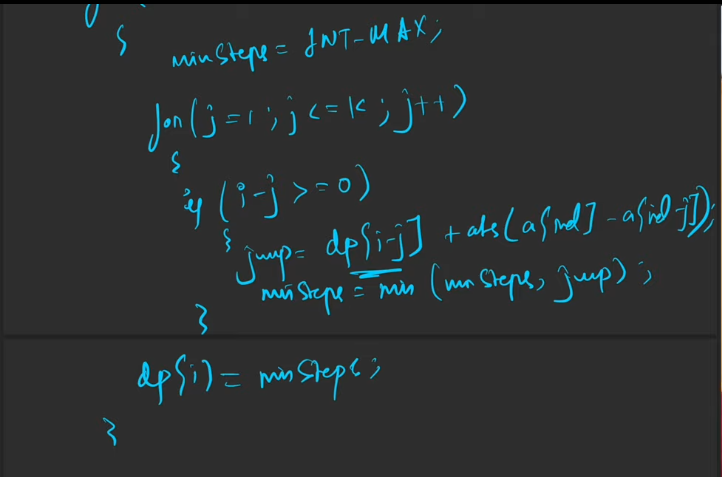
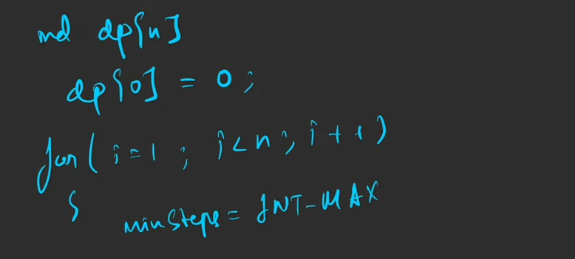
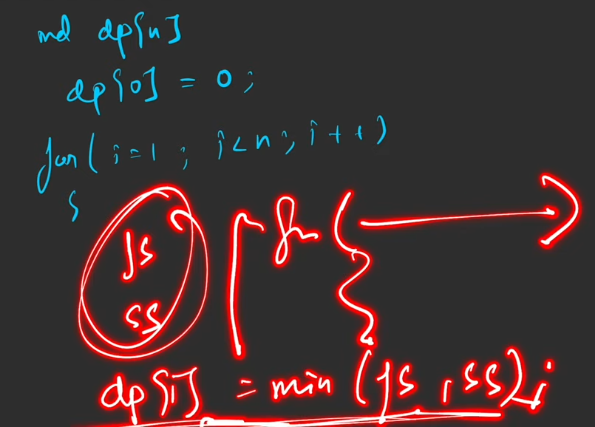
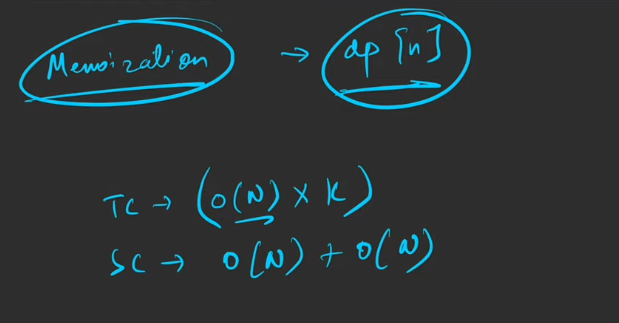
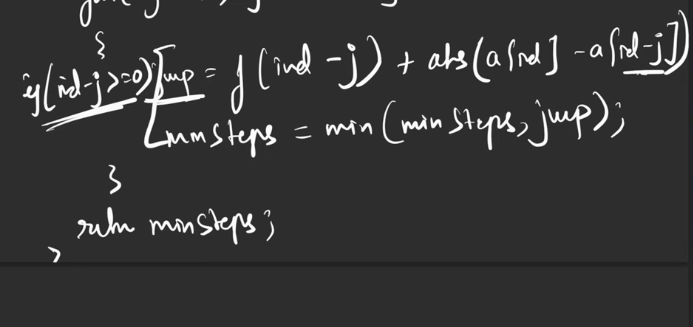
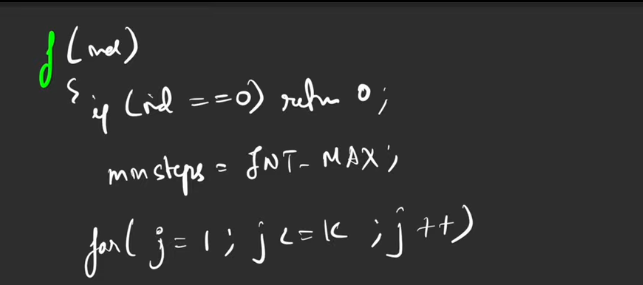
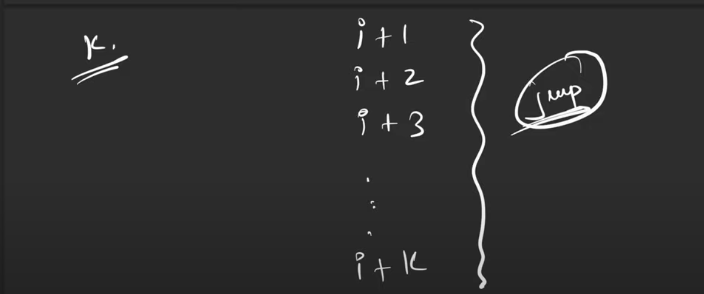
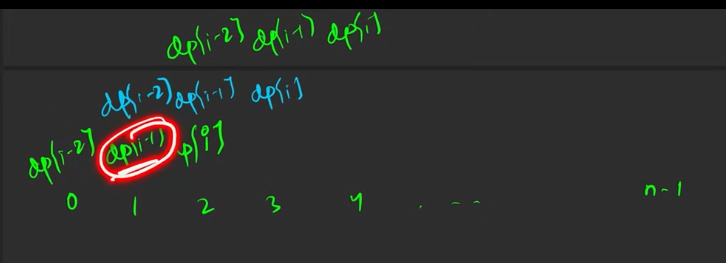
It is simple reccurssion solution.



This is created when the recurssion can be converted to tabulation.

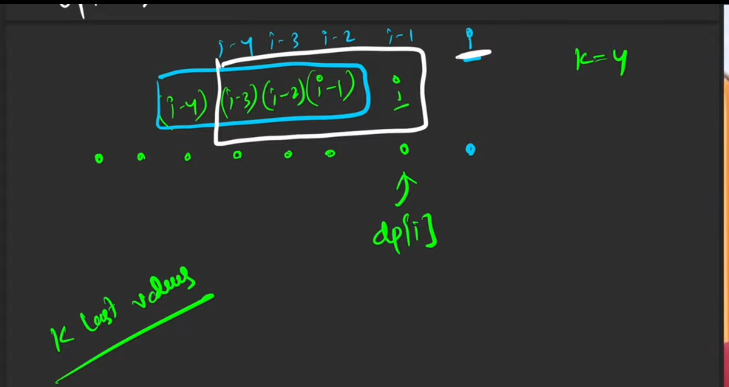


Space optimisation

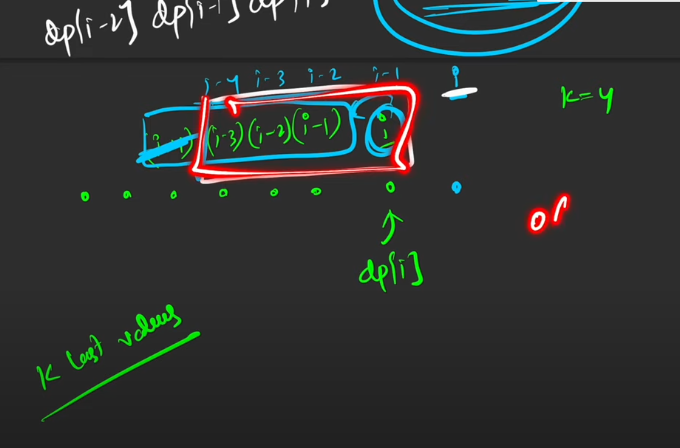


Suppose if I’m going to calculate for the value of k=2, then I need the last 2 dp[i-m] for the computations with the space optimization.

Here if we have the k steps, then we need that many number of dp[i-k] to proceed for calculation.



This is not given for the space optimisatio , since the space optimisation is itlself is not required by any chance. Ex: if w consider n steps, instead of k steps, in that situation, the space otimiastion itself not needed,since the space complexity will be almost the original number of elements. O(N).



**MAXIMUM SUM OF NON ADJACENT ELEMENTS**

