15/01/2024

DYNAMIC PROGRAMMING
CLASS - 5

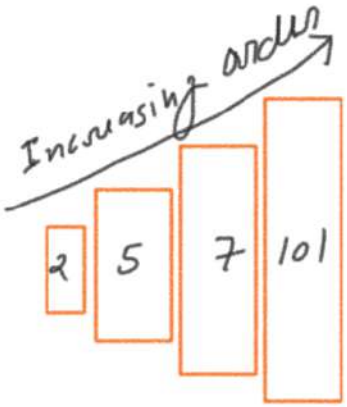
1. Longest Increasing Subsequence (Leetcode-300)

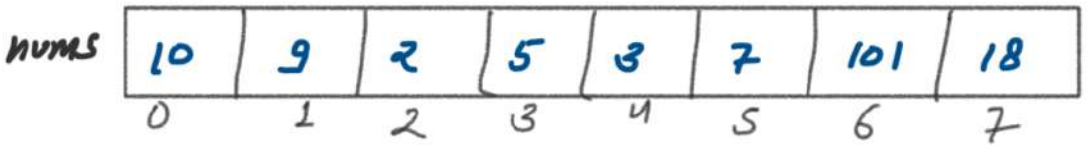
Including and excluding pattern

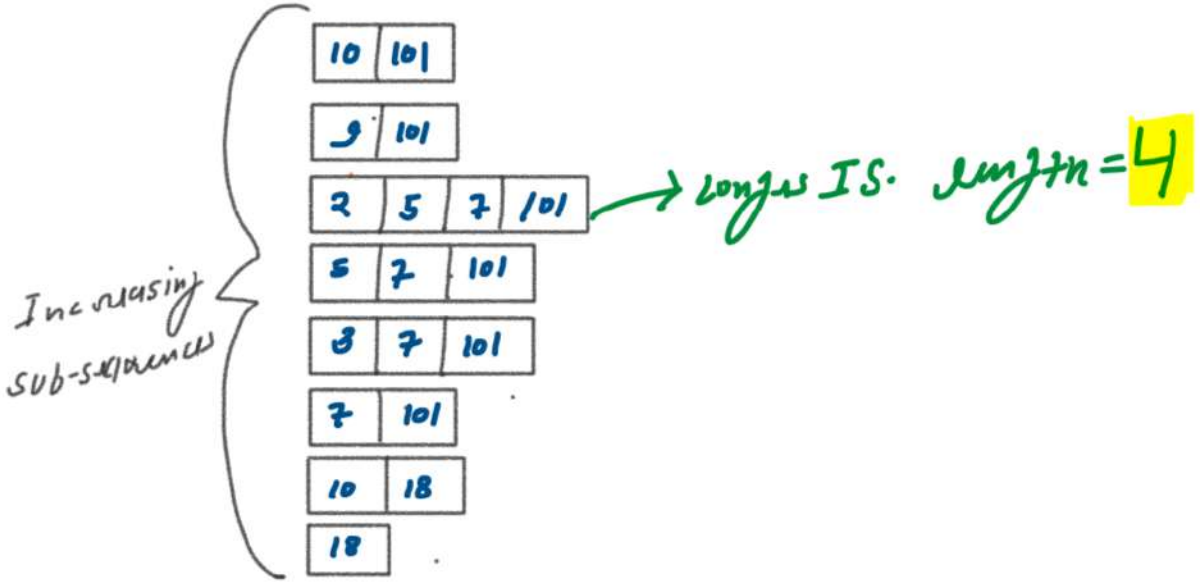
Example 1:

Input: nums = [10,9,2,5,3,7,101,18]

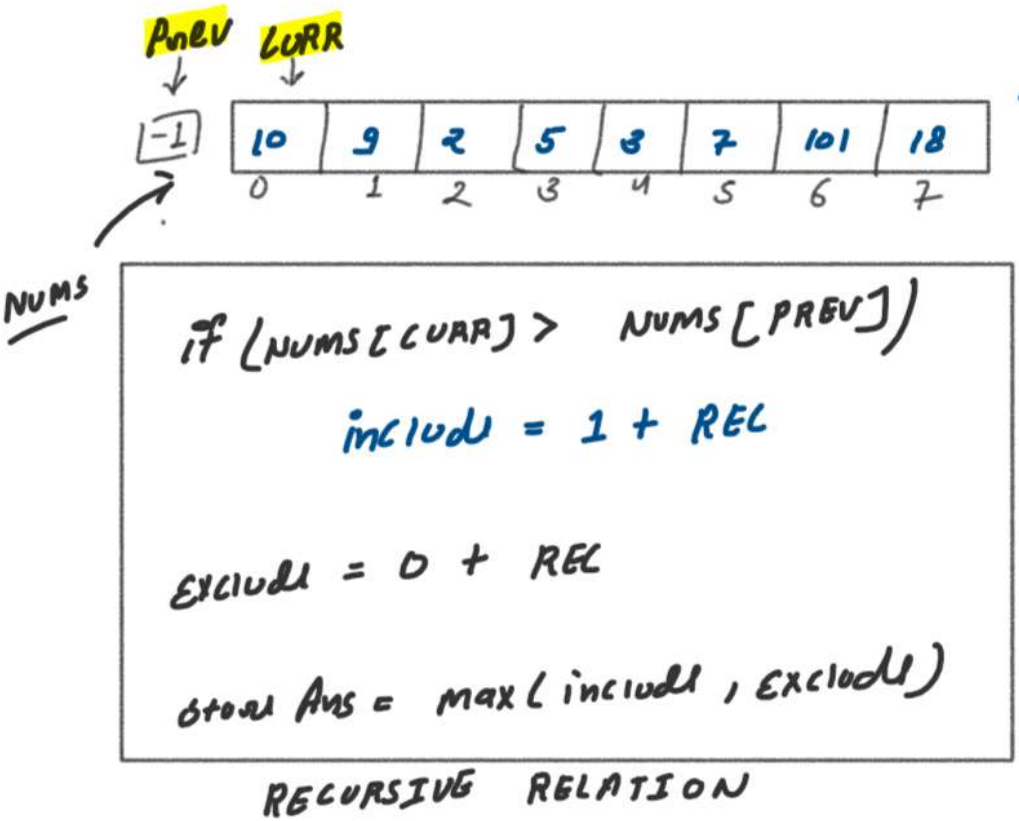
Output: 4





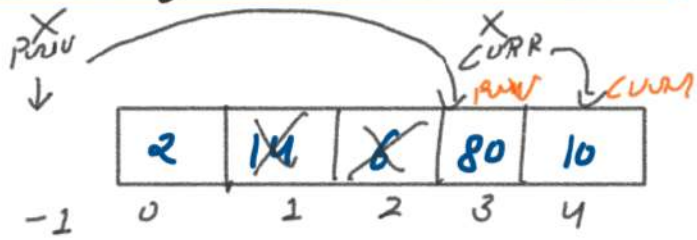


Approach 1: Recursion



```cpp
// 1. Longest Increasing Subsequence (Leetcode-300)
// Approach 1: Normal Recursion Approach

class Solution {
public:
    int solveUsingRec(vector<int> &nums, int curr, int prev){
        // Base case
        if(curr >= nums.size()){
            return 0;
        }

        // Recursive call
        int include = 0;
        if(prev == -1 || nums[curr] > nums[prev]){
            include = 1 + solveUsingRec(nums, curr+1, curr);
        }
        int exclude = 0 + solveUsingRec(nums, curr+1, prev);
        int ans = max(include, exclude);
        return ans;
    }
    int lengthOfLIS(vector<int>& nums) {
        int prev = -1;
        int curr = 0;
        int ans = solveUsingRec(nums, curr, prev);
        return ans;
    }
};
```

(Prev+1) pass Kyun Nahi Kiya Hai?



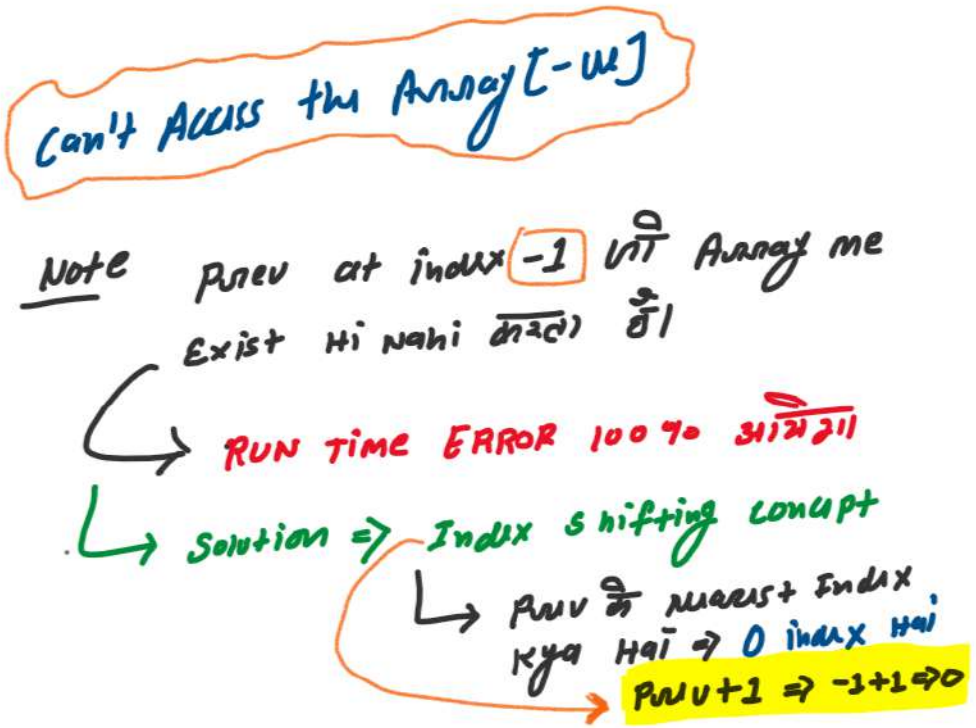
When CURR Index = 3 and Prev Index = Prev+1
= -1 + 1
= 0

Jabki Prev Index Kya Hona Chahiye → Prev Index = CURR = 3

CURR Index = 4

Approach 2: Top Down

```cpp
// 1. Longest Increasing Subsequence (Leetcode-300)
// Approach 2: Top Down Approach
// ❌Runtime Error Due to Invalid Array Index (Array Index Out of Bound)
// ✅How to resolve this error: using index shifting concept

class Solution {
public:
    int solveUsingMemo(vector<int> &nums, int curr, int prev,  vector<vector<int>> &dp){
        // Base case
        if(curr >= nums.size()){
            return 0;
        }

        if(dp[curr][prev] != -1){
            return dp[curr][prev];
        }

        // Recursive call
        int include = 0;
        if(prev == -1 || nums[curr] > nums[prev]){
            include = 1 + solveUsingMemo(nums, curr+1, curr, dp);
        }
        int exclude = 0 + solveUsingMemo(nums, curr+1, prev, dp);
        dp[curr][prev] = max(include, exclude);
        return dp[curr][prev];
    }
    int lengthOfLIS(vector<int>& nums) {
        int prev = -1;
        int curr = 0;
        int n = nums.size();
        vector<vector<int>> dp(n+1, vector<int> (n+1, -1));
        int ans = solveUsingMemo(nums, curr, prev, dp);
        return ans;
    }
};
```



```cpp
// 1. Longest Increasing Subsequence (Leetcode-300)
// Approach 2: Top Down Approach
// Error is Resolved using index shifting concept

class Solution {
public:
    int solveUsingMemo(vector<int> &nums, int curr, int prev,  vector<vector<int>> &dp){
        // Base case
        if(curr >= nums.size()){
            return 0;
        }

        if(dp[curr][prev+1] != -1){
            return dp[curr][prev+1];
        }

        // Recursive call
        int include = 0;
        if(prev == -1 || nums[curr] > nums[prev]){
            include = 1 + solveUsingMemo(nums, curr+1, curr, dp);
        }
        int exclude = 0 + solveUsingMemo(nums, curr+1, prev, dp);
        dp[curr][prev+1] = max(include, exclude);
        return dp[curr][prev+1];
    }
    int lengthOfLIS(vector<int>& nums) {
        int prev = -1;
        int curr = 0;
        int n = nums.size();
        vector<vector<int>> dp(n+1, vector<int> (n+1, -1));
        int ans = solveUsingMemo(nums, curr, prev, dp);
        return ans;
    }
};
```

step3 (at `if(dp[curr][prev+1] != -1)`)

step2 (at `// Recursive call`)

step1 (at `vector<vector<int>> dp(n+1, vector<int> (n+1, -1));`)

Include Condition check करते Time nums[prev] kyu liya hai → yanha v to ERROR AA skta tha

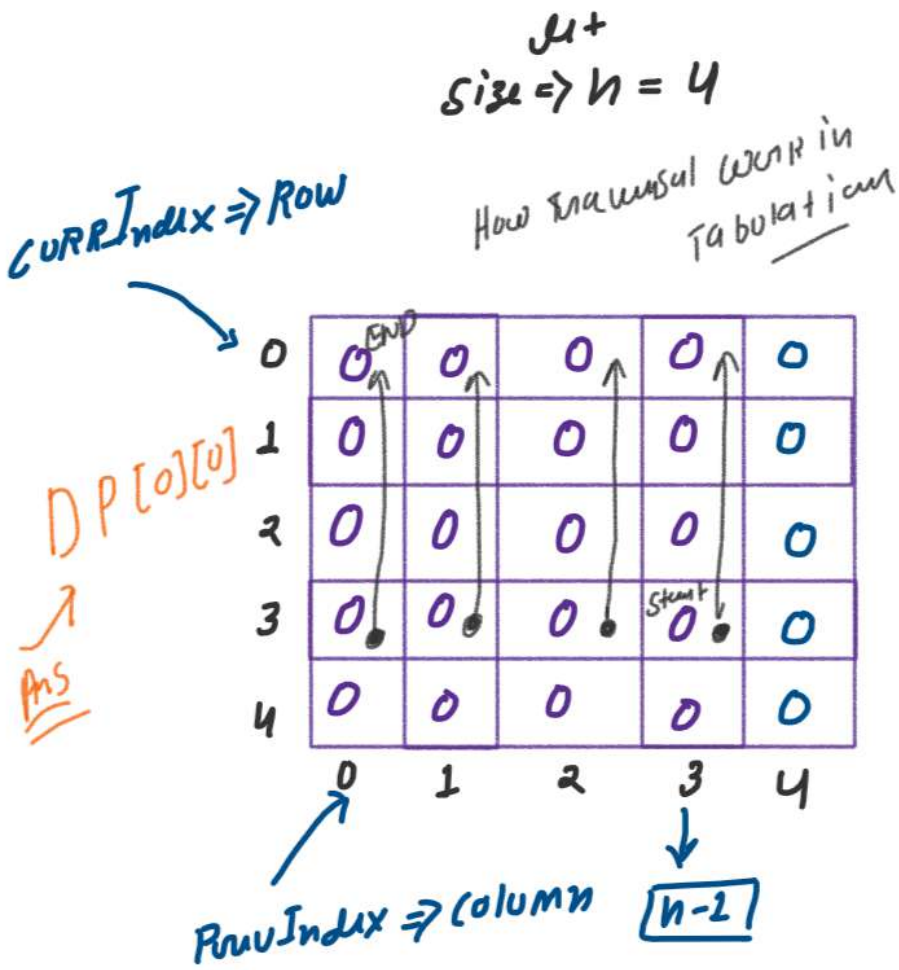
→ नहीं आ सकता है kyonki

[jab prev == -1
↳ to code Direct execute ho jayega.]

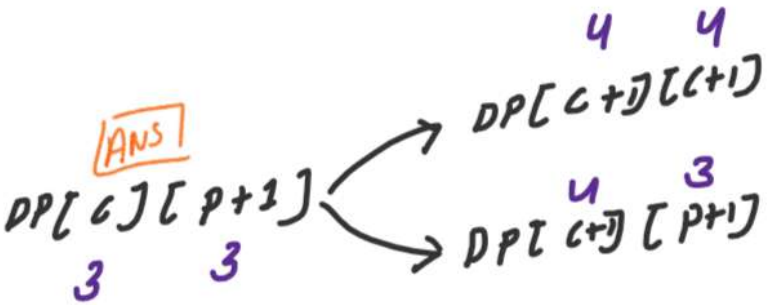
Approach 3: Bottom Up

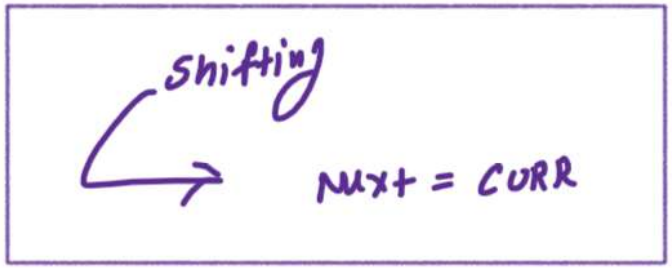
```cpp
// 1. Longest Increasing Subsequence (Leetcode-300)
// Approach 3: Bottom-up

class Solution {
public:
    int solveUsingTabu(vector<int> &nums, int curr, int prev){
        // Step 1: create DP array
        // Step 2: fill initial data in DP array according to recursion base case
        int n = nums.size();
        vector<vector<int>> dp(n+1, vector<int> (n+1, 0));

        // Step 3: fill the remaining DP array according to recursion formula/logic
        for(int currIndex = n-1; currIndex >= 0; currIndex--){
            for(int prevIndex = currIndex-1; prevIndex >= -1; prevIndex--){
                // Recursive call
                int include = 0;
                if(prevIndex == -1 || nums[currIndex] > nums[prevIndex]){
                    include = 1 + dp[currIndex+1][currIndex+1];
                }
                int exclude = 0 + dp[currIndex+1][prevIndex+1];
                dp[currIndex][prevIndex+1] = max(include, exclude);
            }
        }
        // Return ans
        return dp[0][0];
    }
    int lengthOfLIS(vector<int>& nums) {
        int prev = -1;
        int curr = 0;
        int ans = solveUsingTabu(nums, curr, prev);
        return ans;
    }
};
```

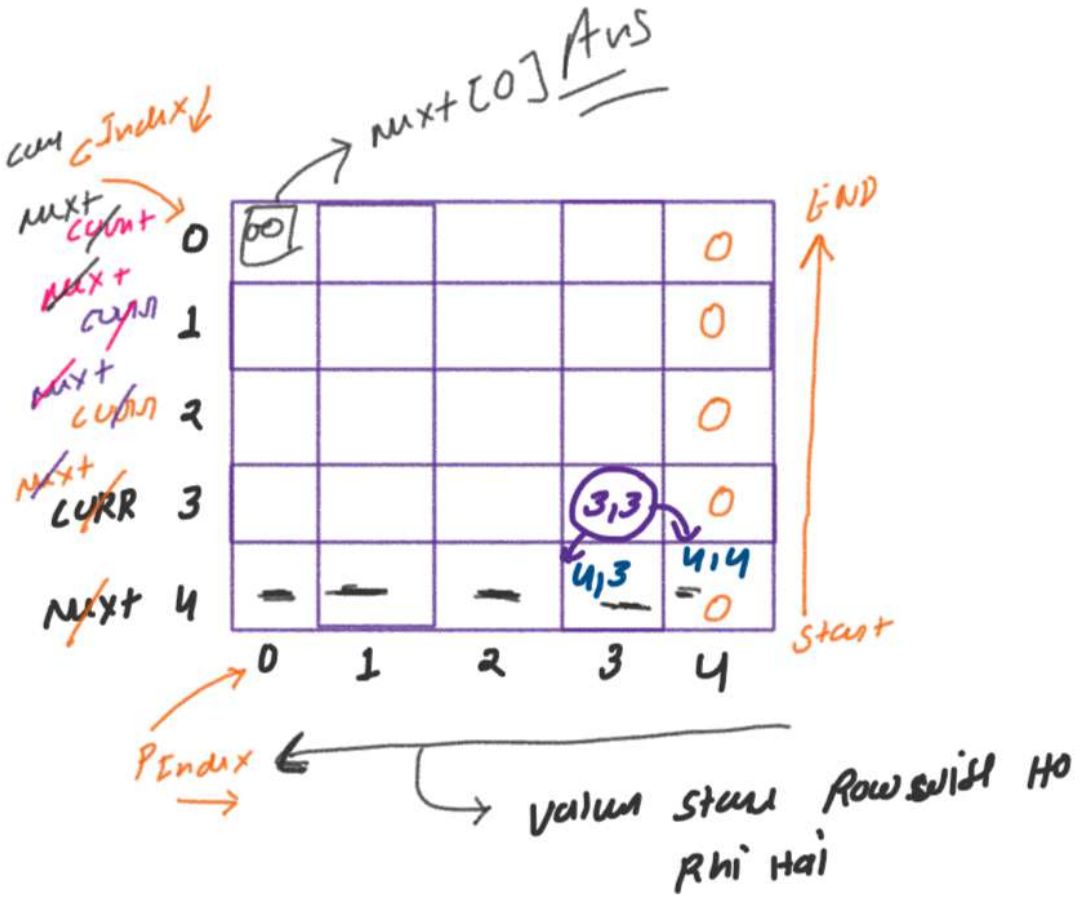
Index shifting



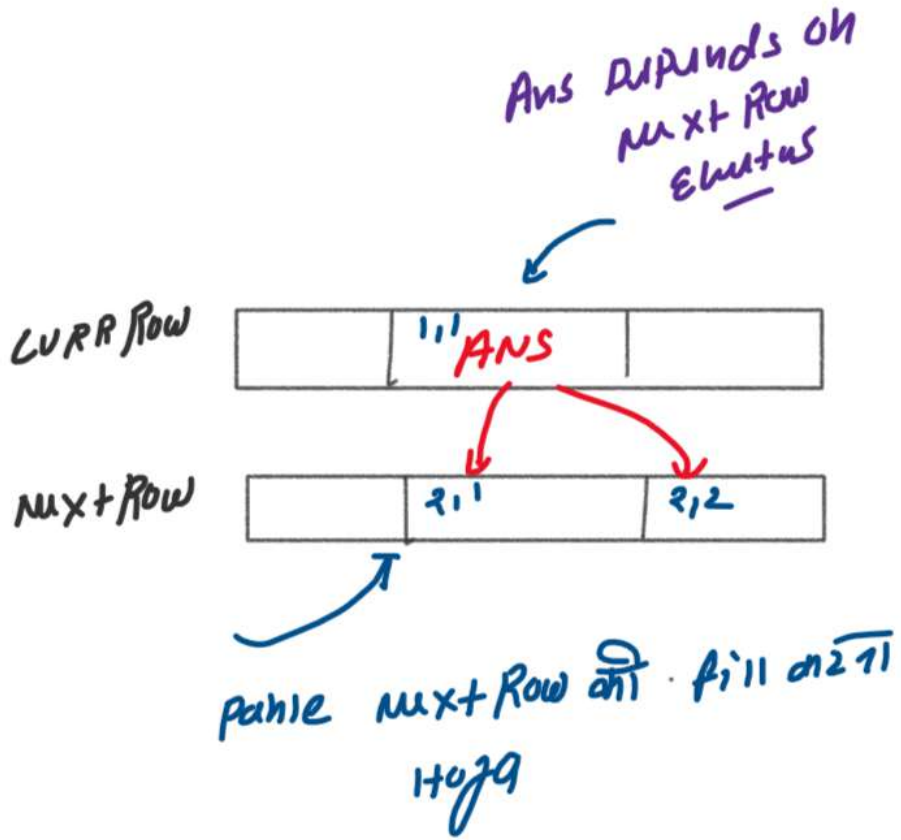
Approach 4: Space Optimization



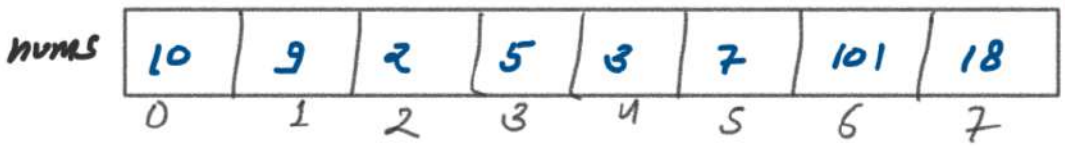




```cpp
// 1. Longest Increasing Subsequence (Leetcode-300)
// Approach 4: Space Optimization
// Without inter changing loop
class Solution {
public:
    int solveUsingTabuOS(vector<int> &nums){
        int n = nums.size();
        vector<int> currRow (n+1, 0);
        vector<int> nextRow (n+1, 0);

        // Loop row wise hi chalana hai mujhe
        for(int currIndex = n-1; currIndex >= 0; currIndex--){
            for(int prevIndex = currIndex-1; prevIndex >= -1; prevIndex--){
                // Recursive call
                int include = 0;
                if(prevIndex == -1 || nums[currIndex] > nums[prevIndex]){
                    include = 1 + nextRow[currIndex+1];
                }
                int exclude = 0 + nextRow[prevIndex+1];
                currRow[prevIndex+1] = max(include, exclude);
            }
            // Shift Karna Bhool Jata hu
            nextRow = currRow;
        }
        return nextRow[0];
    }
    int lengthOfLIS(vector<int>& nums) {
        int ans = solveUsingTabuOS(nums);
        return ans;
    }
};
```

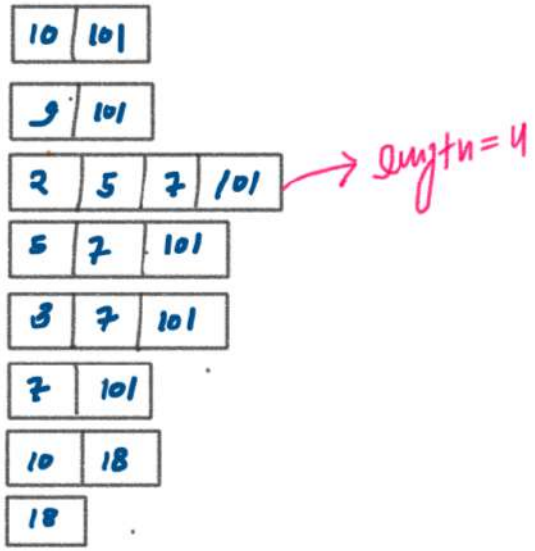


Approach 5: Solve Using Binary Search

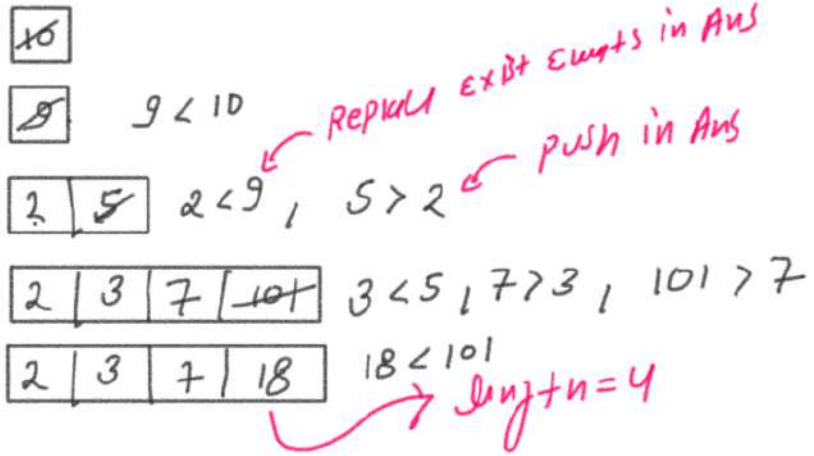


Array par Binary search nahi apply ho sakta hu but jo Brute force se Ans mil rha hai wo ek sorted list ki length hai to B.S. Ans par apply ho sakta hai.

B.F.



B.S.



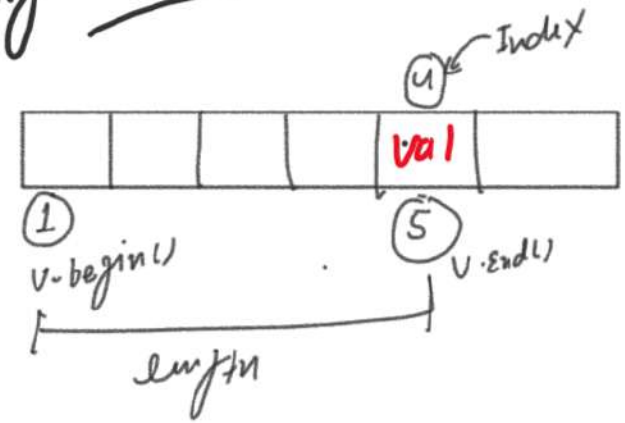
Lowerbound and upperbound concept

① lower_bound of 5 ⇒ 4 index

② upper_bound of 5 ⇒ 6 index

③ lower_bound of 100 ⇒ size of array ⇒ 8
upper_bound of 100 ⇒

Why this line



lower_bound(v.begin(), v.End, val) - v.begin()

val at Index ⇒ (5)-1

⇒ 4

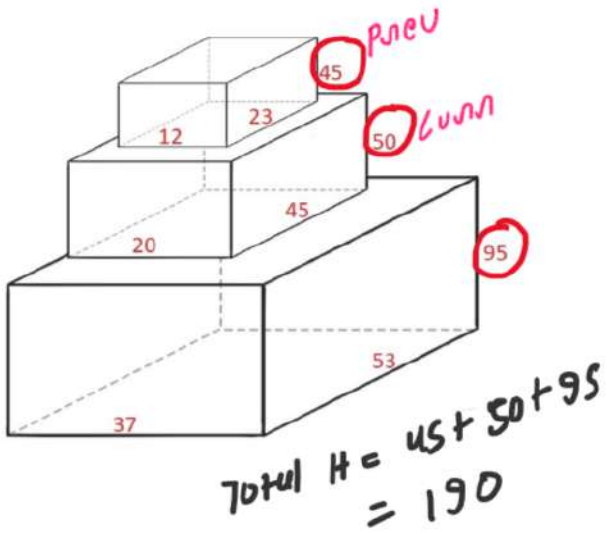
```cpp
// 1. Longest Increasing Subsequence (Leetcode-300)
// Approach 5: Solve Using Binary Search

class Solution {
public:
    int solveUsingBS(vector<int> &nums){
        vector<int> ans;
        // initial state
        ans.push_back(nums[0]);

        for(int i=1; i<nums.size(); i++){
            if(nums[i] > ans.back()){
                // push new element when it is greater then of last element of ans
                ans.push_back(nums[i]);
            }
            else{
                // last element of ans is just greater then of new element to replace krdo
                int index = lower_bound(ans.begin(), ans.end(), nums[i]) - ans.begin();
                // replace last element of ans
                ans[index] = nums[i];
            }
        }
        // return length of ans
        return ans.size();
    }
    int lengthOfLIS(vector<int>& nums) {
        int ans = solveUsingBS(nums);
        return ans;
    }
};
```

2. Maximum Height by Stacking Cuboids (Leetcode-1691)

Including and excluding pattern

Note: You can rearrange any cuboid's dimensions by rotating it to put it on another cuboid.
Ans: Return the maximum height of the stacked cuboids.

Example 1:
Input: cuboids = [[50,45,20],[95,37,53],[45,23,12]]
Output: 190



Total H = 45 + 50 + 95
= 190

Iska matlb kya hai

Cuboids

	W	L	H
0	20	45	50
1	37	53	95
2	12	23	45

SORT each cuboid

Cuboids

	W	L	H
0	12	23	45
1	20	45	50
2	37	53	95

SORT Cuboids

```cpp
// 2. Maximum Height by Stacking Cuboids (Leetcode-1691)
// Approach 5: Solve Using Binary Search
// This question based on Longest Increasing Subsequence (Leetcode-300)

class Solution {
public:

    bool check(vector<int> curr , vector<int> prev){
        if(curr[0] >= prev[0] && curr[1] >= prev[1] && curr[2] >= prev[2]){
            return true;
        }
        else{
            return false;
        }
    }

    int solveUsingTabuOS(vector<vector<int>>& cuboids){
        int n = cuboids.size();
        vector<int> currRow (n+1, 0);
        vector<int> nextRow (n+1, 0);

        // Loop row wise hi chalana hai mujhe
        for(int currIndex = n-1; currIndex >= 0; currIndex--){
            for(int prevIndex = currIndex-1; prevIndex >= -1; prevIndex--){
                // Recursive call
                int include = 0;
                if(prevIndex == -1 || check(cuboids[currIndex], cuboids[prevIndex])){
                    int heightAdded = cuboids[currIndex][2];
                    include = heightAdded + nextRow[currIndex+1];
                }
                int exclude = 0 + nextRow[prevIndex+1];
                currRow[prevIndex+1] = max(include, exclude);
            }
            // Shift Karna Bhool Jata hu
            nextRow = currRow;
        }
        return nextRow[0];
    }

    int maxHeight(vector<vector<int>>& cuboids) {
        // Sort each cuboid (mene rotate kar diya hai)
        for(auto &cuboid: cuboids){
            sort(cuboid.begin(), cuboid.end());
        }
        // Sort cuboids array
        sort(cuboids.begin(), cuboids.end());
        int ans = solveUsingTabuOS(cuboids);
        return ans;
    }
};
```

→ with & Reb SORT Karna zaroori Hai Otherwise Answer Original Nahi Hoga

This function tells us
↳ Ek Box ke uppar Dusra Box Kaisa Rakha jaaye

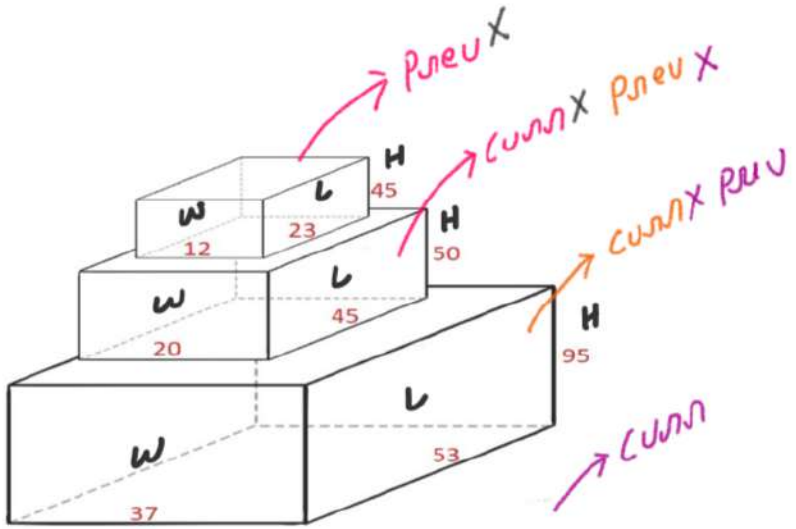
Cuboids

12	23	45	Prev
20	45	50	Curr
37	53	95	

↳ Height kab Add. Kar Sakta Hun

↳ yab (CURR wala dabba Bada Hoga Prev wale dabbe se)

(Wc 20 > Wp 12
Lc 45 > Lp 23
Hc 50 > Lp 45)



Output
↳ return nextRow[0]

3. Russian Doll Envelopes (Leetcode-354)

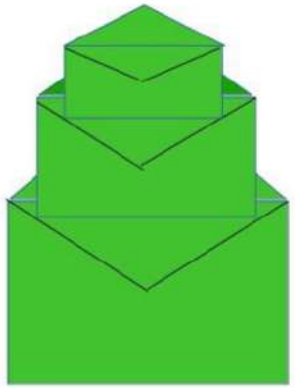
Including and excluding pattern

Problem Statement:
You are given a 2D array of integers envelopes where **envelopes[i] = [wi, hi]** represents the width and the height of an envelope.

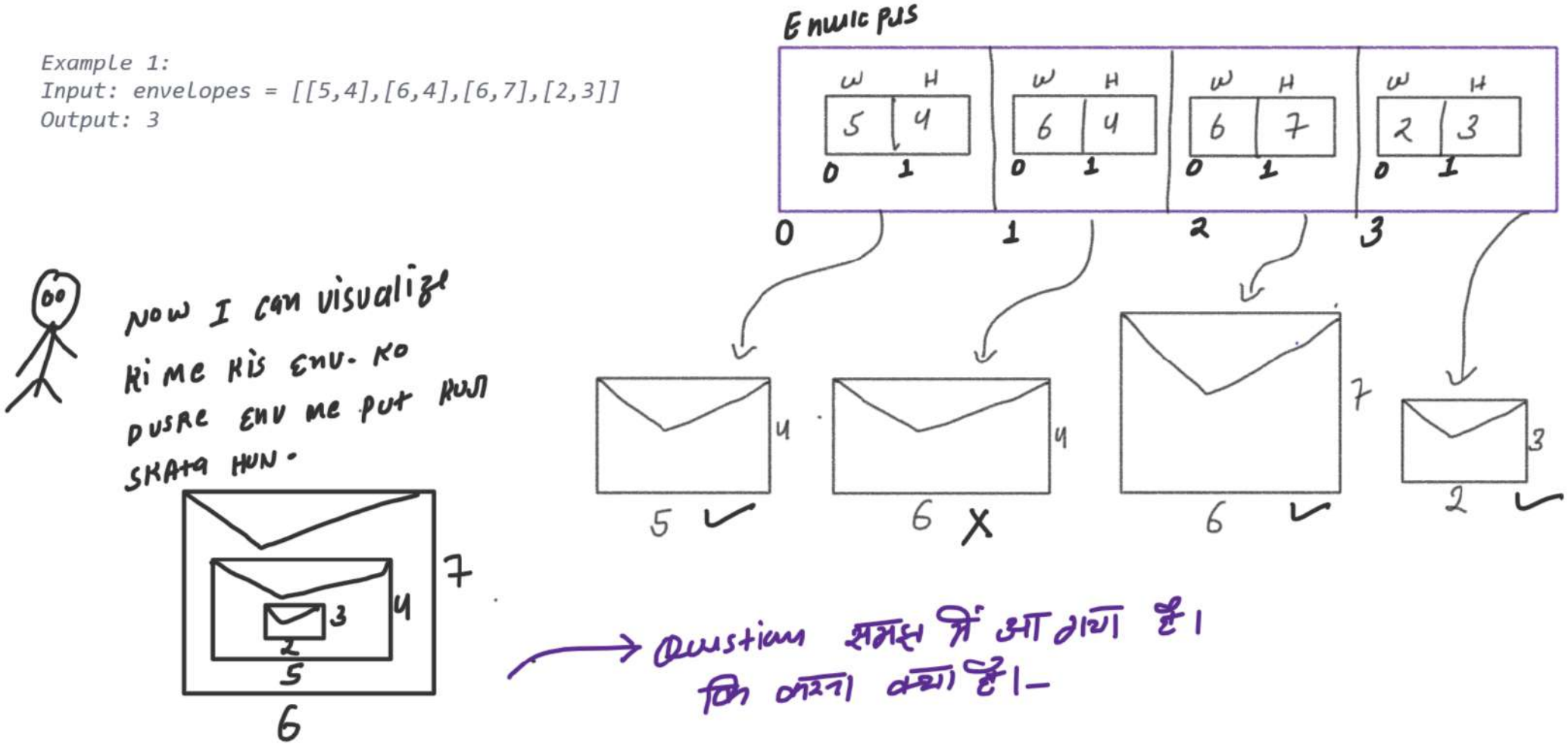
One envelope can fit into another **if and only if both the width and height of one envelope are greater than the other envelope's width and height.**

Return the **maximum number of envelopes** you can Russian doll (i.e., put one inside the other).

Note: You cannot rotate an envelope.



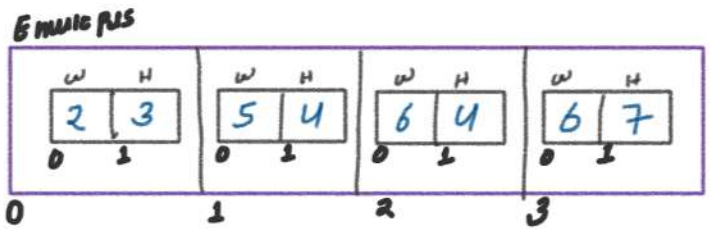
Example 1:
Input: envelopes = [[5,4],[6,4],[6,7],[2,3]]
Output: 3

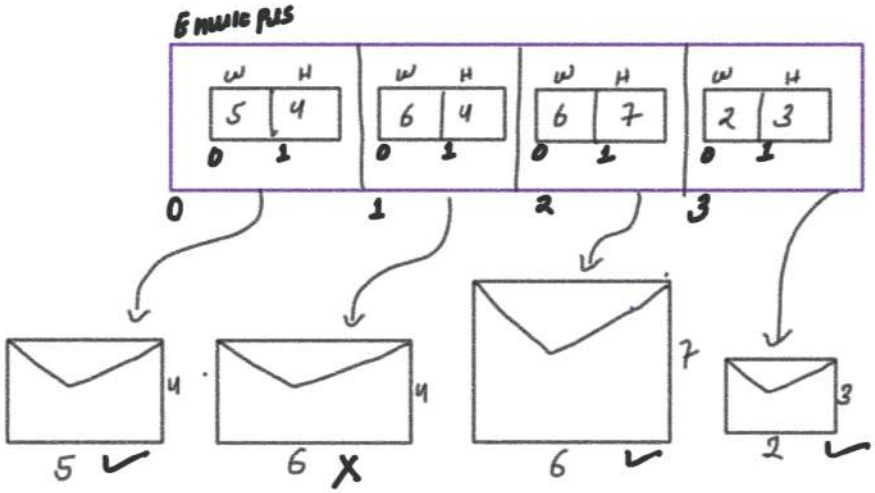


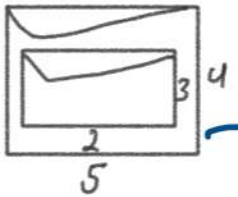
APPROACH

```
Input: envelopes = [[5,4],[6,4],[6,7],[2,3]]
```

SORT THE ENV in increasing order by width





① $2 < 5$ and $3 < 4$ → 

② $5 < 6$ and $4 \overset{\times}{<} 4$ → can't put

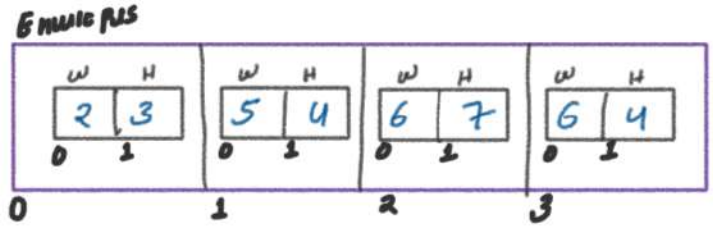
③ $6 < 6$ and $4 < 7$ → can't put

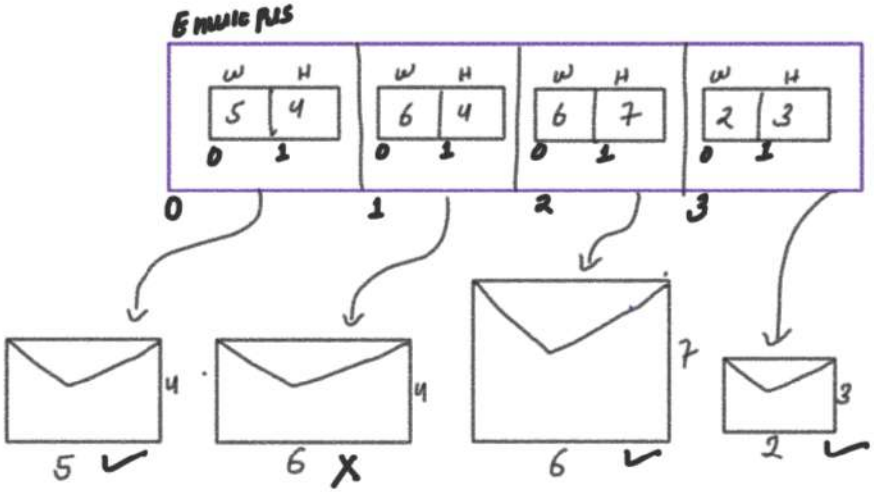
→ output = 2 WRONG ANS

→ SORT ENV in increasing order by width

→ and sort ENV in decreasing order by height

when $w_i == w_j$

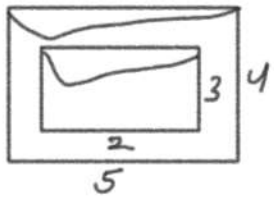


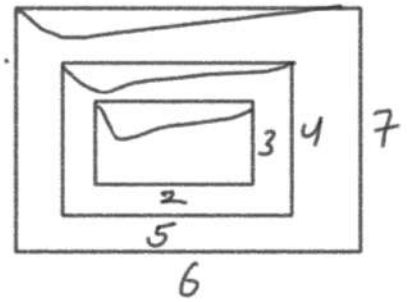


① $2 < 5$ and $3 < 4$ → put

② $5 < 6$ and $4 < 7$ → put

③ $6 \overset{\times}{<} 6$ and $7 < 4$ → cont + put

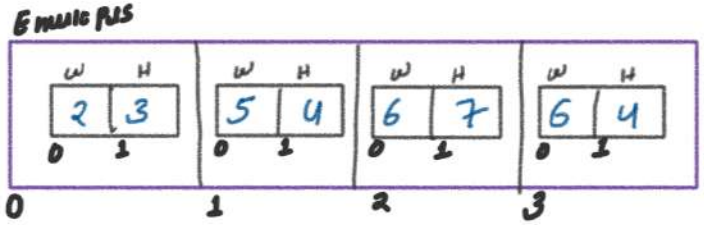


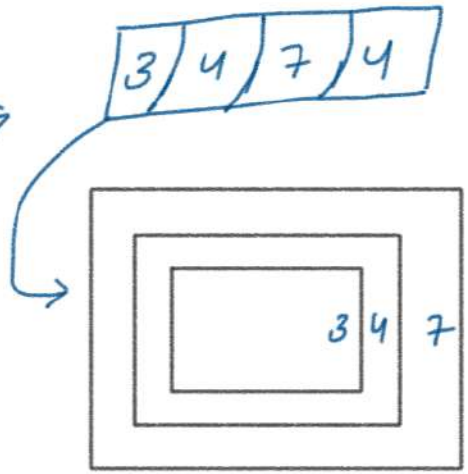


output = 3

Right Ans

⇒ Kya me only Height wiki Longest Increasing subsequence Pattern Apply karu sirf HUN





① 3<4 put

② 4<7 put

③ 7<4 ✗ can't put

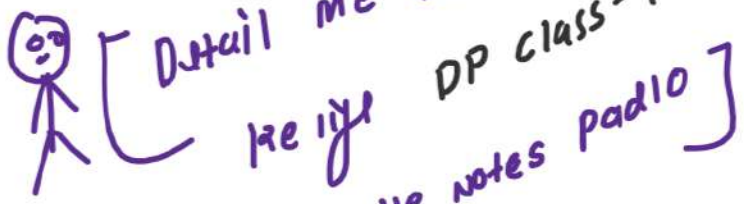
```cpp
// 3. Russian Doll Envelopes (Leetcode-354)
// Approach 5: Solve Using Binary Search
// No TLE

class Solution {
public:
    // Own Comprator
    static bool com(vector<int> a , vector<int> b){
        if(a[0] == b[0]){
            // if Wi == Wj then sort envelop in decreasing order by height
            return a[1] > b[1];
        }
        // sort envelope in increasing order by width
        return a[0] < b[0];
    }

    // Apply longest increasing subsequence pattern on only height of envelop
    int solveUsingBS(vector<vector<int>> &envelopes){
        // Sort envelope in increasing order by width
        // and if Wi == Wj then sort envelop in decreasing order by height
        sort(envelopes.begin(), envelopes.end(), com);

        vector<int> ans;
        // initial state
        ans.push_back(envelopes[0][1]);

        for(int i=1; i<envelopes.size(); i++){
            if(envelopes[i][1] > ans.back()){
                // push new element when it is greater then of last element of ans
                ans.push_back(envelopes[i][1]);
            }
            else{
                // last element of ans is just greater then of new element to replace krdo
                int index = lower_bound(ans.begin(), ans.end(), envelopes[i][1]) - ans.begin();
                // replace last element of ans
                ans[index] = envelopes[i][1];
            }
        }
        // return length of ans
        return ans.size();
    }
    int maxEnvelopes(vector<vector<int>>& envelopes) {
        int ans = solveUsingBS(envelopes);
        return ans;
    }
};
```

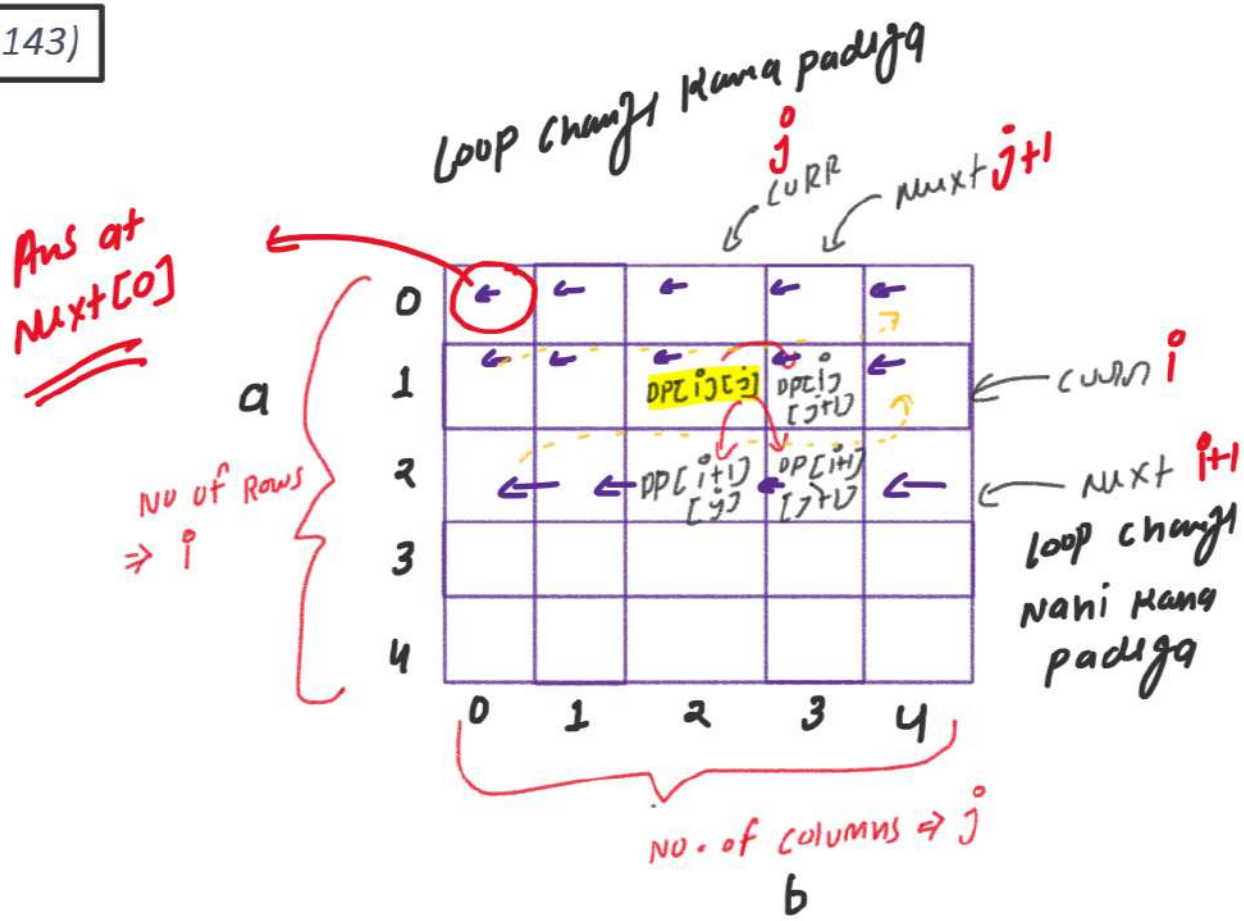
Using Binary search
↳ NO TLE

```cpp
// 3. Russian Doll Envelopes (Leetcode-354)
// Approach 4: Space Optimization Approach
// TLE

class Solution {
public:
        bool check(vector<int> curr , vector<int> prev){
        // Curr - bade wale dibbe ko represent krta hai
        // Prev - chotte wale dibbe ko represent krta hai
        if(curr[0] > prev[0] && curr[1] > prev[1]){
            return true;
        }
        else{
            return false;
        }
    }
    int solveUsingTabuOS(vector<vector<int>>& envelopes){
        int n = envelopes.size();
        vector<int> currRow (n+1, 0);
        vector<int> nextRow (n+1, 0);

        // Loop row wise hi chalana hai mujhe
        for(int currIndex = n-1; currIndex >= 0; currIndex--){
            for(int prevIndex = currIndex-1; prevIndex >= -1; prevIndex--){
                // Recursive call
                int include = 0;
                if(prevIndex == -1 || check(envelopes[currIndex], envelopes[prevIndex])){
                    include = 1 + nextRow[currIndex+1];
                }
                int exclude = 0 + nextRow[prevIndex+1];
                currRow[prevIndex+1] = max(include, exclude);
            }
            // Shift Karna Bhool Jata hu
            nextRow = currRow;
        }
        return nextRow[0];
    }

    int maxEnvelopes(vector<vector<int>>& envelopes) {
        sort(envelopes.begin(), envelopes.end());
        int ans = solveUsingTabuOS(envelopes);
        return ans;
    }
};
```

Tabulation space optimization solution
↳ TLE

1. Longest Common Subsequence (Leetcode-1143)





```cpp
// 1. Longest Common Subsequence (Leetcode-1143)
// Approach 4: Space Optimization
// Without changing loop

class Solution {
public:
        int solveUsingTabuOSNoChangingLoop(string &a, string &b, int i, int j) {
        vector<int> curr (b.length()+1, 0);
        vector<int> next (b.length()+1, 0);

        for(int i = a.length()-1; i >= 0; i--){
            for(int j = b.length()-1; j >= 0; j--){
                // Recursive call
                int ans = 0;
                if(a[i] == b[j]) {
                    // Does match the subsequence character
                    ans = 1 + next[j+1];
                }
                else {
                    // Does not match the subsequence character
                    ans = 0 + max(curr[j+1], next[j]);
                }
                curr[j] = ans;
            }
            // shifting
            next = curr;
        }
        return next[0];
    }

    int longestCommonSubsequence(string text1, string text2) {
        int i = 0;
        int j = 0;
        int ans = solveUsingTabuOSNoChangingLoop(text1, text2, i, j);
        return ans;
    }
};
```

```cpp
// 1. Longest Common Subsequence (Leetcode-1143)
// Approach 4: Space Optimization
// With changing loop

class Solution {
public:
        int solveUsingTabuOS(string &a, string &b, int i, int j) {
        vector<int> curr (a.length()+1, 0);
        vector<int> next (a.length()+1, 0);

        for(int j = b.length()-1; j >= 0; j--){
            for(int i = a.length()-1; i >= 0; i--){
                // Recursive call
                int ans = 0;
                if(a[i] == b[j]) {
                    // Does match the subsequence character
                    ans = 1 + next[i+1];
                }
                else {
                    // Does not match the subsequence character
                    ans = 0 + max(next[i], curr[i+1]);
                }
                curr[i] = ans;
            }
            // Shift Karna Bhool Jata hu
            next = curr;
        }
        return next[0];
    }

    int longestCommonSubsequence(string text1, string text2) {
        int i = 0;
        int j = 0;
        int ans = solveUsingTabuOS(text1, text2, i, j);
        return ans;
    }
};
```