Dynamic Programming Lecture 1

Saturday, 7 September 2024 2:04 PM

https://www.hackerrank.com/challenges/coin-change/problem

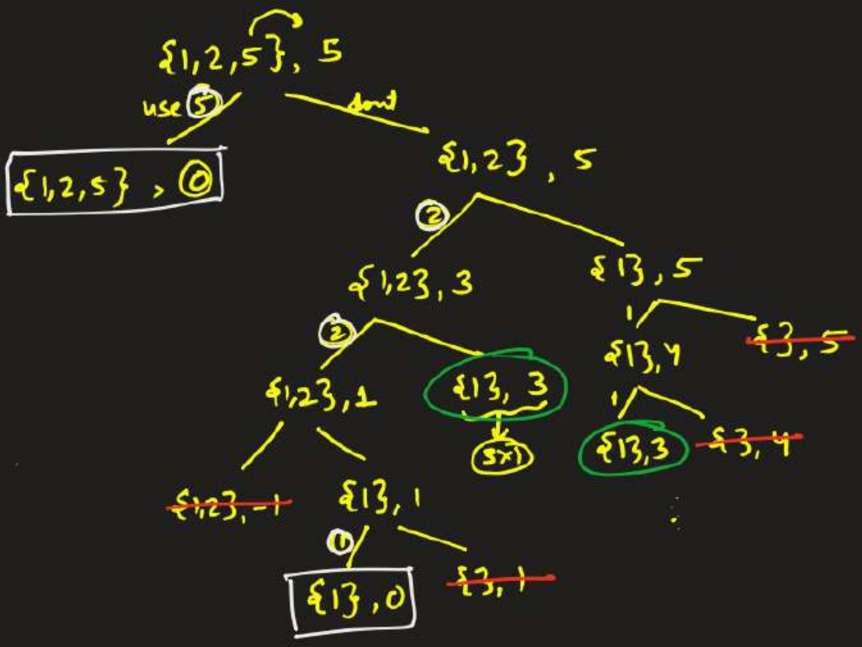
$$\text{count}\,(\overset{\substack{\text{currency}\\ \text{denomination}\\ \downarrow}}{\text{coins}},\ \overset{\substack{\text{index}\\ \downarrow}}{\underline{\underline{n}}},\ \text{sum})$$

$$= \text{count}\,(\underline{\underline{\text{coins}}},\ \underline{\underline{n}},\ \underline{\underline{\text{sum} - \text{coins}[n-1]}})$$

$$+$$

$$\text{count}\,(\text{coins},\ n-1,\ \text{sum})$$

$$T = O(n \cdot \text{sum})$$



$\underline{\underline{n}} = 10$

$$C = [\overset{0}{2}, \overset{1}{5}, \overset{2}{3}, \overset{3}{\boxed{6}}]$$

$n = 10$ $\qquad C = [\overset{0}{2}, \overset{1}{5}, \overset{2}{3}, \overset{3}{6}]$

① dp → state → { idx, amt }

idx → 0, 1, 2, 3

amt → 0, 1, 2 - - - 10

② ways (c, idx, ⓝ)

use → way (c, idx, n − c[idx])

dont use → ways(c, idx −1, n)

dp → (m × n)

m → length of c

n → amount

dp → idx × amt

idx → −1 → m−1 → 0 → m → m+1

amt → 0 → n → (n+1)

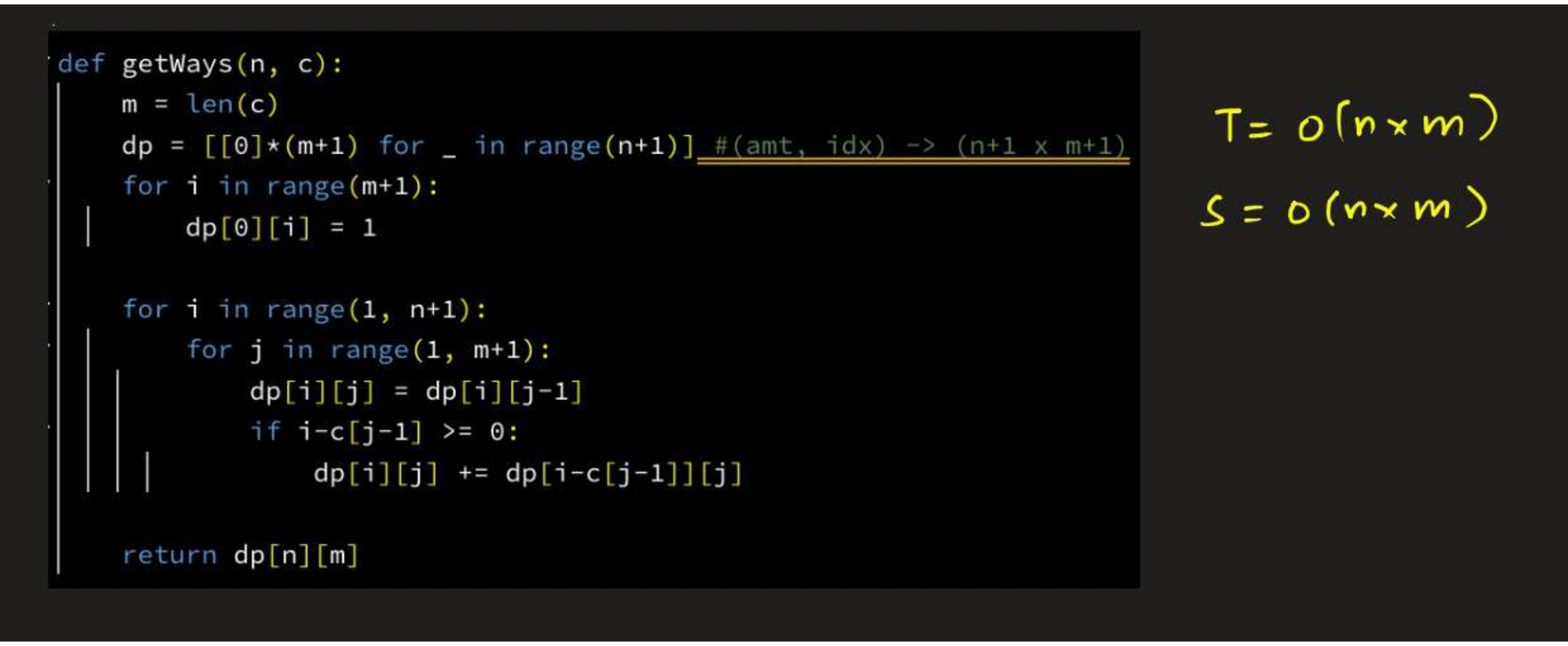
dp [m+1] [n+1]

③ Base case:-

idx = 0 < amt = 0 ✓ ① ways
idx = 0 < amt > 0 ⓪ ways

amt = 0 , idx ≠ 0 → ① way

④ Ans :- idx = m , amt = n



```
def getWays(n, c):
    m = len(c)
    dp = [[0]*(m+1) for _ in range(n+1)] #(amt, idx) -> (n+1 x m+1)
    for i in range(m+1):
        dp[0][i] = 1

    for i in range(1, n+1):
        for j in range(1, m+1):
            dp[i][j] = dp[i][j-1]
            if i-c[j-1] >= 0:
                dp[i][j] += dp[i-c[j-1]][j]

    return dp[n][m]
```

$T = O(n \times m)$

$S = O(n \times m)$

https://www.geeksforgeeks.org/problems/minimal-cost/1

$h = \{10, 30, 40, 50, 20\}$ $k = 3$

$|50-10| + |20-50| = 40 + 30 = \underline{70}$

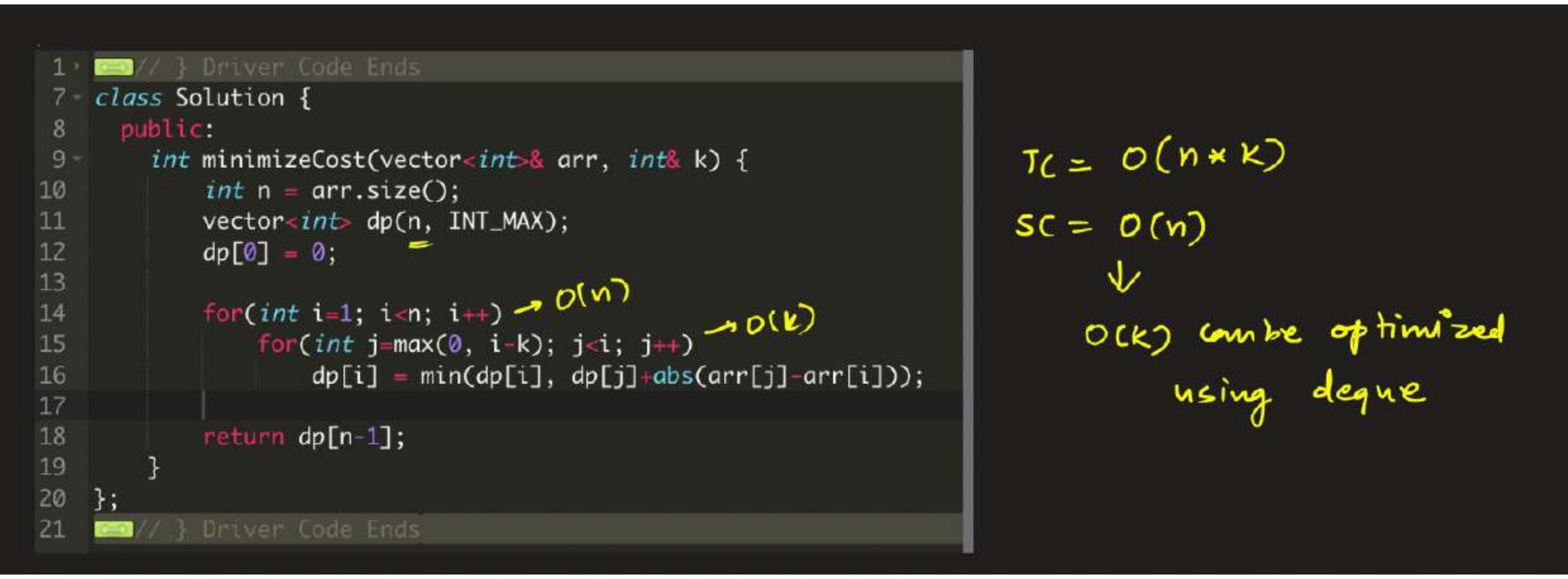
$|30-10| + |20-30| = 10 + 10 = \underline{\underline{20}}$

State :- pos^n $dp[pos^n] \rightarrow$ cost incurred when you come to pos^n

Recurrence Relation :- $dp[i] = \min\limits_{i-k \le j \le i-1} dp[j] + |h[j] - h[i]|$

Base Case :- $dp[0] = 0$

Ans :- $dp[n-1]$



```
// } Driver Code Ends
class Solution {
  public:
    int minimizeCost(vector<int>& arr, int& k) {
        int n = arr.size();
        vector<int> dp(n, INT_MAX);
        dp[0] = 0;

        for(int i=1; i<n; i++)
            for(int j=max(0, i-k); j<i; j++)
                dp[i] = min(dp[i], dp[j]+abs(arr[j]-arr[i]));

        return dp[n-1];
    }
};
// } Driver Code Ends
```

→ O(n)

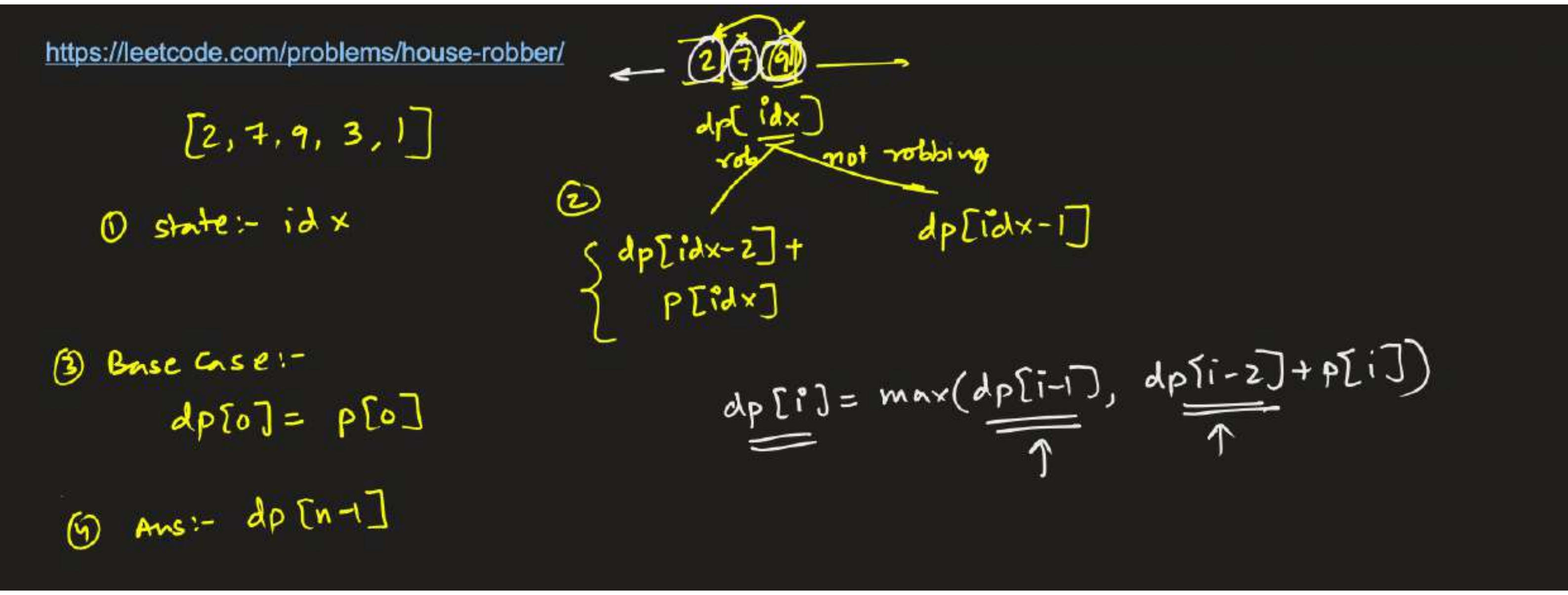
→ O(k)

TC = O(n * K)

SC = O(n)

↓

O(k) can be optimized using deque



https://leetcode.com/problems/house-robber/

[2, 7, 9, 3, 1]

① state:- idx

$dp[idx]$

②

rob: $dp[idx-2] + p[idx]$

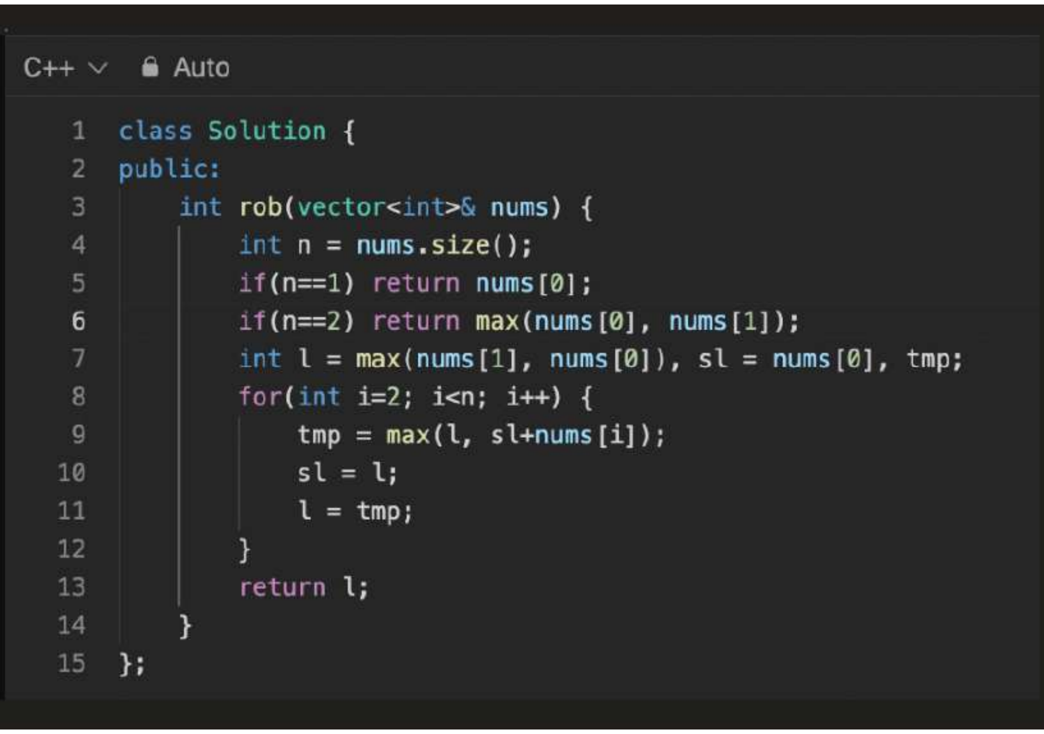
not robbing: $dp[idx-1]$

$$dp[i] = \max(dp[i-1],\ dp[i-2]+p[i])$$

③ Base case:-

$dp[0] = p[0]$

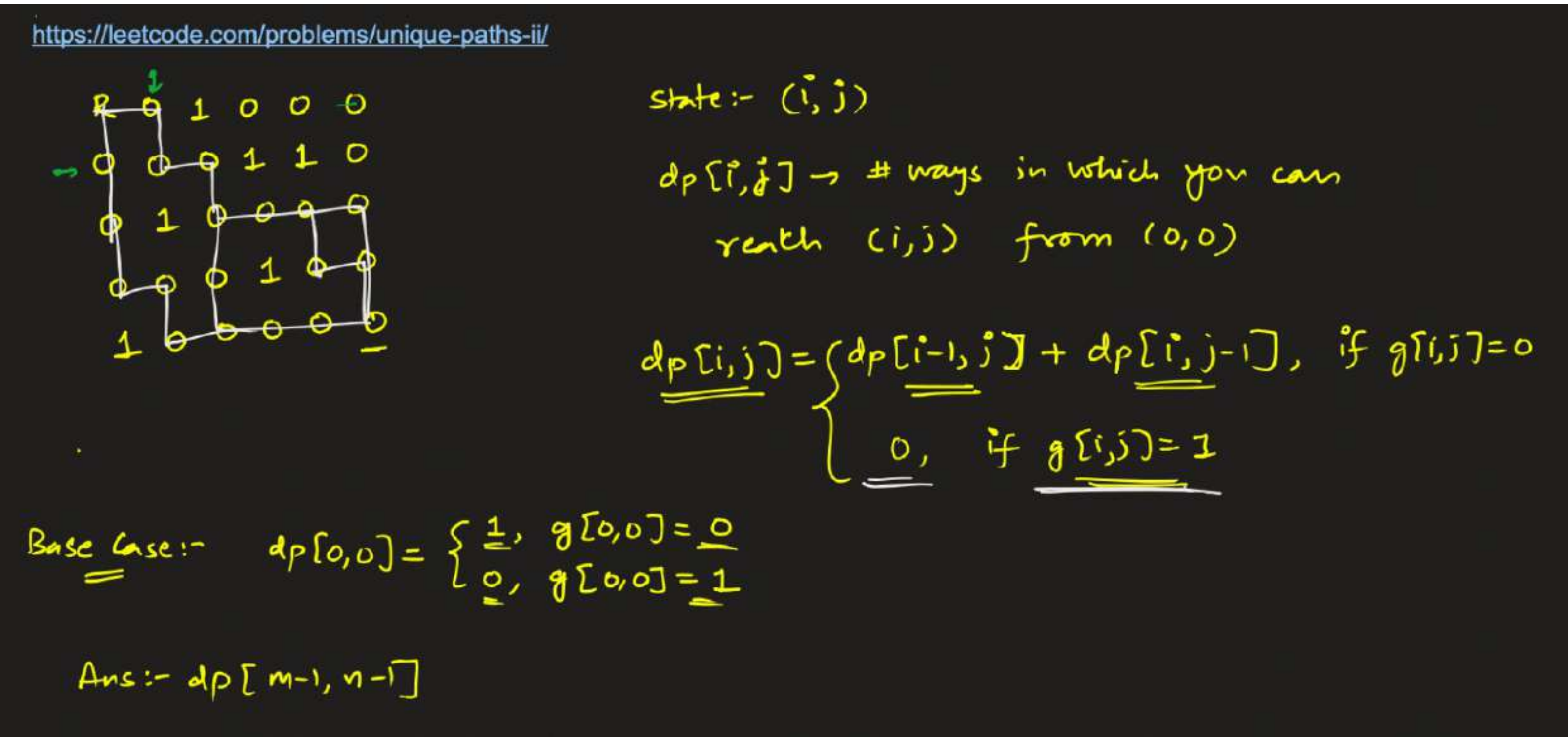
④ Ans:- $dp[n-1]$



```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();
        if(n==1) return nums[0];
        if(n==2) return max(nums[0], nums[1]);
        int l = max(nums[1], nums[0]), sl = nums[0], tmp;
        for(int i=2; i<n; i++) {
            tmp = max(l, sl+nums[i]);
            sl = l;
            l = tmp;
        }
        return l;
    }
};
```

Time = O(n)

Space = O(1)



https://leetcode.com/problems/unique-paths-ii/

R	0	1	0	0	0
0	0	0	1	1	0
0	1	0	0	0	0
0	0	0	1	0	0
1	0	0	0	0	0

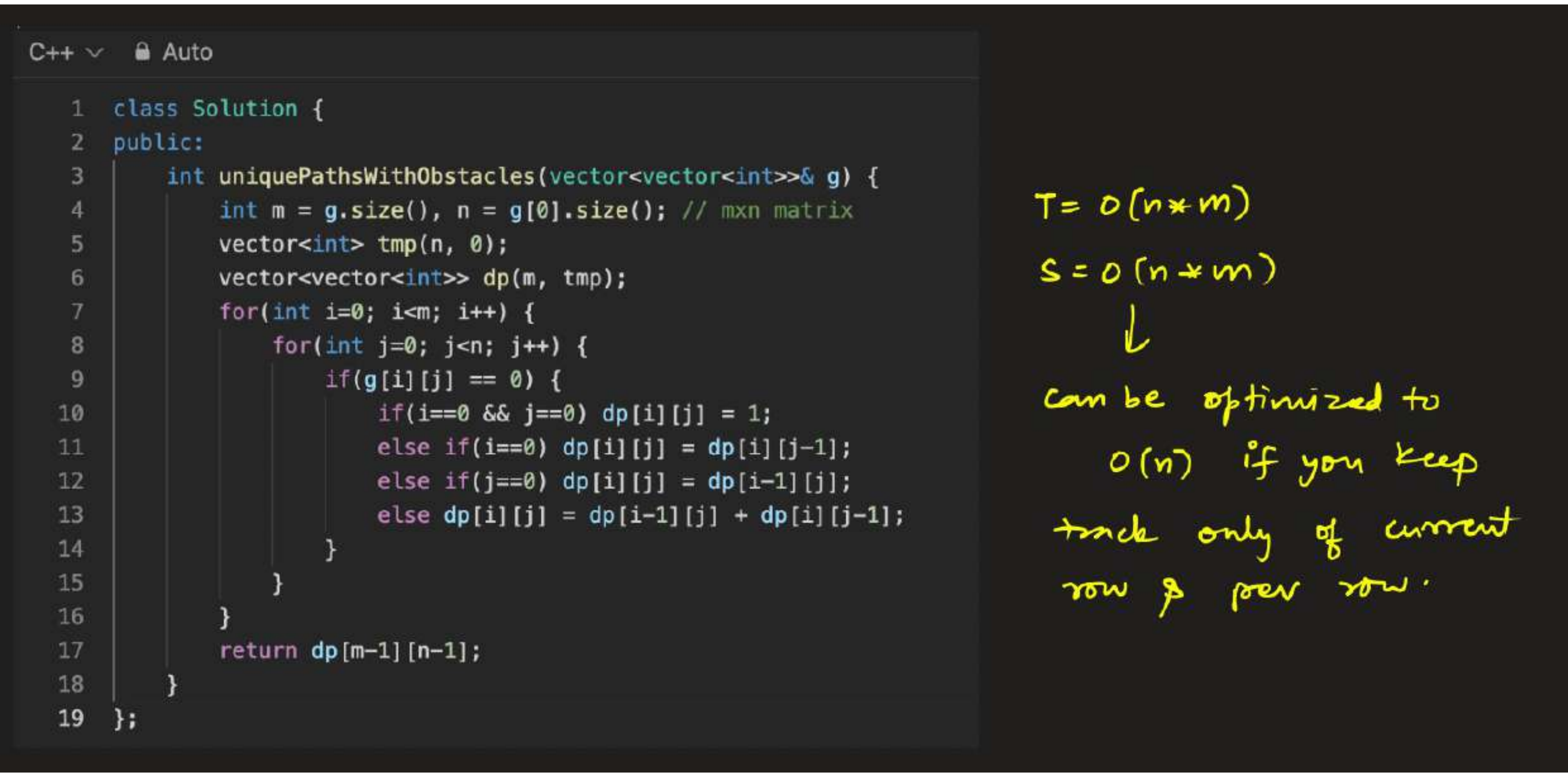
state :- (i, j)

$dp[i,j]$ → # ways in which you can reach (i,j) from $(0,0)$

$$dp[i,j] = \begin{cases} dp[i-1,j] + dp[i,j-1], & \text{if } g[i,j]=0 \\ 0, & \text{if } g[i,j]=1 \end{cases}$$

Base Case :- $dp[0,0] = \begin{cases} 1, & g[0,0]=0 \\ 0, & g[0,0]=1 \end{cases}$

Ans :- $dp[m-1, n-1]$



```cpp
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& g) {
        int m = g.size(), n = g[0].size(); // mxn matrix
        vector<int> tmp(n, 0);
        vector<vector<int>> dp(m, tmp);
        for(int i=0; i<m; i++) {
            for(int j=0; j<n; j++) {
                if(g[i][j] == 0) {
                    if(i==0 && j==0) dp[i][j] = 1;
                    else if(i==0) dp[i][j] = dp[i][j-1];
                    else if(j==0) dp[i][j] = dp[i-1][j];
                    else dp[i][j] = dp[i-1][j] + dp[i][j-1];
                }
            }
        }
        return dp[m-1][n-1];
    }
};
```

$T = O(n*m)$

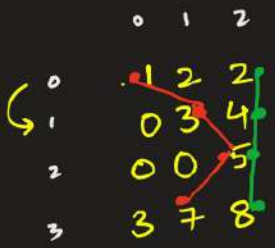
$S = O(n*m)$

↓

can be optimized to $O(n)$ if you keep track only of current row & prev row.

$$\begin{cases} 0 \le (c1+d_1) \le m-1 \\ 0 \le (c2+d_2) \le m-1 \end{cases}$$

https://www.geeksforgeeks.org/problems/chocolates-pickup/1



	0	1	2
0	1	2	2
1	0	3	4
2	0	0	5
3	3	7	8

4×3
n m

● → $2+4+\overset{\checkmark}{5}+8$

● → $1+3+\underset{\checkmark}{0}+7$

Both start from row 0

→ any point in time, row number of both robots will be same.

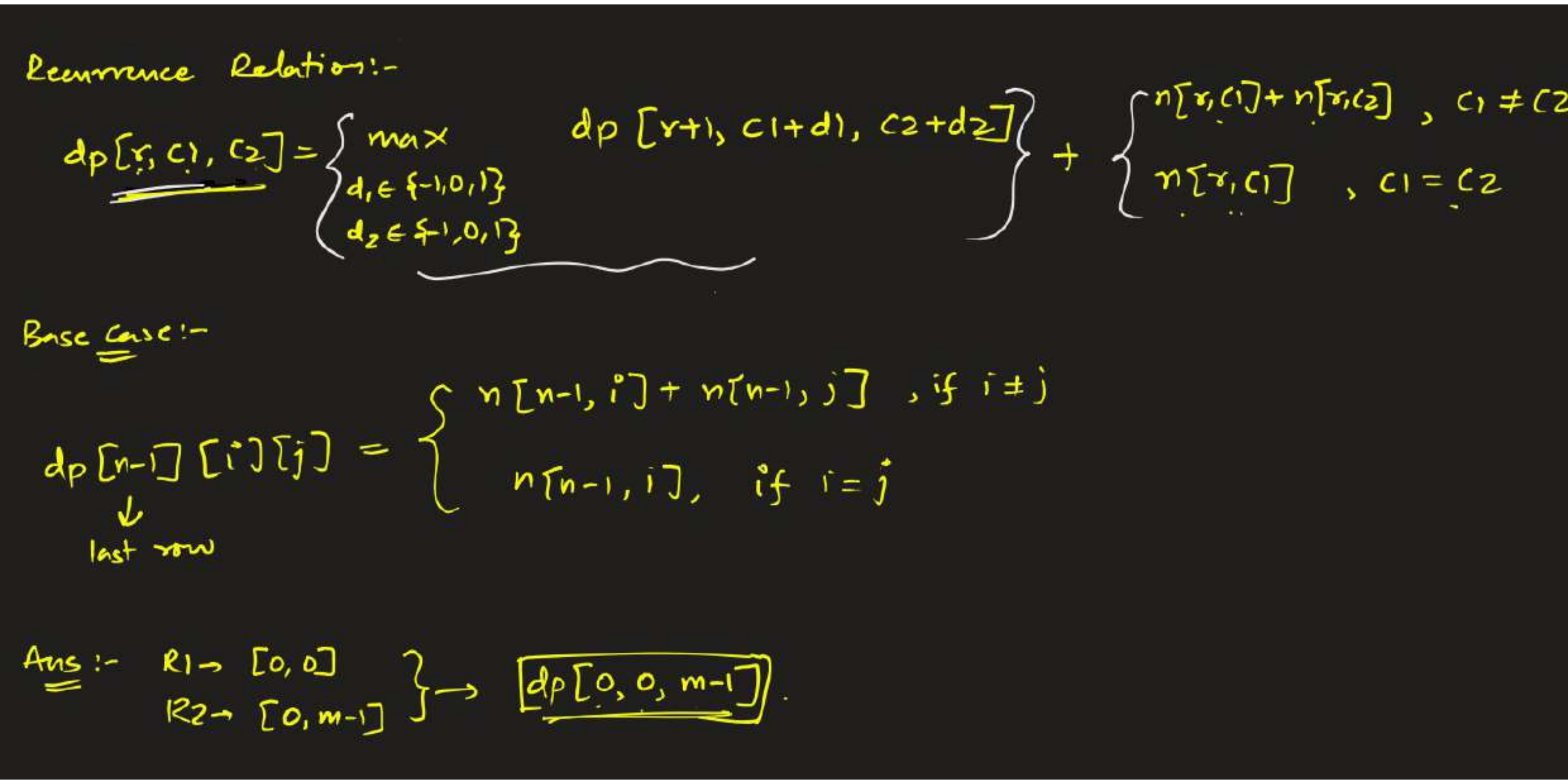
but the column number can be different.

State:- $\underline{(r, c1, c2)}$

(r+1, c1-1, c2-1)   (r+1, c1-1, c2) . . .   (r+1, c1+1, c2+1)

(r, c1, c2) → (r, c1+d1, c2+d2),

$d_1 = -1, 0, 1$
$d_2 = -1, 0, 1$



Recurrence Relation:-

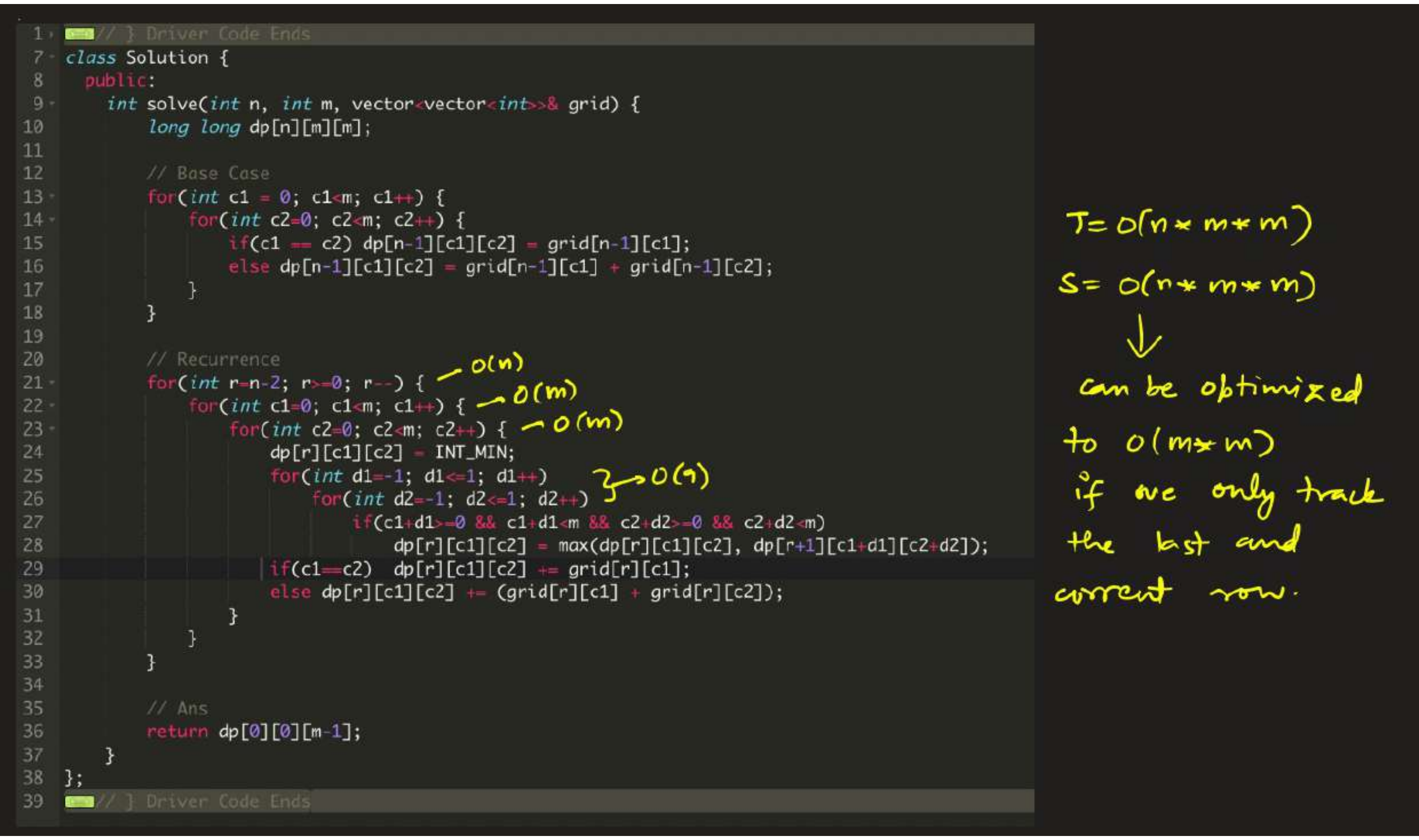
$$\underline{dp[r, c1, c2]} = \underbrace{\max_{\substack{d_1 \in \{-1,0,1\} \\ d_2 \in \{-1,0,1\}}} dp[r+1, c1+d1, c2+d2]} + \begin{cases} n[r, c1] + n[r, c2] & , c1 \neq c2 \\ n[r, c1] & , c1 = c2 \end{cases}$$

Base case:-

$$\underset{\substack{\downarrow \\ \text{last row}}}{dp[n-1]}[i][j] = \begin{cases} n[n-1, i] + n[n-1, j] & , \text{if } i \neq j \\ n[n-1, i], & \text{if } i = j \end{cases}$$

Ans:-

$$\left.\begin{array}{l} R1 \rightarrow [0, 0] \\ R2 \rightarrow [0, m-1] \end{array}\right\} \rightarrow \boxed{dp[0, 0, m-1]}.$$



```
// } Driver Code Ends
class Solution {
  public:
    int solve(int n, int m, vector<vector<int>>& grid) {
        long long dp[n][m][m];

        // Base Case
        for(int c1 = 0; c1<m; c1++) {
            for(int c2=0; c2<m; c2++) {
                if(c1 == c2) dp[n-1][c1][c2] = grid[n-1][c1];
                else dp[n-1][c1][c2] = grid[n-1][c1] + grid[n-1][c2];
            }
        }

        // Recurrence
        for(int r=n-2; r>=0; r--) {
            for(int c1=0; c1<m; c1++) {
                for(int c2=0; c2<m; c2++) {
                    dp[r][c1][c2] = INT_MIN;
                    for(int d1=-1; d1<=1; d1++)
                        for(int d2=-1; d2<=1; d2++)
                            if(c1+d1>=0 && c1+d1<m && c2+d2>=0 && c2+d2<m)
                                dp[r][c1][c2] = max(dp[r][c1][c2], dp[r+1][c1+d1][c2+d2]);
                    if(c1==c2)  dp[r][c1][c2] += grid[r][c1];
                    else dp[r][c1][c2] += (grid[r][c1] + grid[r][c2]);
                }
            }
        }

        // Ans
        return dp[0][0][m-1];
    }
};
// } Driver Code Ends
```

→ O(n)

→ O(m)

→ O(m)

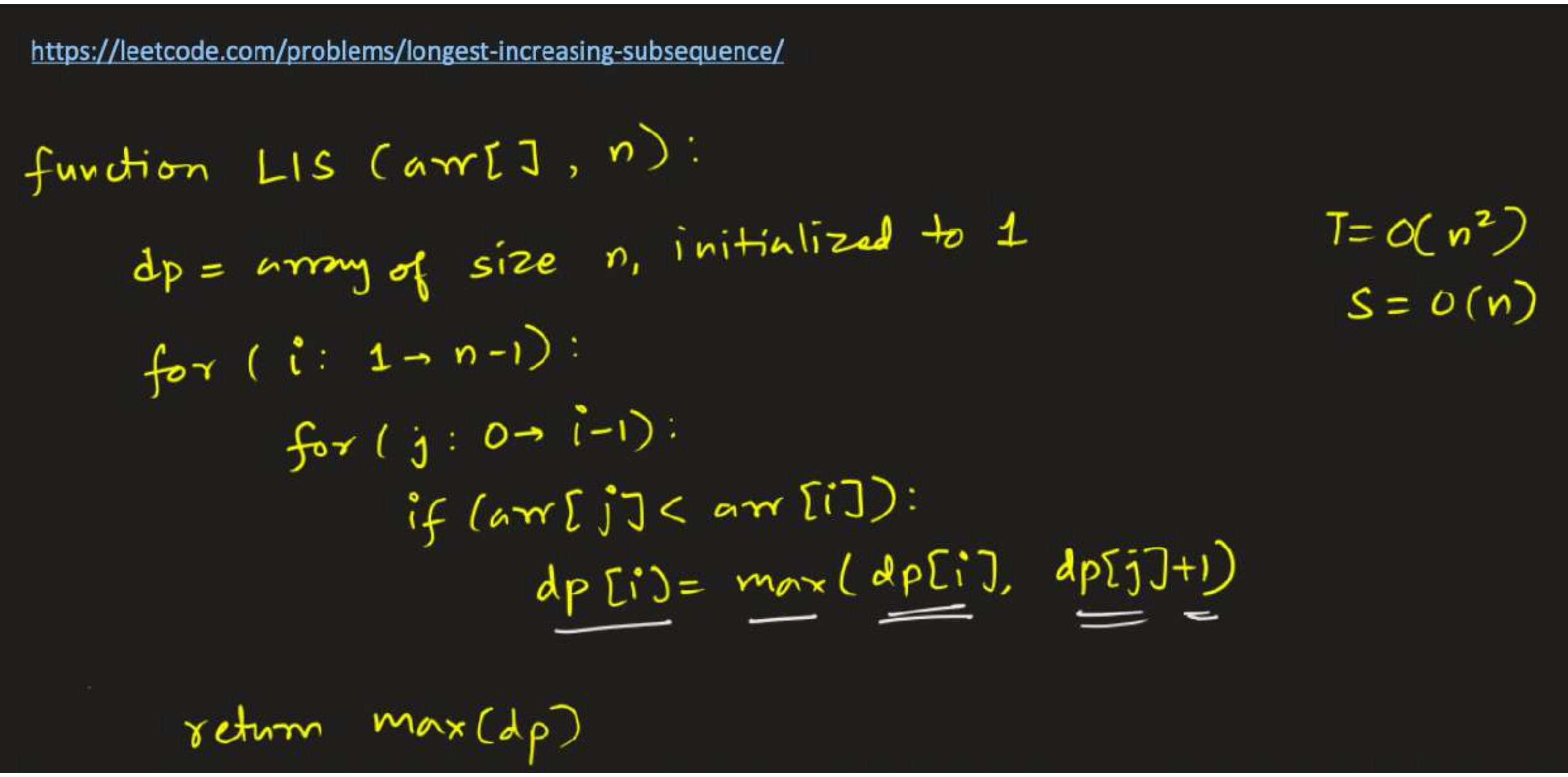
} → O(9)

T= O(n*m*m)

S= O(n*m*m)

↓

can be optimized to O(m*m) if we only track the last and current row.



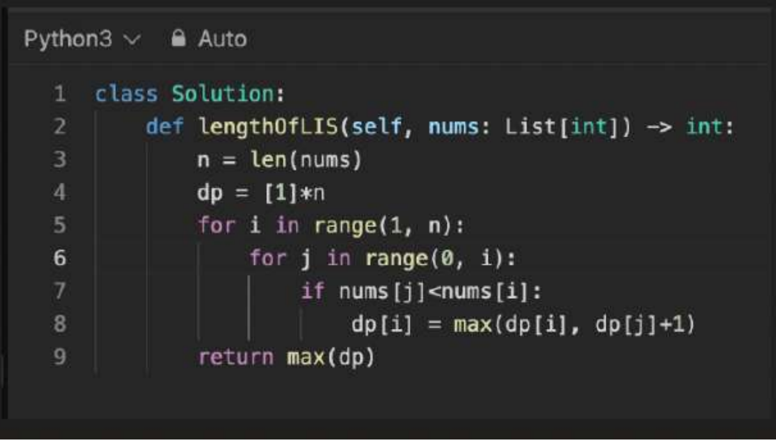
https://leetcode.com/problems/longest-increasing-subsequence/

```
function LIS (arr[ ], n):
    dp = array of size n, initialized to 1
    for ( i: 1 → n-1):
        for ( j : 0 → i-1):
            if (arr[ j ] < arr [i]):
                dp [i] = max ( dp[i], dp[j]+1)

    return max(dp)
```

$T = O(n^2)$

$S = O(n)$

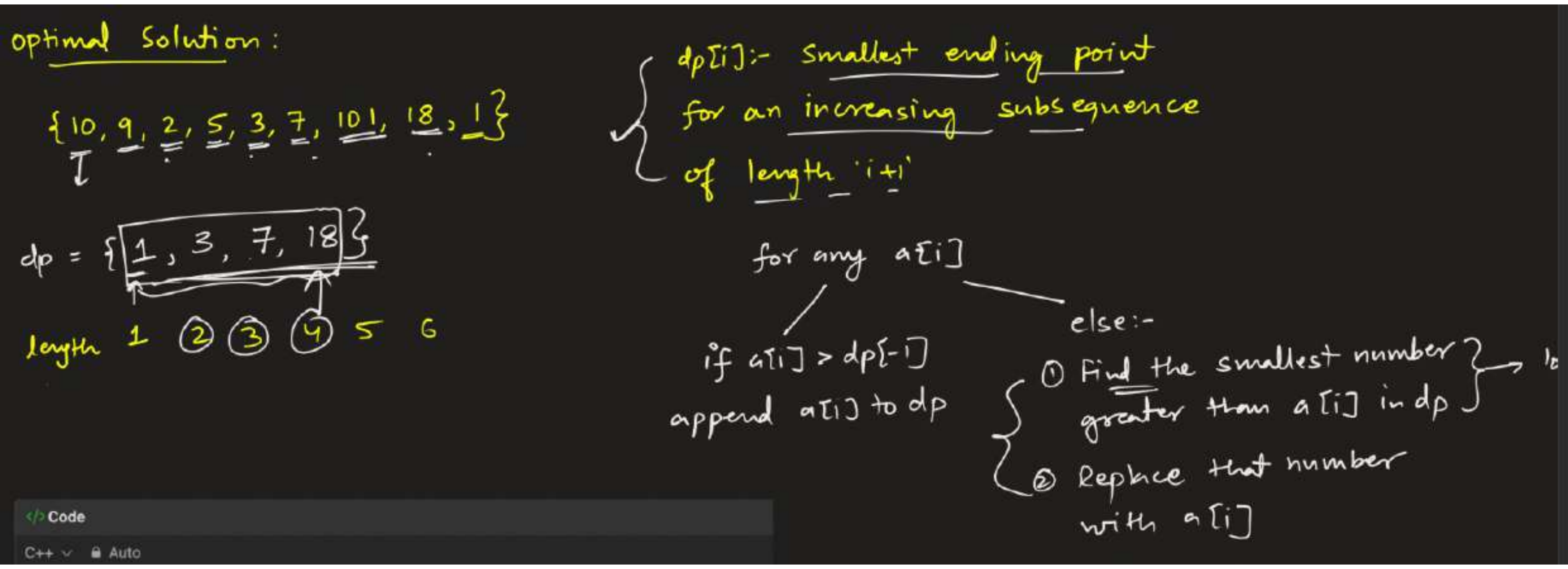


```python
class Solution:
    def lengthOfLIS(self, nums: List[int]) -> int:
        n = len(nums)
        dp = [1]*n
        for i in range(1, n):
            for j in range(0, i):
                if nums[j]<nums[i]:
                    dp[i] = max(dp[i], dp[j]+1)
        return max(dp)
```

$T = O(n^2)$

$S = O(n)$

dp[i]
↓
longest subsequence
length ending at
index 'i'



optimal Solution :

{10, 9, 2, 5, 3, 7, 101, 18, 1}

dp = {1, 3, 7, 18}

length 1 ② ③ ④ 5 6

dp[i] :- smallest ending point for an increasing subsequence of length 'i+1'

for any a[i]

if a[i] > dp[-1]
append a[i] to dp

else :-
① Find the smallest number greater than a[i] in dp → lo
② Replace that number with a[i]

```cpp
class Solution {
public:
    int lengthOfLIS(vector<int>& nums) {
        int n = nums.size();
        vector<int> dp = {nums[0]};
        for(int i=1; i<n; i++) {
            if(nums[i] > dp.back())
                dp.push_back(nums[i]);
            else {
                int low = lower_bound(dp.begin(), dp.end(), nums[i]) - dp.begin();
                dp[low] = nums[i];
            }
        }
        return dp.size();
    }
};
```

$T = O(n \log n)$

$S = O(n)$ ✓