**Name:** Aditya Singh

**UID:** 22BCS15063

**Section:** FL\_IOT\_601 - A

**Assignment – 7 Solutions:-**

1. **Climbing Stairs:-**

class Solution {

public int climbStairs(int n) {

if(n == 0) return 1;

if(n == 1) return 1;

int prev1 = 1, prev2 = 1;

for(int i = 2; i <= n; i++){

int curr = prev1 + prev2;

prev1 = prev2;

prev2 = curr;

}

return prev2;

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Best Time to Buy and Sell a Stock:-**

class Solution {

public:

int maxProfit(vector<int>& prices){

int buy = 0, sell = 1;

int profit = 0;

for(int i = 1; i < prices.size(); i++){

profit = profit > prices[sell] - prices[buy] ? profit : prices[sell] - prices[buy];

if(prices[sell] < prices[buy]){

buy = sell;

}

sell++;

}

return profit;

}

};

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Maximum Subarray:-**

class Solution {

public:

int maxSubArray(vector<int>& nums) {

int ans = nums[0], f = nums[0];

for (int i = 1; i < nums.size(); ++i) {

f = max(f, 0) + nums[i];

ans = max(ans, f);

}

return ans;

}

};

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **House Robber:-**

class Solution {

public int rob(int[] nums) {

int first = 0;

int second = 0;

int max = 0;

for(int num : nums){

max = Math.max(first + num, second);

first = second;

second = max;

}

return max;

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Jump Game:-**

class Solution {

public boolean canJump(int[] nums) {

boolean arr[] = new boolean[nums.length];

int final1 = nums.length - 1;

arr[nums.length-1] = true;

for(int i = nums.length-2; i >= 0; i--){

if(nums[i]+i >= final1){

arr[i] = true;

final1 = i;

}

else{

arr[i] = false;

}

}

return arr[0];

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Unique Paths:-**

class Solution {

public int uniquePaths(int m, int n) {

int [][] dp = new int[m][n];

for(int[] row:dp){

Arrays.fill(row,1);

}

for(int i=1; i<m;i++){

for(int j=1; j<n; j++){

dp[i][j] = dp[i-1][j]+dp[i][j-1];

}

}

return dp[m-1][n-1];

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Coin Change:-**

class Solution {

public int coinChange(int[] coins, int amount) {

int[] amt = new int[amount+1];

Arrays.fill(amt, amount+1);

amt[0] = 0;

for(int i = 1; i <= amount; i++){

for(int j = 0; j < coins.length; j++){

if(i >= coins[j]){

amt[i] = Math.min(amt[i], 1 + amt[i-coins[j]]);

}

}

}

if(amt[amount] < amount + 1) return amt[amount];

else return -1;

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Longest Increasing Subsequence:-**

class Solution {

public int lengthOfLIS(int[] nums) {

int[] lis = new int[nums.length];

Arrays.fill(lis,1);

int max = 1;

for(int i=1; i<nums.length; i++){

for(int j=0; j<i; j++){

if(nums[i]>nums[j]){

lis[i] = Math.max(lis[i], 1+lis[j]);

max = Math.max(max, lis[i]);

}

}

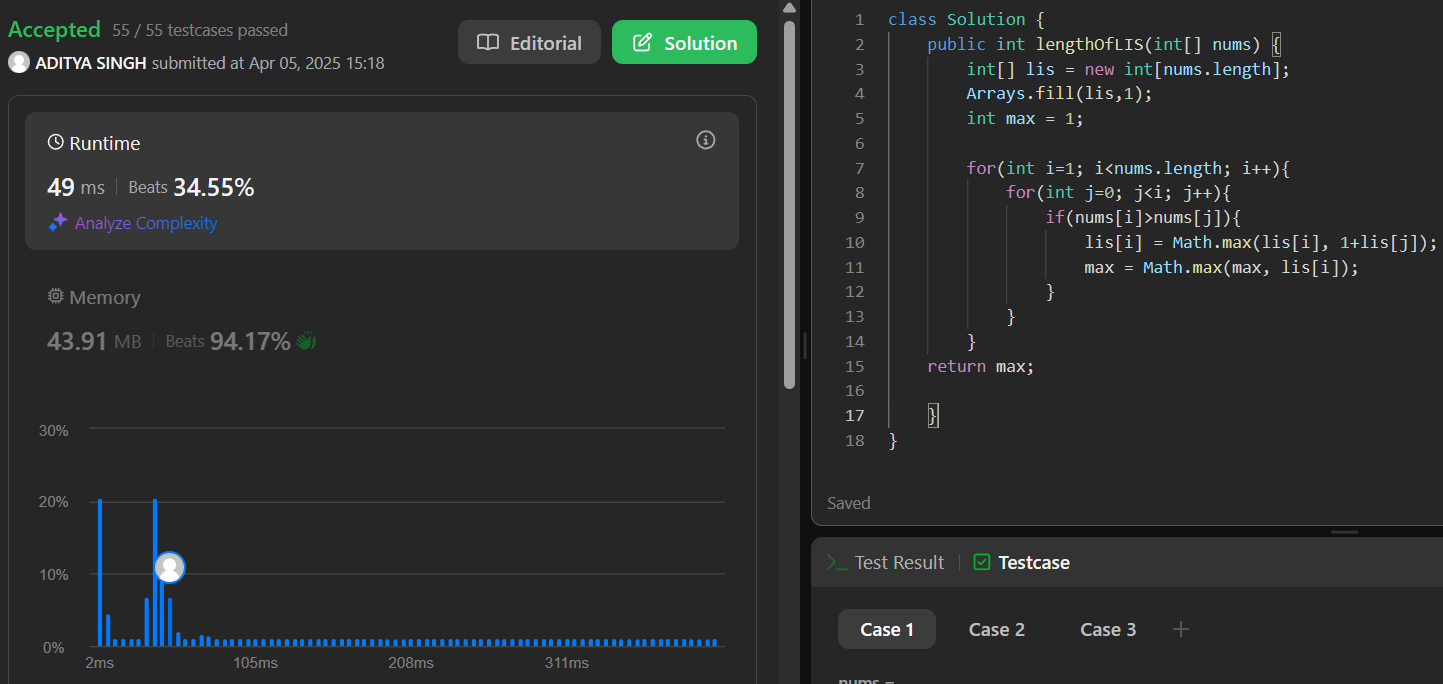
}

return max;

}

}

Result:-

****

1. **Maximum Product Subarray:-**

class Solution {

public int maxProduct(int[] nums) {

if(nums.length == 0) return 0;

int min = nums[0], max = nums[0], result = max;

for(int i=1; i<nums.length; i++){

int cur = nums[i];

int temp = Math.max(cur, Math.max(max\*cur, min\*cur));

min = Math.min(cur, Math.min(max\*cur, min\*cur));

max = temp;

result = Math.max(result, max);

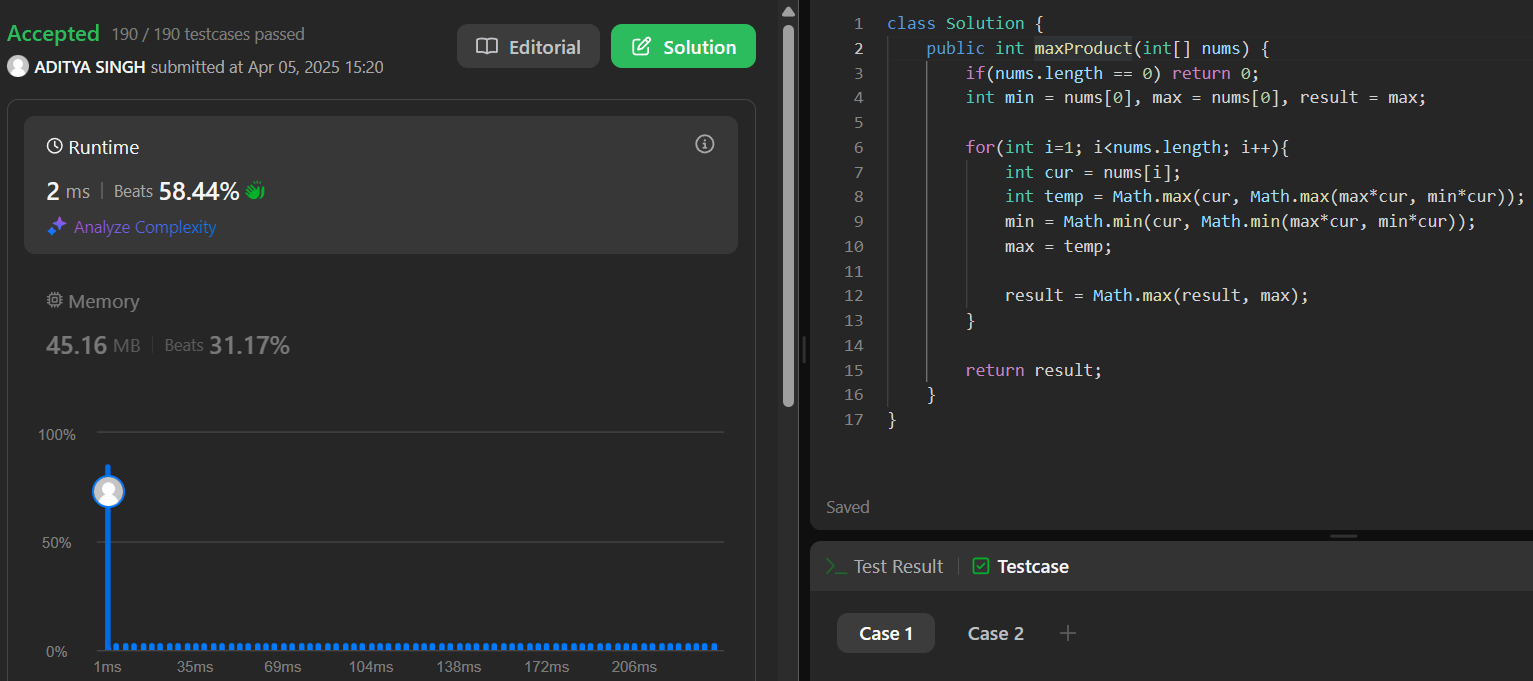
}

return result;

}

}

Result:-



1. **Decode Ways:-**

class Solution {

public int numDecodings(String s) {

if(s.charAt(0) == '0') return 0;

int one = 1, two = 1;

for(int i = 1; i < s.length(); i++){

int curr = 0;

if(s.charAt(i) != '0'){

curr = one;

}

int val = Integer.parseInt(s.substring(i-1, i+1));

if(val >= 10 && val <=26){

curr = curr + two;

}

two = one;

one = curr;

}

return one;

}

}

Result:-

A screenshot of a computer

AI-generated content may be incorrect.

1. **Best Time to Buy and Sell a Stock with Cooldown:-**

class Solution {

public int maxProfit(int[] prices) {

if (prices == null || prices.length <= 1) return 0;

int n = prices.length;

int[] buy = new int[n];

int[] sell = new int[n];

int[] rest = new int[n];

buy[0] = -prices[0];

sell[0] = 0;

rest[0] = 0;

for (int i = 1; i < n; i++) {

buy[i] = Math.max(buy[i-1], rest[i-1] - prices[i]);

sell[i] = Math.max(sell[i-1], buy[i-1] + prices[i]);

rest[i] = Math.max(rest[i-1], sell[i-1]);

}

return Math.max(sell[n-1], rest[n-1]);

}

}

Result:-

