**Problem 1**

**Aim:**

Longest Increasing Subsequence

**Code:**

public class Solution {

    public int lengthOfLIS(int[] nums) {

        if (nums == null || nums.length == 0) {

            return 0;

        }

        int n = nums.length;

        int[] dp = new int[n];

        Arrays.fill(dp, 1);

        for (int i = 1; i < n; ++i) {

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

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

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

                }

            }

        }

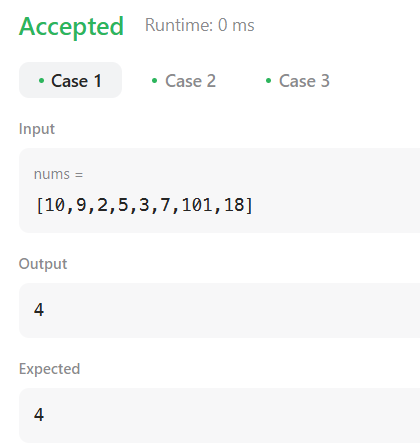
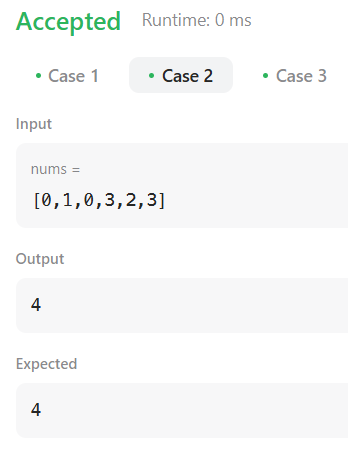
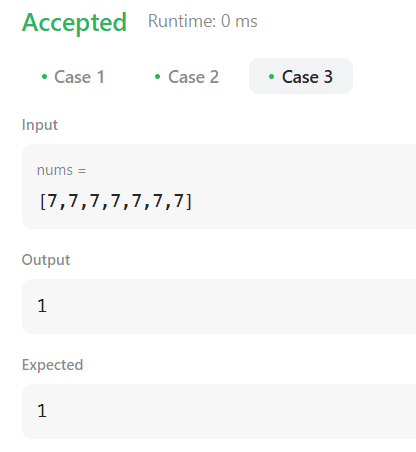
        int maxLength = Arrays.stream(dp).max().orElse(0);

        return maxLength;

    }

}

**Output:**

Case 1 Case 2 Case 3

**Problem 2**

**Aim:**

Maximum Product Subarray

**Code:**

class Solution {

    public int maxProduct(int[] nums) {

        int res = Integer.MIN\_VALUE;

        for (int n : nums) {

            res = Math.max(res, n);

        }

        int curMax = 1, curMin = 1;

        for (int n : nums) {

            int temp = curMax \* n;

            curMax = Math.max(temp, Math.max(curMin \* n, n));

            curMin = Math.min(temp, Math.min(curMin \* n, n));

            res = Math.max(res, curMax);

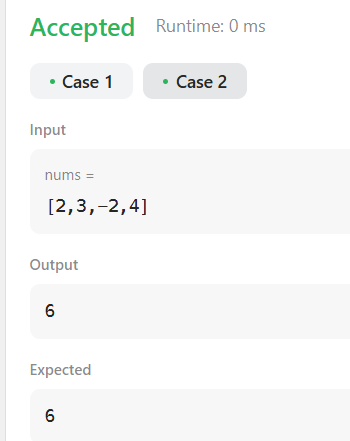
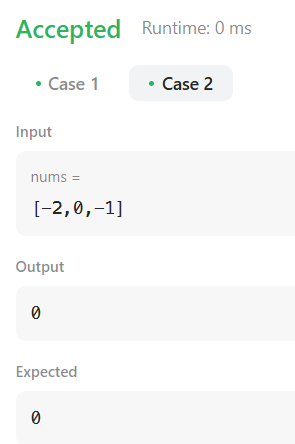
        }

        return res;

    }

}

**Output:**

Test Case 1 Test Case 2

**Problem 3**

**Aim:**

Decode Ways

**Code:**

class Solution {

    public int numDecodings(String s) {

        if (s.charAt(0) == '0') {

            return 0;

        }

        int n = s.length();

        int[] dp = new int[n + 1];

        dp[0] = dp[1] = 1;

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

            int one = Character.getNumericValue(s.charAt(i - 1));

            int two = Integer.parseInt(s.substring(i - 2, i));

            if (1 <= one && one <= 9) {

                dp[i] += dp[i - 1];

            }

            if (10 <= two && two <= 26) {

                dp[i] += dp[i - 2];

            }

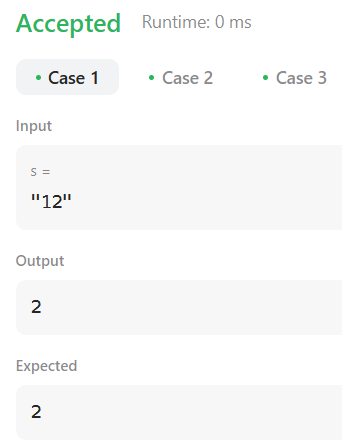
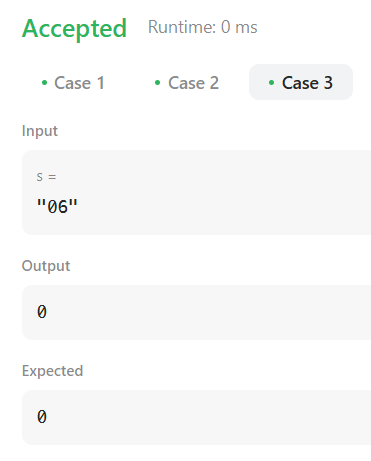
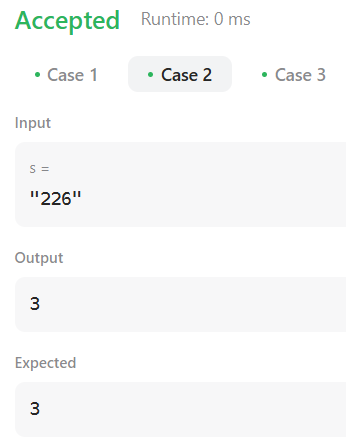
        }

        return dp[n];

    }

}

**Output:**

Case 1 Case 2 Case 3

**Problem 4**

**Aim:**

Best time to buy and Sell a Stock with Cooldown

**Code:**

class Solution {

    public int maxProfit(int[] prices) {

     int[][] dp = new int[prices.length+3][prices.length+3];

     for(int i = prices.length-1 ; i>=0 ; i--){

        for( int j = prices.length -1 ; j >=0 ; j--){

            int skip1 = dp[i+1][j];

            int skip2  = dp[j+2][j+1];

            int solve = 0;

            if(j<i&&prices[j]<prices[i]){

                 solve = prices[i]-prices[j]+dp[i+3][i+2];

            }

          dp[i][j]=Math.max(skip2,Math.max(skip1,solve));

        }

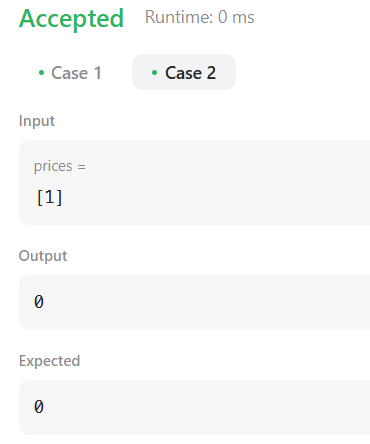
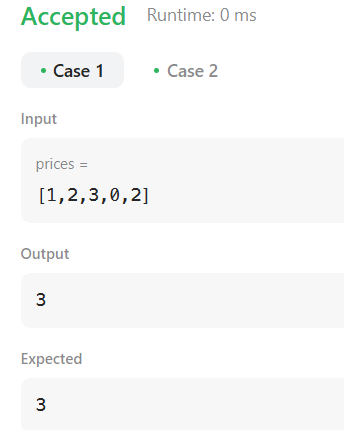
     }

     return dp[0][0];

    }

}

**Output:**



Case 1 Case 2

**Problem 5**

**Aim:**

Perfect Squares

**Code:**

class Solution {

    public int numSquares(int n) {

        int[] dp = new int[n + 1];

        Arrays.fill(dp, Integer.MAX\_VALUE);

        dp[0] = 0;

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

            int min\_val = Integer.MAX\_VALUE;

            for (int j = 1; j \* j <= i; ++j) {

                min\_val = Math.min(min\_val, dp[i - j \* j] + 1);

            }

            dp[i] = min\_val;

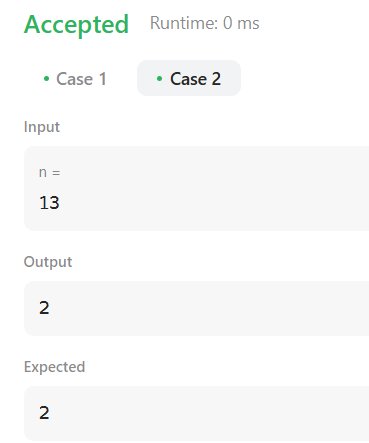
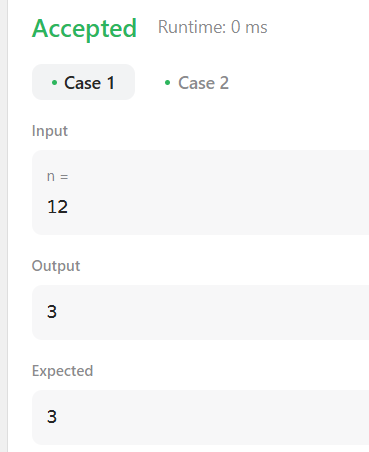
        }

        return dp[n];

    }

}

**Output:**



Case 1 Case 2

**Problem 6**

**Aim:**

Word Break

**Code:**

class Solution {

    public boolean wordBreak(String s, List<String> wordDict) {

        return recWay1(s, wordDict);

    }

    boolean recWay2(String s, List<String> wordDict) {

        Boolean[] memo = new Boolean[s.length() + 1];

        return wordBreak2(s, new HashSet<>(wordDict), 0, memo);

    }

    boolean wordBreak2(String s, Set<String> wordDict, int k, Boolean[] memo) {

        int n = s.length();

        if (k == n) return true;

        if (memo[k] != null) return memo[k];

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

            String word = s.substring(k, i);

            if (wordDict.contains(word) && wordBreak2(s, wordDict, i, memo)) {

                return memo[k] = true;

            }

        }

        return memo[k] = false;

    }

    boolean recWay1(String s, List<String> wordDict) {

        Boolean[] memo = new Boolean[s.length() + 1];

        return wordBreak(s, wordDict, 0, memo);

    }

    boolean wordBreak(String s, List<String> wordDict, int k, Boolean[] memo) {

        if (k == s.length()) {

            return true;

        }

        if (memo[k] != null) {

            return memo[k];

        }

        for (int i=0; i<wordDict.size(); i++) {

            String word = wordDict.get(i);

            if (s.startsWith(word, k)) {

                if(wordBreak(s, wordDict, k + word.length(), memo)) return memo[k] = true;

            }

        }

        return memo[k] = false;

    }

}

**Output:**

Case 1 Case 2 Case 3

**Problem 7**

**Aim:**

Word Break 2

**Code:**

import java.util.\*;

class Solution {

    public List<String> wordBreak(String s, List<String> wordDict) {

        Set<String> wordSet = new HashSet<>(wordDict);

        Map<Integer, List<String>> memo = new HashMap<>();

        return backtrack(s, 0, wordSet, memo);

    }

    private List<String> backtrack(String s, int start, Set<String> wordSet, Map<Integer, List<String>> memo) {

        if (memo.containsKey(start)) {

            return memo.get(start);

        }

        List<String> result = new ArrayList<>();

        if (start == s.length()) {

            result.add("");

            return result;

        }

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

            String word = s.substring(start, end);

            if (wordSet.contains(word)) {

                List<String> sublist = backtrack(s, end, wordSet, memo);

                                for (String sub : sublist) {

                    if (sub.isEmpty()) {

                        result.add(word);

                    } else {

                        result.add(word + " " + sub);

                    }

                }

            }

        }

                memo.put(start, result);

        return result;

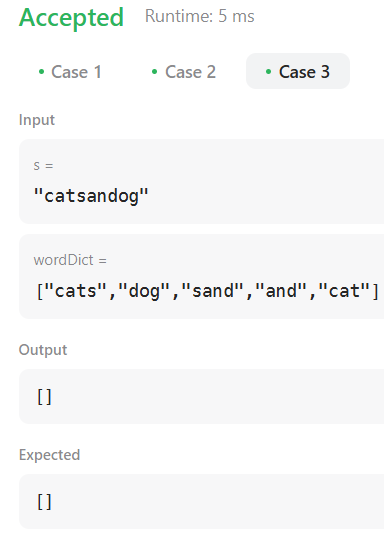
    }

}

**Output:**



Case 1 Case 2



Case 3