EXPERIMENT-5

Student Name: Hritik Kumar **UID:**22BET10113

Branch: BE -IT **Section/Group:**22BET_IOT-703(A)

Semester: 6th **Subject Code:** 22ITP-351

PROBLEM-1

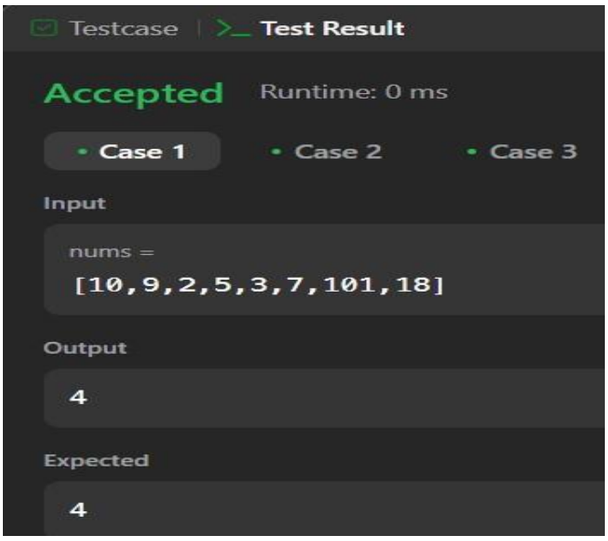
AIM:-

Longest Increasing Subsequence

CODE:-

```
class Solution {
    public int lengthOfLIS(int[] nums) {
        List<Integer> res = new ArrayList<>();

        for (int n : nums) {            if (res.isEmpty() ||
res.get(res.size() - 1) < n) {                res.add(n);
} else {
                int idx = binarySearch(res, n);
res.set(idx, n);
            }
        }

        return res.size();
    }

    private int binarySearch(List<Integer> arr, int target) {
int left = 0;        int right = arr.size() - 1;

        while (left <= right) {
int mid = (left + right) / 2;
if (arr.get(mid) == target) {
return mid;
            } else if (arr.get(mid) > target) {
right = mid - 1;
            } else {
left = mid + 1;
            }
        }

        return left;
    }
}
```

OUTPUT:-



PROBLEM-2

AIM:-

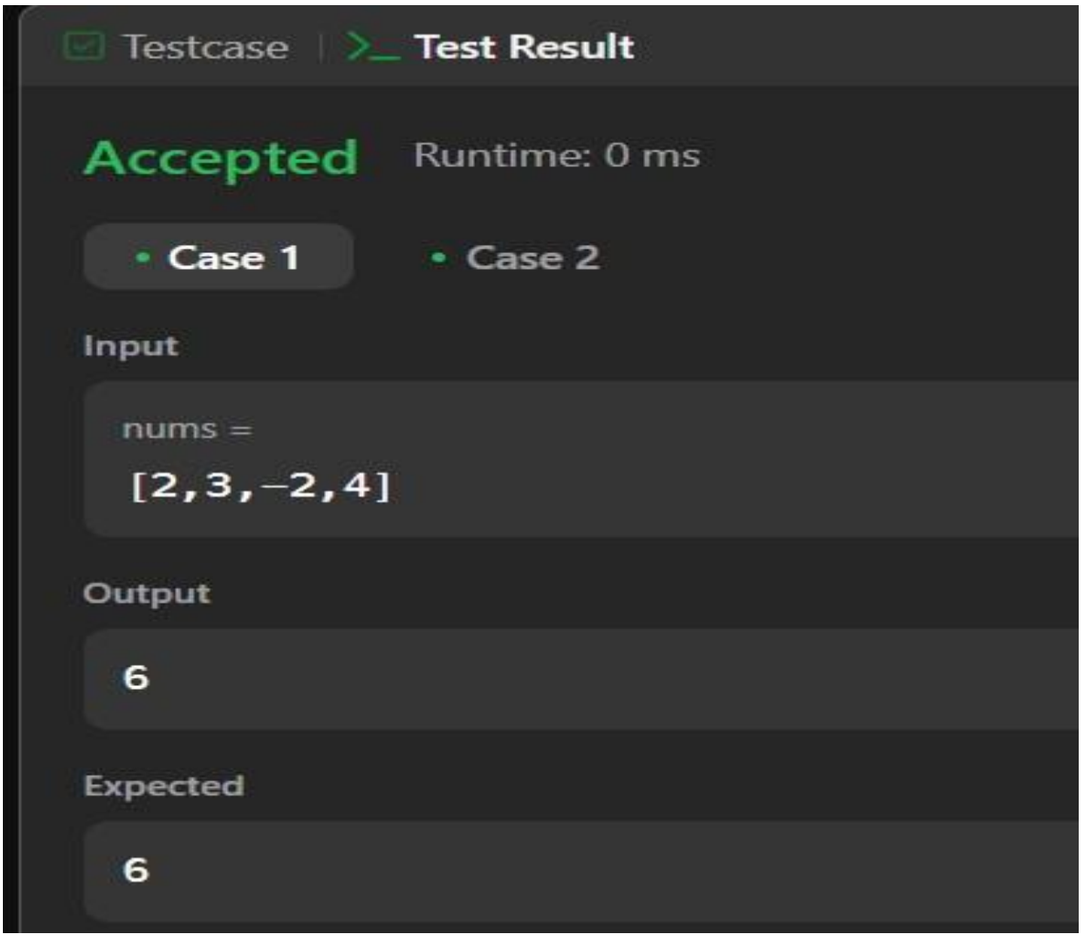
Maximum Product Subarray

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:-



PROBLEM-3

AIM:-

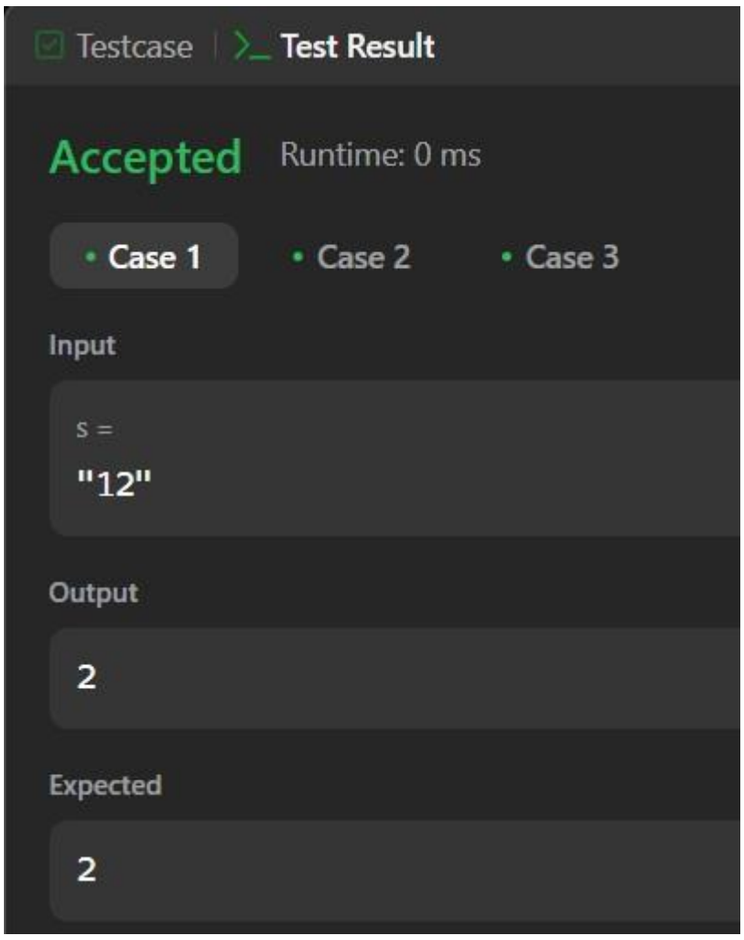
Decode Ways

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:-



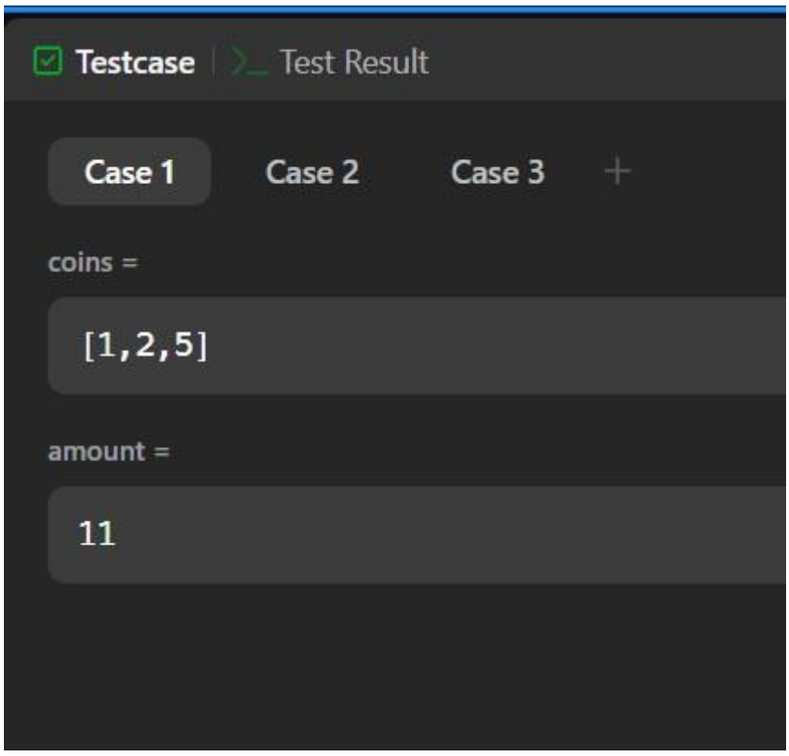
PROBLEM-4

AIM:- Coin Change

CODE:-

```
class Solution {     public int coinChange(int[]
coins, int amount) {         int[] minCoins = new
int[amount + 1];         Arrays.fill(minCoins,
amount + 1);         minCoins[0] = 0;


        for (int i = 1; i <= amount; i++) {             for (int j = 0; j < coins.length;
j++) {                 if (i - coins[j] >= 0) {                     minCoins[i] =
Math.min(minCoins[i], 1 + minCoins[i - coins[j]]);
                }
            }
        }

        return minCoins[amount] != amount + 1 ? minCoins[amount] : -1;
    }
}
```

OUTPUT:-



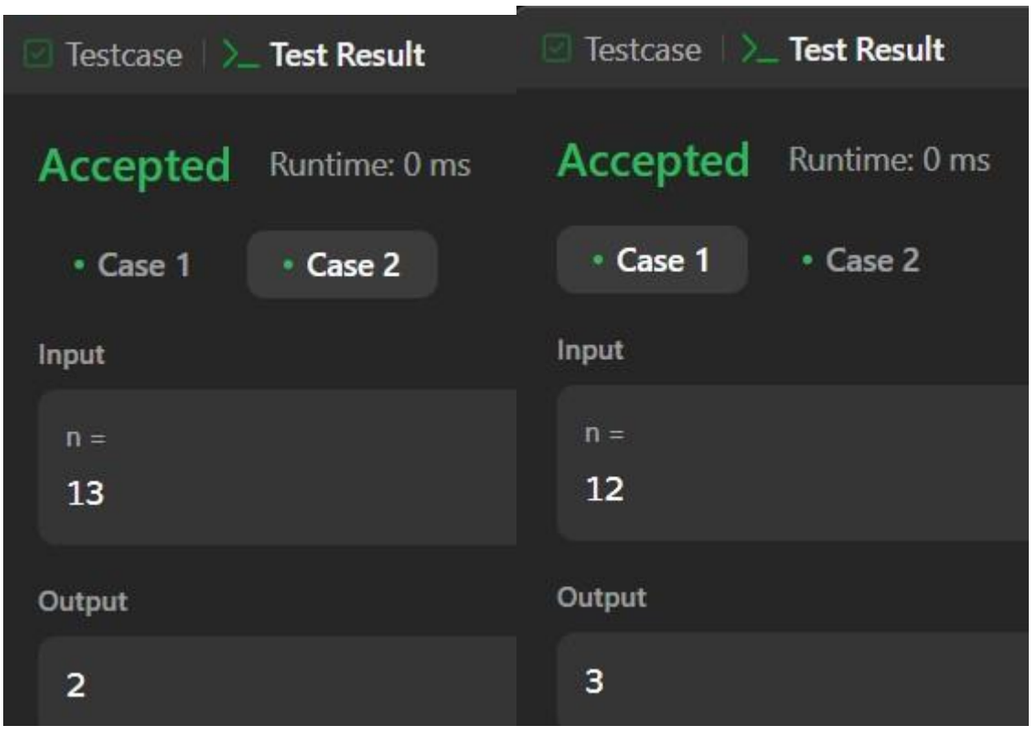
PROBLEM-5

AIM:-

Perfect Squares

CODE:-

```
public class Solution {
 public boolean searchMatrix(int[][] matrix, int target) {
if(matrix == null || matrix.length < 1 || matrix[0].length <1) {
return false;
    }
    int col = matrix[0].length-1;      int row = 0;
while(col >= 0 && row <= matrix.length-1) {
if(target == matrix[row][col]) {            return
true;
        } else if(target < matrix[row][col]) {
          col--;
        } else if(target > matrix[row][col]) {
row++;
        }    }
return false;
  }
}
```



PROBLEM-6

AIM:- Word Break

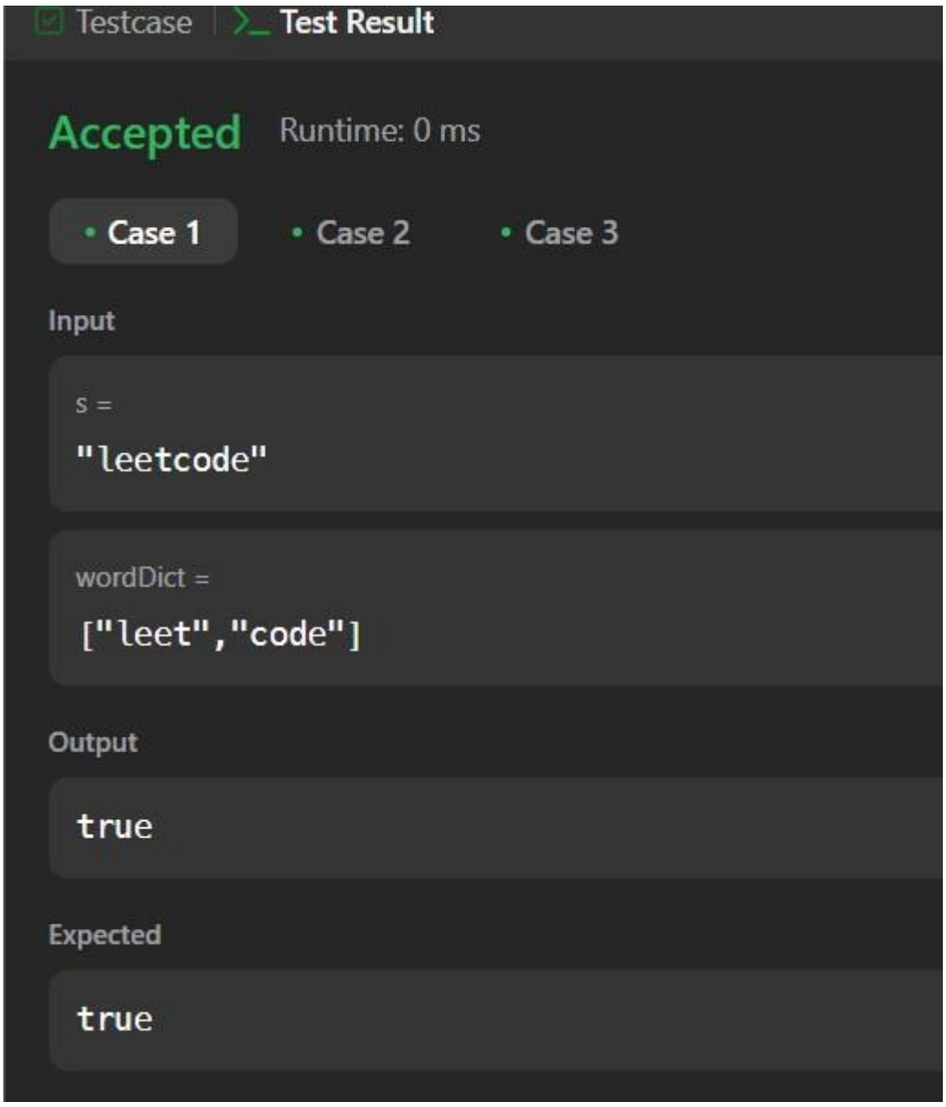
CODE:-

OUTPUT:-

```
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;
    }
}
```



PROBLEM-7

AIM:-

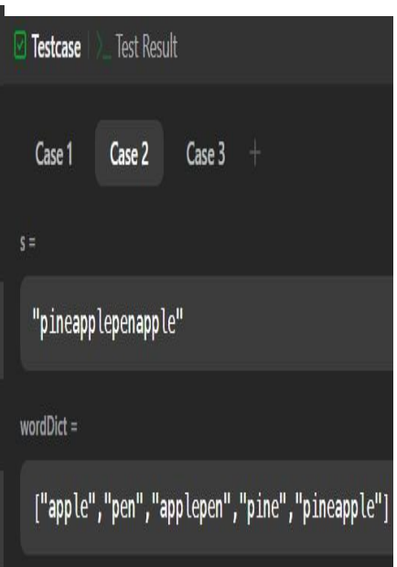
Word Break 2

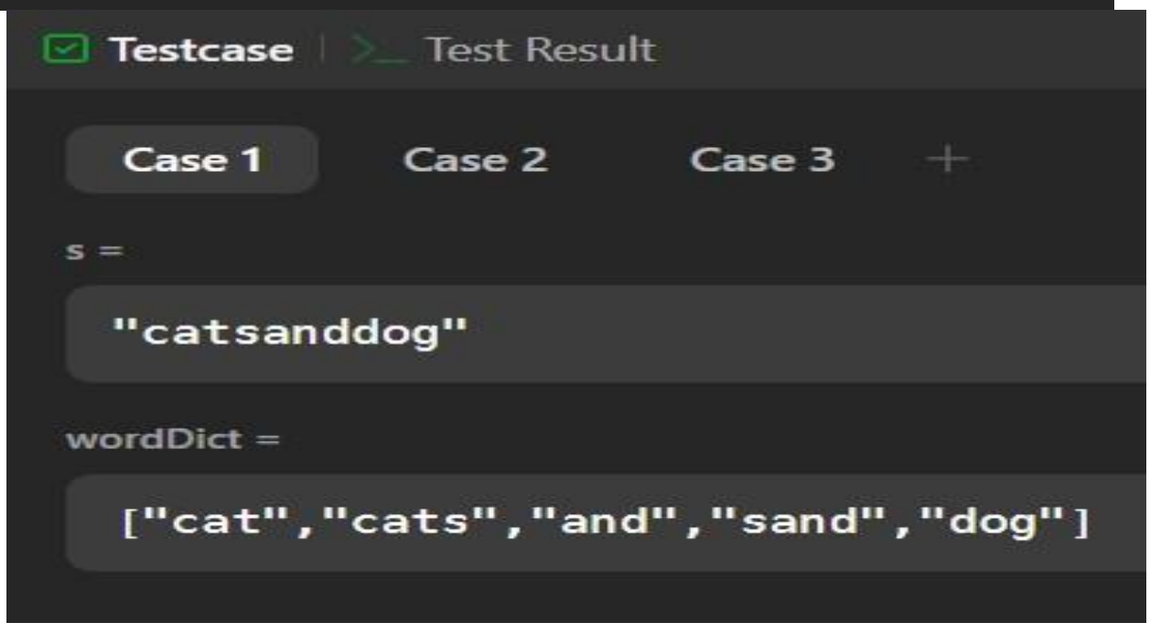
CODE:- import

```
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;
    }
}
```


Testcase
Test Result
Case 1
Case 2
Case 3
s =
"catsandog"
wordDict =
["cats","dog","sand","and","cat"]


Testcase
Test Result
Case 1
Case 2
Case 3
s =
"pineapplepenapple"
wordDict =
["apple","pen","applepen","pine","pineapple"]


Testcase
Test Result
Case 1
Case 2
Case 3
s =
"catsanddog"
wordDict =
["cat","cats","and","sand","dog"]