Experiment-1.1

Student Name: Aafreen Khan **UID:** 21BCS1397
Branch: BE-CSE **Section/Group:**CC-606 A
Semester: 6th **Date of Performance:**12-01-2024
Subject Name: Advanced Programming lab-2 **Subject Code:**21CSP-351

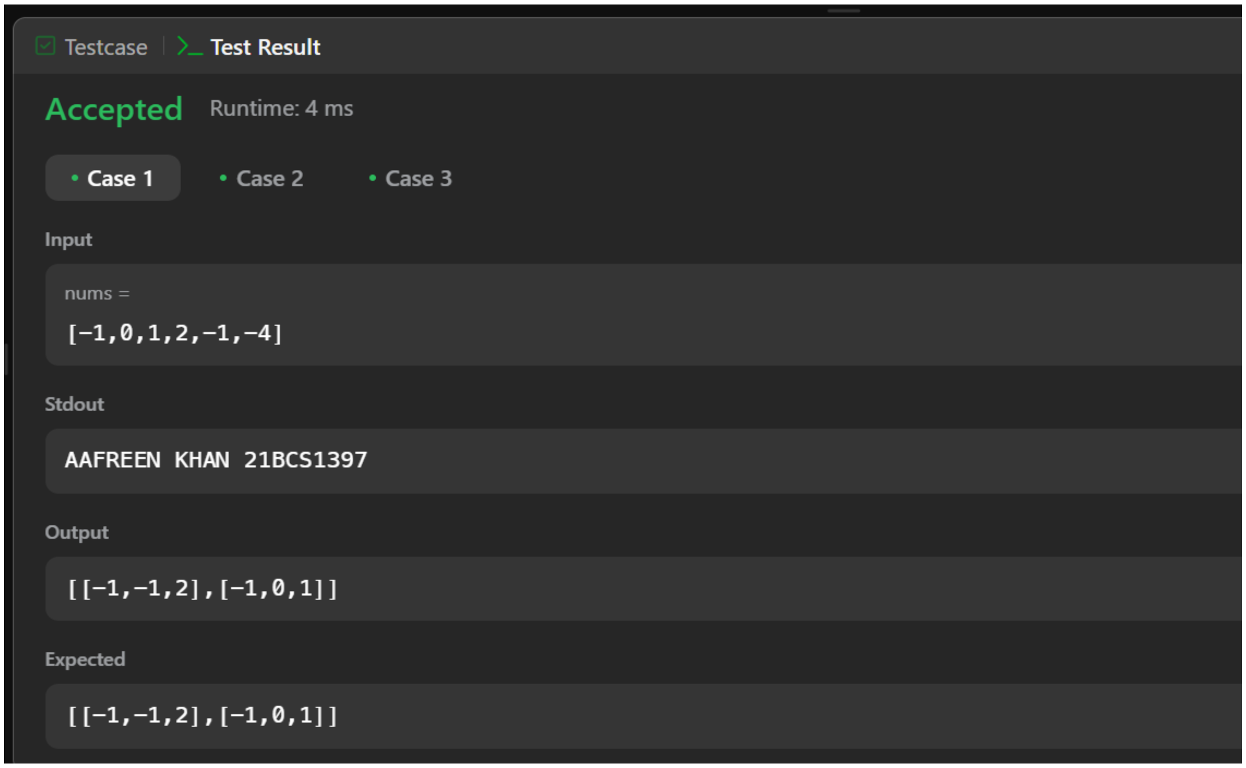
Aim:
- To Solve the 3 SUM Problem
- To Solve the Jump Game 2

Objective:
- Given an integer array nums, return all the triplets [nums[i], nums[j], nums[k]] such that i! = j,i! = k, and j! = k, and nums[i] + nums[j] + nums[k] == 0.
- You are given a 0-indexed array of integers nums of length n. You are initially positioned at nums[0].

Code(A):

```
class Solution
{public:
    vector<vector<int>> threeSum(vector<int>& nums)
        {sort(nums.begin(),nums.end());

        cout<<"AAFREEN KHAN"<<"  "<<"21BCS1397";

        vector<vector<int>>res;
        for(int i=0;i<nums.size()-2;i++)
        {
            if(i==0 || (i>0 && nums[i] != nums[i-1]))
            {
                int low=i+1;
                int hi=nums.size()-
                1;int sum=0-nums[i];
                while(low<hi)
                {
                    if(nums[low]+nums[hi]==sum)
                    {
                        vector<int>temp;
                        temp.push_back(nums[i]);
                        temp.push_back(nums[low]);
                        temp.push_back(nums[hi]);
                        res.push_back(temp);
                        while(low<hi && nums[low]==nums[low+1])
                        low++;while(low<hi && nums[hi]==nums[hi-1])
                        hi--;

                        low++; hi--;
                    }
                    else if(nums[low]+nums[hi]<sum)
                    low++;else hi--;
                }
            }
        }
        return res;
    }
};
```

Output(A):


Testcase | Test Result

Accepted Runtime: 4 ms

Case 1 Case 2 Case 3

Input

nums =
```
[-1,0,1,2,-1,-4]
```

Stdout
```
AAFREEN KHAN 21BCS1397
```

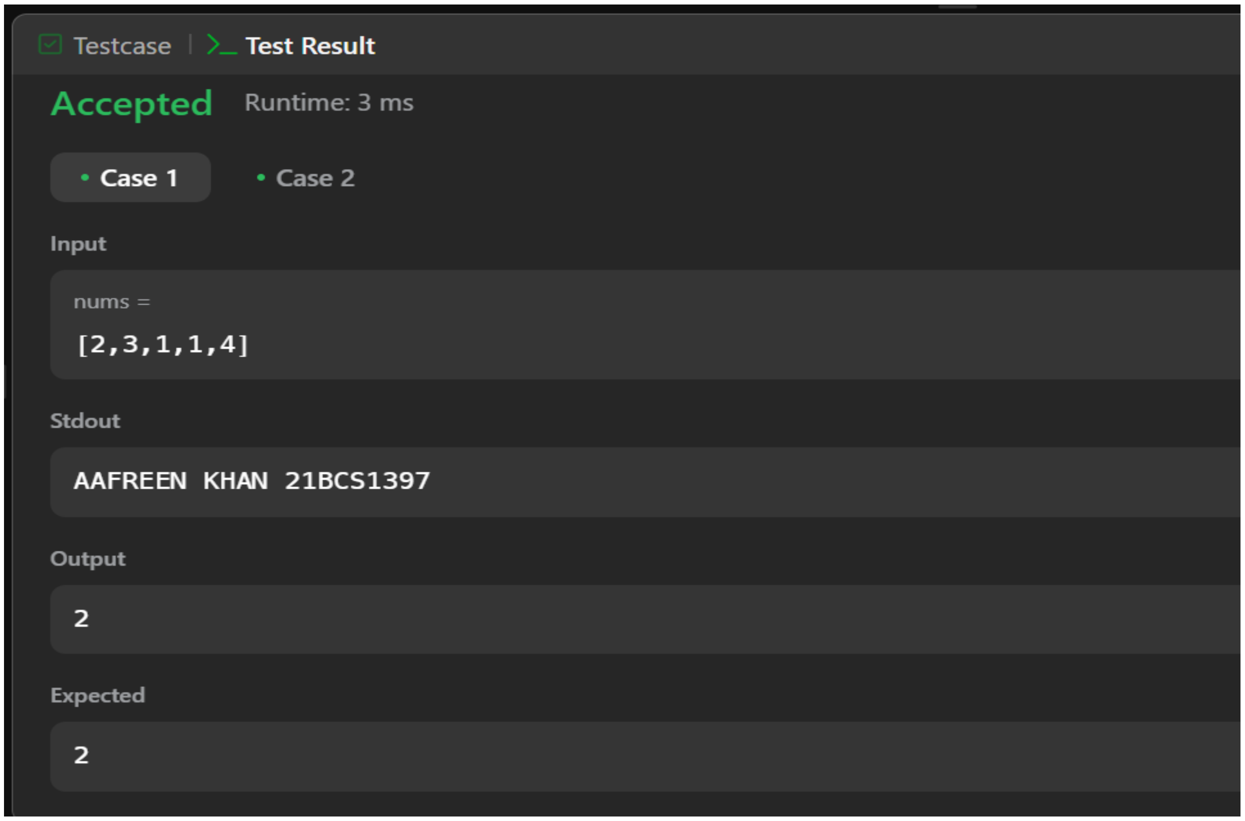
Output
```
[[-1,-1,2],[-1,0,1]]
```

Expected
```
[[-1,-1,2],[-1,0,1]]
```

Code(B):

```
class Solution {
public:
    int jump(vector<int>& nums) {

        cout<<"AAFREEN KHAN"<<"  "<<"21BCS1397";

        for(int i=1;i<nums.size();i++){
            nums[i]=max(nums[i]+1,nums[i-1]);
        }
        int ant=0;
        int ans=0;
        while(ant<nums.size()-1){
            ans++;
            ant=nums[ant];
        }
        return ans;
    }
};
```

Output(B):



```
Accepted   Runtime: 3 ms

Input
nums =
[2,3,1,1,4]

Stdout
AAFREEN KHAN 21BCS1397

Output
2

Expected
2
```

Experiment 1.2

Student Name: AAFREEN KHAN **UID: 21BCS 1397**
Branch:BE-CSE **Section/Group: 606-A**
Semester: 6TH **Date of Performance:19-01-2023**
Subject Name: AP Lab **Subject Code: 21CSP-351**

1. Aim: To demonstrate the concept of String Matching algorithms.

2. Objective: To learn about string matching algorithms

3. Script and Output:

1. Problem statement - Given two strings s and goal, return true if and only if s

can become goal after some number of shifts on s.A shift on s consists of moving the leftmost character of s to the rightmost

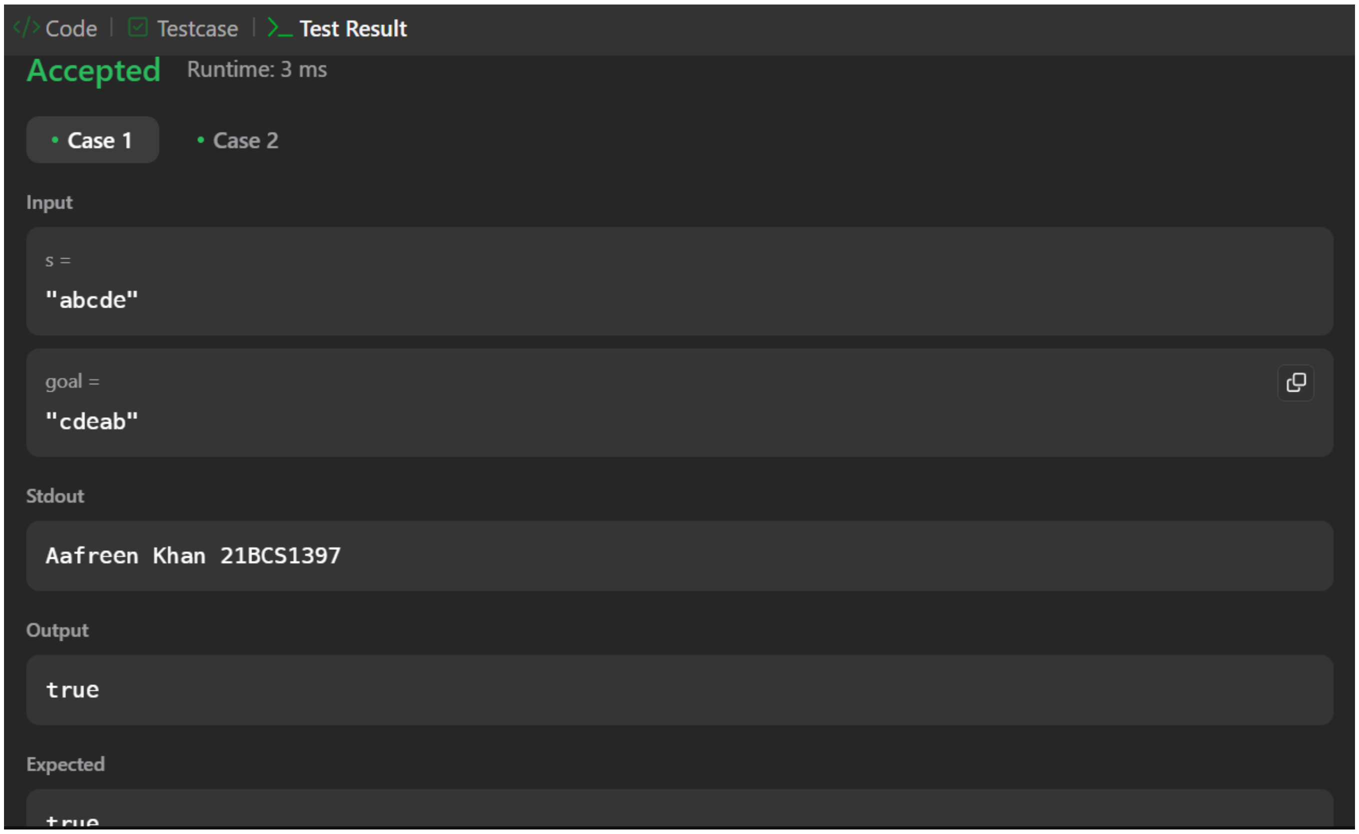
position.

For example, if s = "abcde", then it will be "bcdea" after one shift.

Code:

```
class Solution {
bool solve(queue<int> queOne, queue<int> queTwo, int size){
    while(size--)
    {
        int front = queTwo.front();
        queTwo.pop();
        queTwo.push(front);
        if(queOne == queTwo){
            return true;
        }
    }
    return false;
}
public:
    bool rotateString(string s, string goal) {
        queue<int> queOne;
        queue<int> queTwo;
        if(s.size() != goal.size()){
            return false;
        }
        cout<<"Aafreen Khan 21BCS1397";
        for(int i = 0; i < s.size(); i++){
            queOne.push(s[i]);
        }
        for(int i = 0; i < goal.size(); i++){
            queTwo.push(goal[i]);
        }
        int size = goal.size();
        return solve(queOne, queTwo, size);
    }
};
```

Output:



Accepted Runtime: 3 ms

Case 1 Case 2

Input

s =
"abcde"

goal =
"cdeab"

Stdout

```
Aafreen Khan 21BCS1397
```

Output

```
true
```

Expected

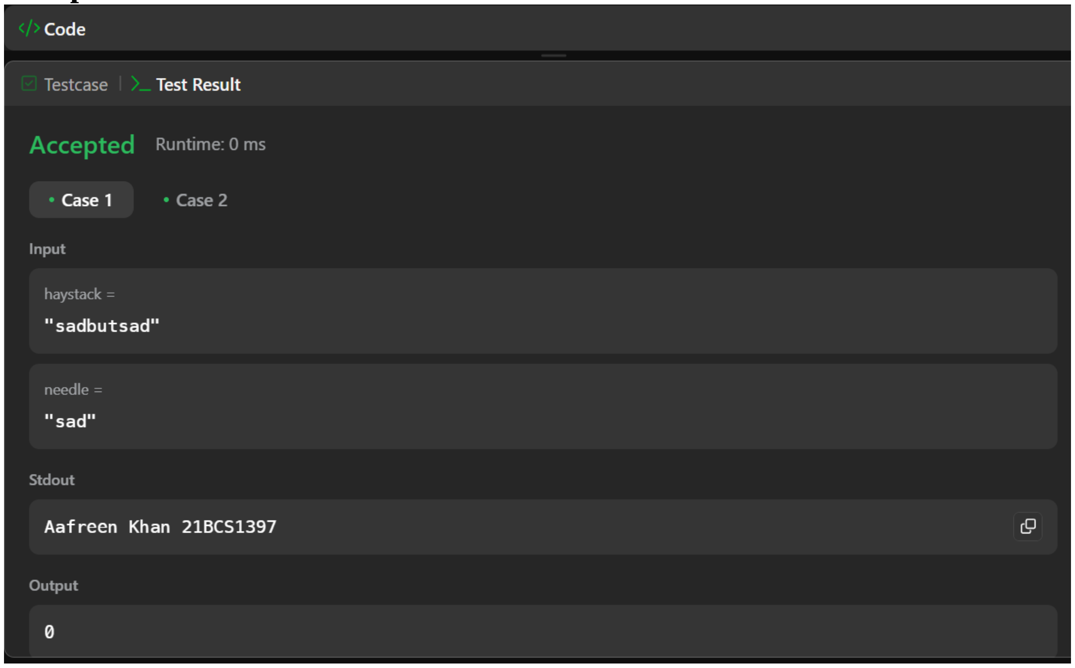
```
true
```

2. Problem statement: Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

Code:

```
class Solution {
public:
    int strStr(string haystack, string needle) {
        int n=haystack.size();
        int m=needle.size();
        for(int i=0;i<n;i++){
            if(haystack[i]==needle[0]){
                bool flag=true;
                cout<<"Aafreen Khan
                21BCS1397";
                for(int k=0;k<m;k++){
                    if(needle[k]!=haystack[i+k]){
                        flag=false;
                        break;
                    }
                }
                if(flag)
                return i;
            }
        }
        return -1;
    }
};
```

Output:



Accepted Runtime: 0 ms

Case 1 Case 2

Input

haystack =
"sadbutsad"

needle =
"sad"

Stdout

```
Aafreen Khan 21BCS1397
```

Output

```
0
```

Experiment 1.3

Student Name: Aafreen Khan **UID:** 21BCS1397
Branch: BE-CSE **Section/Group:** CC-606-A
Semester: 6th **Date of Performance:** 02-02-2024
Subject Name: Advance Programming-2 **Subject Code:** 21CSP-251

1. Aim:

- To Solve the Last Stone Weight.
- To Solve the Cheapest Flight Booking with K stops.

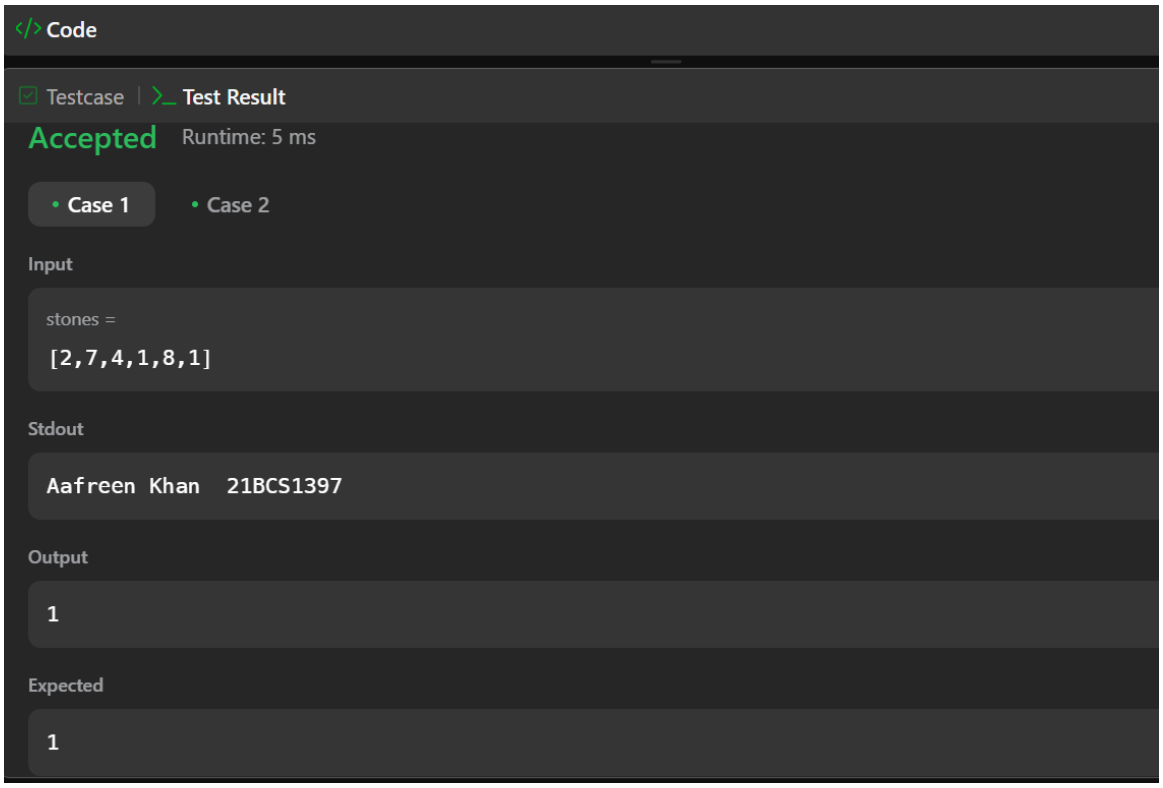
2. Objective:

- You are given an array of integers stones where stones[i] is the weight of the ith stone. We are playing a game with the stones. On each turn, we choose the heaviest two stones and smash them together. Suppose the heaviest two stones have weights x and y with x <= y.
- There are n cities connected by some number of flights. You are given an array flights where flights[i] = [fromi, toi, pricei] indicates that there is a flight from city fromi to city toi with cost pricei.

3. Algo. /Approach and output:

1st:

```
class Solution {
public:
    int lastStoneWeight(vector<int>& stones)
    {
        int a,b;
        priority_queue<int>pq;
        for(int i=0;i<stones.size();i++)
        {
            pq.push(stones[i]);
        }
        while(pq.size()!=1)
        {
            a=pq.top();
            pq.pop();
            b=pq.top();
            pq.pop();
            int c=a-b;
            pq.push(c);
        }
        return pq.top();
    }
};
```



Code

Testcase | Test Result

Accepted Runtime: 5 ms

Case 1 Case 2

Input

stones =

```
[2,7,4,1,8,1]
```

Stdout

```
Aafreen Khan  21BCS1397
```

Output

```
1
```

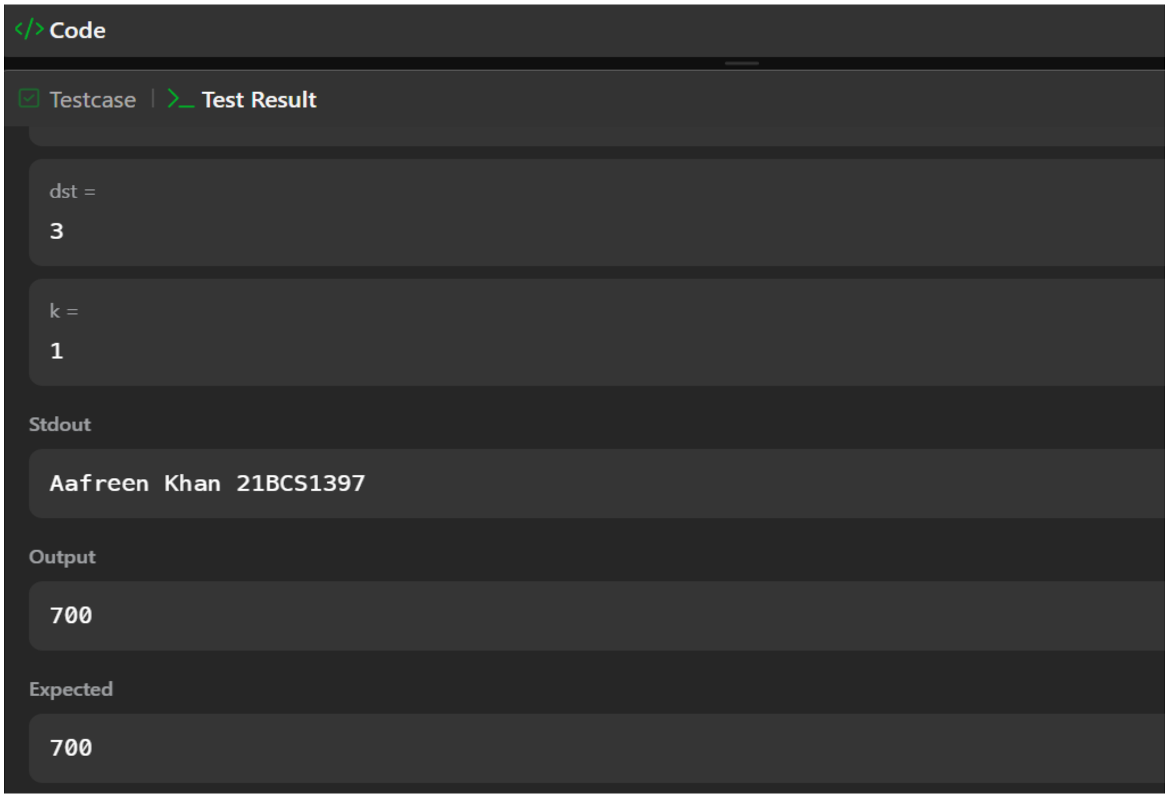
Expected

```
1
```

2nd:

```
class Solution {
public:
    int findCheapestPrice(int n, vector<vector<int>>& flights, int src, int dst, int k) {
        vector<vector<pair<int,int>>> adj(n,vector<pair<int,int>>{});
        for(auto x:flights){
            adj[x[0]].push_back({x[1],x[2]});
        }

        queue<pair<int,pair<int,int>>> q;
        vector<int> dist(n,1e9);
        dist[src]=0;
```

```
            q.push({0,{src,0}});

            while(!q.empty()){
                auto front=q.front();
                q.pop();
                int stops=front.first;
                int cost=front.second.second;
                int node=front.second.first;

                if(stops>k){continue;}

                for(auto it:adj[node]){
                    if(cost+it.second < dist[it.first] && stops<=k){
                        dist[it.first]=cost+it.second;
                        q.push({stops+1,{it.first,dist[it.first]}});
                    }
                }
            }
            if(dist[dst]!=1e9){return dist[dst];}
            return -1;
        }
};
```



```
dst =
3

k =
1

Stdout
Aafreen Khan 21BCS1397

Output
700

Expected
700
```

Experiment - 1.4

Student Name: Aafreen Khan **UID: 21BCS1397**
Branch: BE-CSE **Section/Group: 606 'A'**
Date of Performance: 16th Feb 2024 **Semester: 6**
Subject Name: Advanced Programming Lab - II **Subject Code: 21CSP-351**

1. **Aim**:
   - To Solve Missing Number.
   - To Solve longest duplicate substring.

2. **Objective**:

   1) Problem statement - Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

   2) Problem Statement - Given a string s, consider all duplicated substrings: (contiguous) substrings of s that occur 2 or more times. The occurrences may overlap.
   Return any duplicated substring that has the longest possible length. If s does not have a duplicated substring, the answer is "".

3. **Script and Output**:

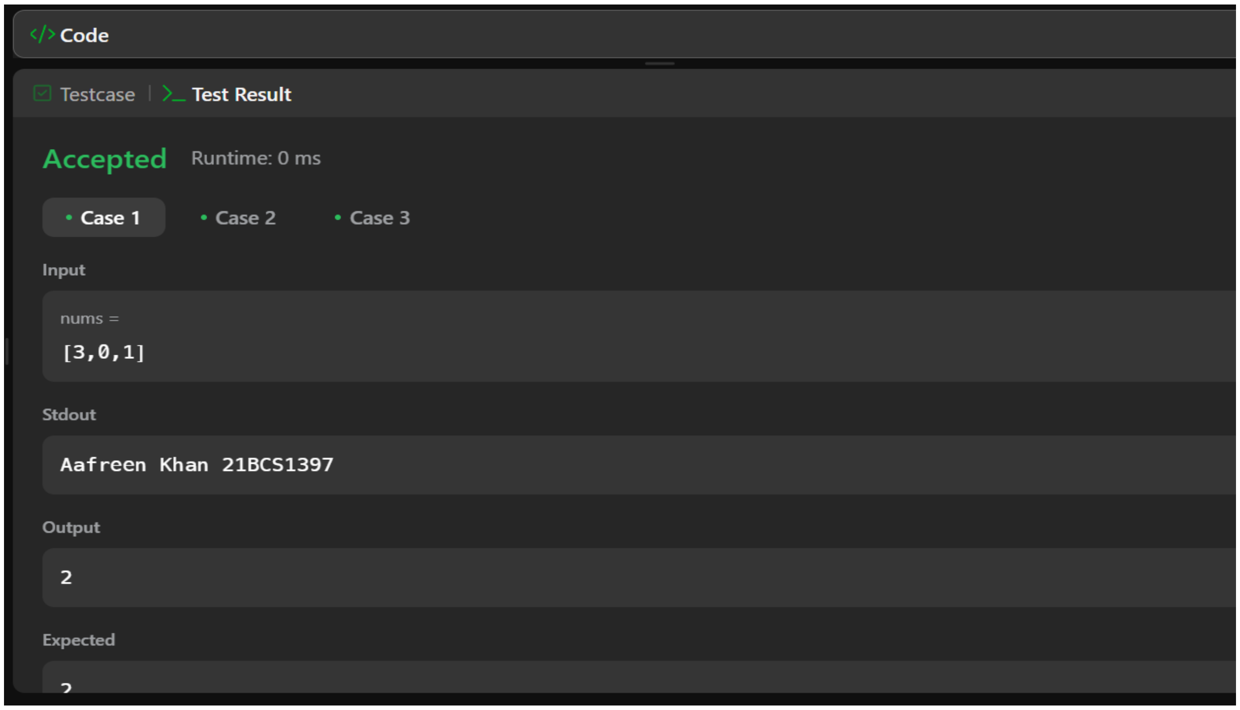
   Problem 1:

Code:

```
class Solution {
   public int missingNumber(int[] nums) {
      cout<<"Aafreen Khan 21BCS1397";
      int n = nums.length;
       int[] v = new int[n+1];
       Arrays.fill(v, -1);
       for(int i = 0; i < nums.length; i++) {
           v[nums[i]] = nums[i];
```

```
            }
            for(int i = 0; i < v.length; i++) {
                if(v[i] == -1) return i;
            }
            return 0;
        }
    }
```

OUTPUT:


Code
Testcase | Test Result
Accepted Runtime: 0 ms
• Case 1 • Case 2 • Case 3
Input
```
nums =
[3,0,1]
```
Stdout
```
Aafreen Khan 21BCS1397
```
Output
```
2
```
Expected
```
2
```

Time and Space Complexity:

 The time complexity of this program is O(N).
The space complexity is O(1).

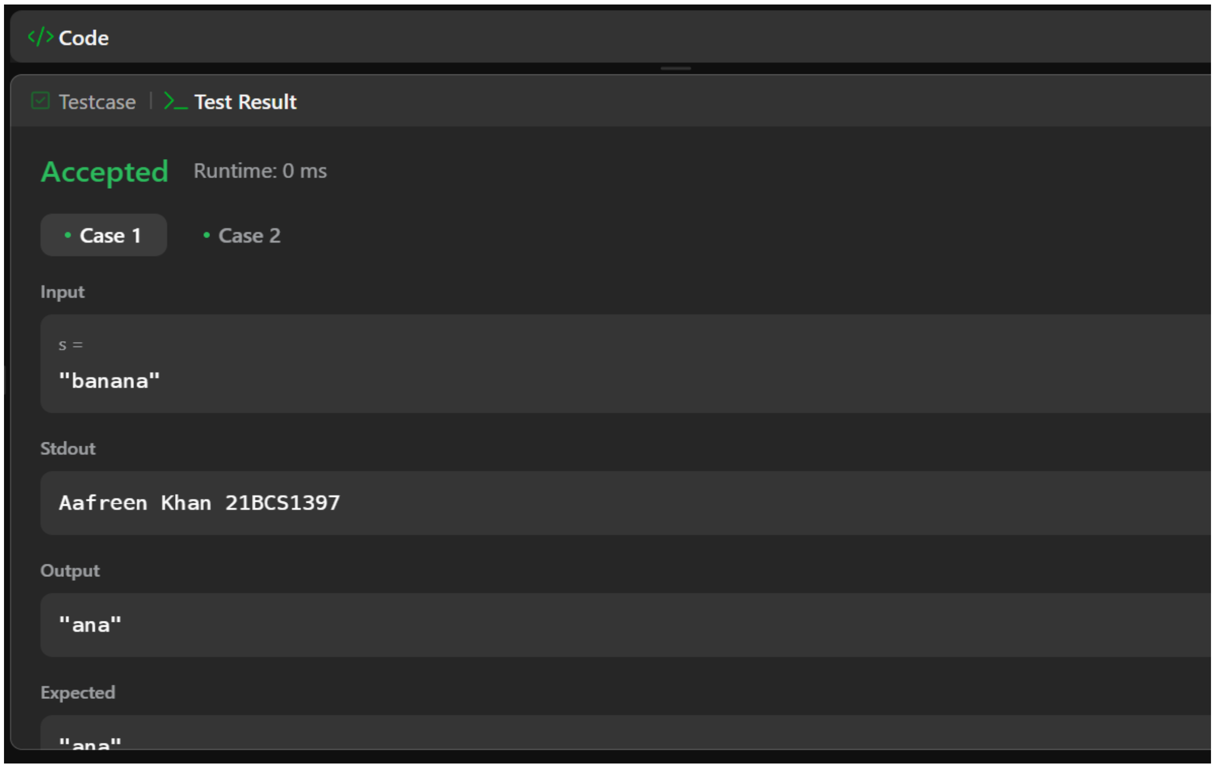
Problem 2:

Code:

```
class Solution {
String str;
int len;
```

```
private class SubStr {
    private int hash;
    private int off;


    public SubStr(int off, int hash) {
    cout<<"Aafreen Khan 21BCS1397";
    this.off = off;
        this.hash = hash;
        //System.out.println(off+" "+hash);
    }

    public int hashCode() {
        return this.hash;
    }

    public boolean equals(Object o) {
        SubStr x = (SubStr)o;
        if (x.off == this.off) {
            return true;
        }

        if (x.hash != this.hash) {
            return false;
        }
        for (int i=0; i<len; i++) {
            if (str.charAt(x.off+i) != str.charAt(this.off+i)) {
                return false;
            }
        }
        return true;
    }

    public int getOffset() {
        return this.off;
    }


    public String toString() {
        return str.substring(this.off, this.off+len);
    }
}

private String checkDup(int len) {
    this.len = len;
    HashSet<SubStr> s = new HashSet();
    int hash = 0;
    int p = 1;

    for (int i=0; i<len; i++) {
        p = p * 33;

        hash = hash * 33 + str.charAt(i);

    }

    s.add(new SubStr(0, hash));

    for (int i=len; i<str.length(); i++) {

        hash = hash * 33 - str.charAt(i-len)*p + str.charAt(i);
        SubStr ss = new SubStr(i-len+1, hash);
        if (s.contains(ss)) {
            return ss.toString();
        } else {

            s.add(ss);

        }

    }

    return null;

}

public String longestDupSubstring(String S) {
    this.str = S;
    //System.out.println(checkDup(3));

    String r = "";
    int s = 0;
    int e = S.length();
    while (s < e) {
        int m = (s+e)/2;

        String tmp = checkDup(m);
        if (tmp != null) {
                r = tmp;
                s = m+1;

            } else {

                e = m;

            }

        }

        return r;

    }

}
```

OUTPUT:



Code

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
s =
"banana"
```

Stdout

```
Aafreen Khan 21BCS1397
```

Output

```
"ana"
```

Expected

```
"ana"
```

<u>**Time and Space complexity:**</u>

- The time complexity of the code O(n log n)
- Space complexity of the code is O(n)

Experiment-2.1

Student Name: Aafreen Khan **UID:** 21BCS1397
Branch: BE-CSE **Section/Group:**CC-606 A
Semester: 6th **Date of Performance:** 23-02-2024
Subject Name: Advanced Programming lab-2 **Subject Code:**21CSP-351

Aim:

- Same Tree
- Symmetric Tree

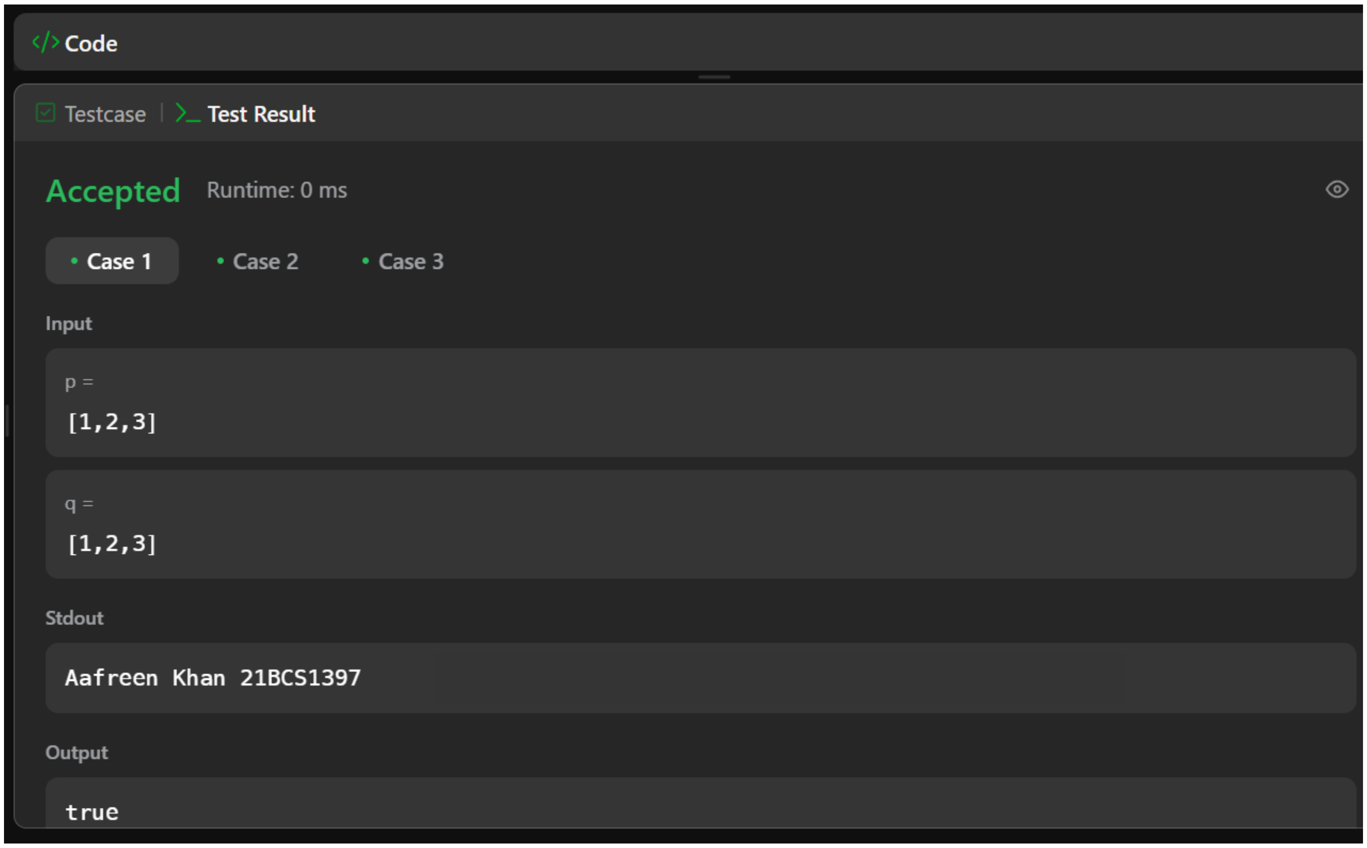
Objective:

- Given the roots of two binary trees p and q, write a function to check if they are the same or not.Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.
- Given the root of a binary tree, check whether it is a mirror of itself (i.e., symmetric around its center).

Code(A):

```
public class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode() {}

    TreeNode(int val) {
        this.val = val;
    }

    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
class Solution {
    public boolean isSameTree(TreeNode p, TreeNode q) {
        if (p == null && q == null) return true;
        if (p == null || q == null) return false;
        return (p.val == q.val) && isSameTree(p.left, q.left) && isSameTree(p.right, q.right);
    }
}
```

Output(A):



Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

```
p =
[1,2,3]
q =
[1,2,3]
```

Stdout

```
Aafreen Khan 21BCS1397
```

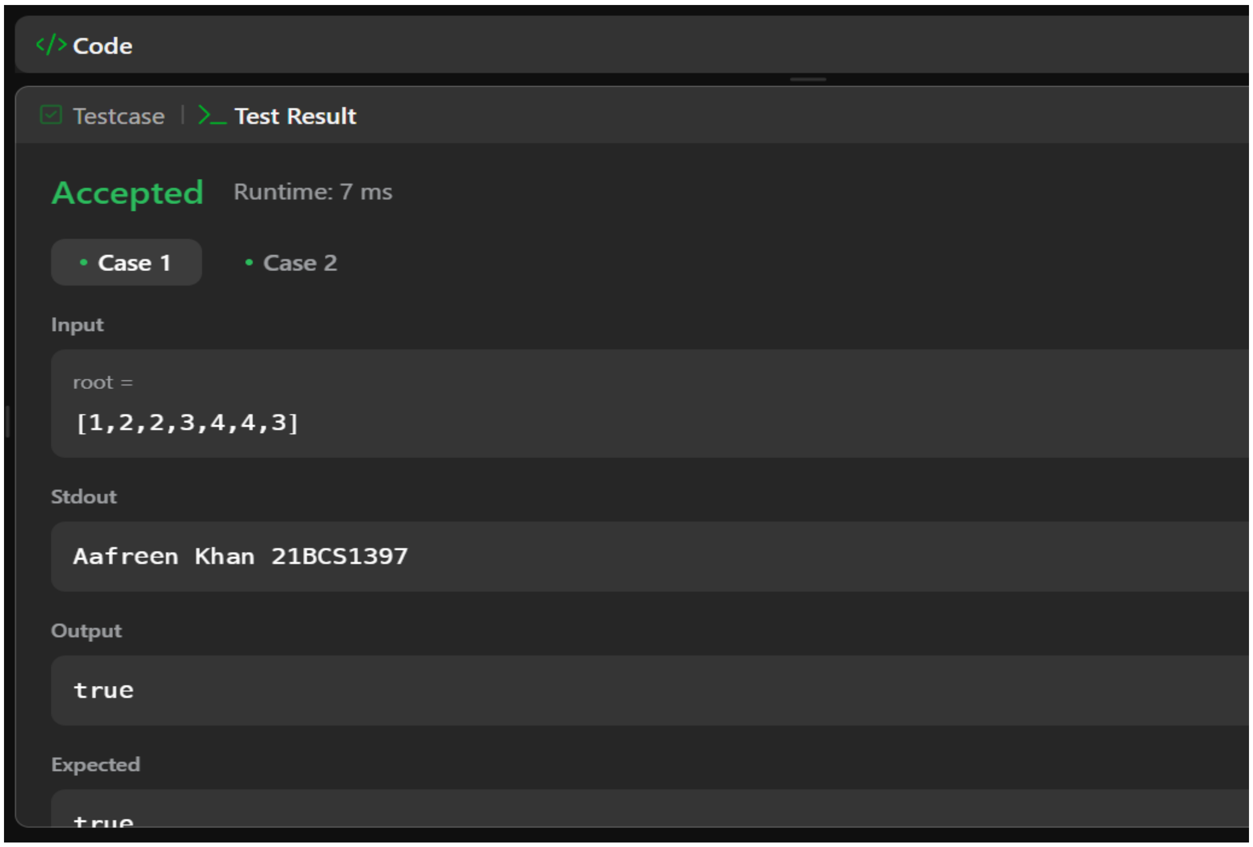
Output

```
true
```

Code(B):

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null)
            return true;
        return isMirror(root.left, root.right);
    }

    private boolean isMirror(TreeNode left, TreeNode right) {
        if (left == null && right == null)
            return true;
        if (left == null || right == null)
            return false;
        return (left.val == right.val)
            && isMirror(left.left, right.right)
            && isMirror(left.right, right.left);
    }
}
```

Output(B):



Accepted Runtime: 7 ms

• Case 1 • Case 2

Input

root =
```
[1,2,2,3,4,4,3]
```

Stdout
```
Aafreen Khan 21BCS1397
```

Output
```
true
```

Expected
```
true
```

Experiment – 2.2

Student Name: Aafreen Khan **UID:** 21BCS1397
Branch: BE-CSE **Section/Group**: 606‘A’
Date of Performance: 1st Mar 2024 **Semester:** 6
Subject Name: Advanced Programming Lab - II **Subject Code:** 21CSP-351

1. Aim:

- To Solve the skyline problem.
- To Solve is graph bipartite.

2. Objective:

1) **Problem statement –** A city's skyline is the outer contour of the silhouette formed by all the buildings in that city when viewed from a distance. Given the locations and heights of all the buildings, return the skyline formed by these buildings collectively.

2) **Problem Statement –** There is an undirected graph with n nodes, where each node is numbered between 0 and n - 1. You are given a 2D array graph, where graph[u] is an array of nodes that node u is adjacent to. More formally, for each v in graph[u], there is an undirected edge between node u and node v. Return true if and only if it is bipartite.

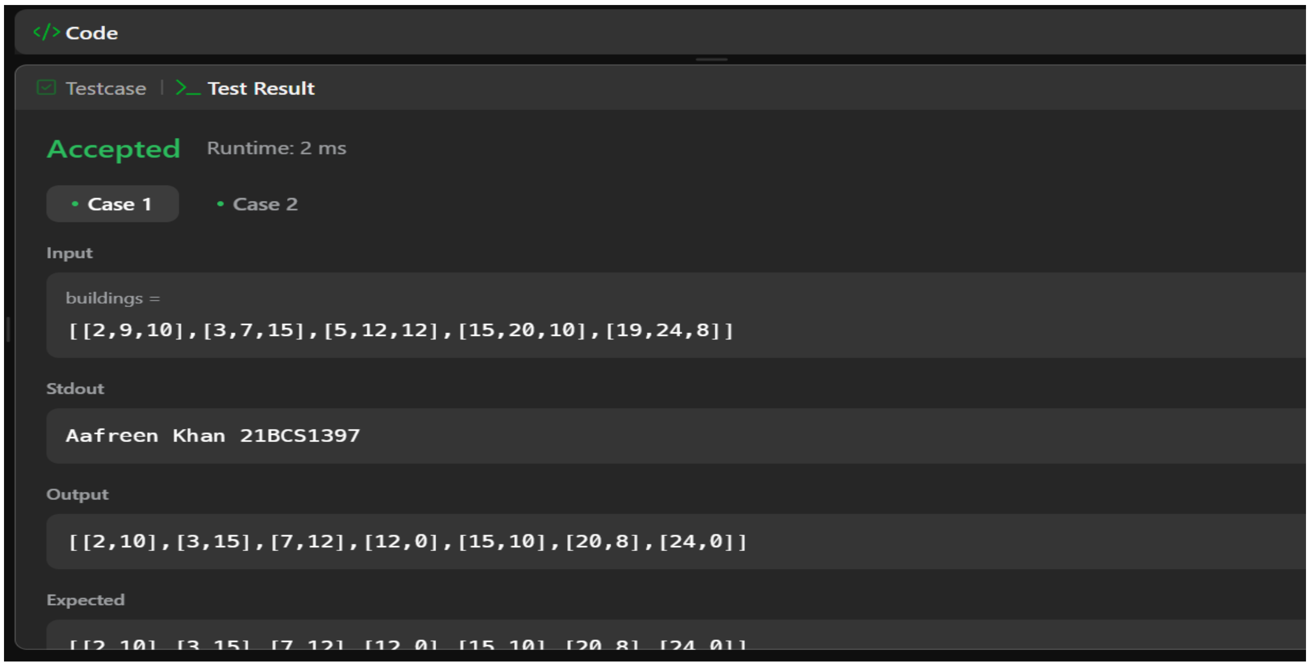
3. Code:

Problem 1:

```
class Solution {
public:
    vector<vector<int>> getSkyline(vector<vector<int>>& buildings) {
        vector<vector<int>> ans;
        multiset<int> pq{0};

        vector<pair<int, int>> points;

        for(auto b: buildings){
            points.push_back({b[0], -b[2]});
            points.push_back({b[1], b[2]});
        }

        sort(points.begin(), points.end());
```

```
            int ongoingHeight = 0;

            // points.first = x coordinate, points.second = height
            for(int i = 0; i < points.size(); i++){
                int currentPoint = points[i].first;
                int heightAtCurrentPoint = points[i].second;

                if(heightAtCurrentPoint < 0){
                    pq.insert(-heightAtCurrentPoint);
                } else {
                    pq.erase(pq.find(heightAtCurrentPoint));
                }

                // after inserting/removing heightAtI, if there's a change
                auto pqTop = *pq.rbegin();
                if(ongoingHeight != pqTop){
                    ongoingHeight = pqTop;
                    ans.push_back({currentPoint, ongoingHeight});
                }
            }

            return ans;
        }
    };
```

Output:



Accepted Runtime: 2 ms

Input

buildings =
```
[[2,9,10],[3,7,15],[5,12,12],[15,20,10],[19,24,8]]
```

Stdout
```
Aafreen Khan 21BCS1397
```

Output
```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

Expected
```
[[2,10],[3,15],[7,12],[12,0],[15,10],[20,8],[24,0]]
```

Time and Space Complexity:

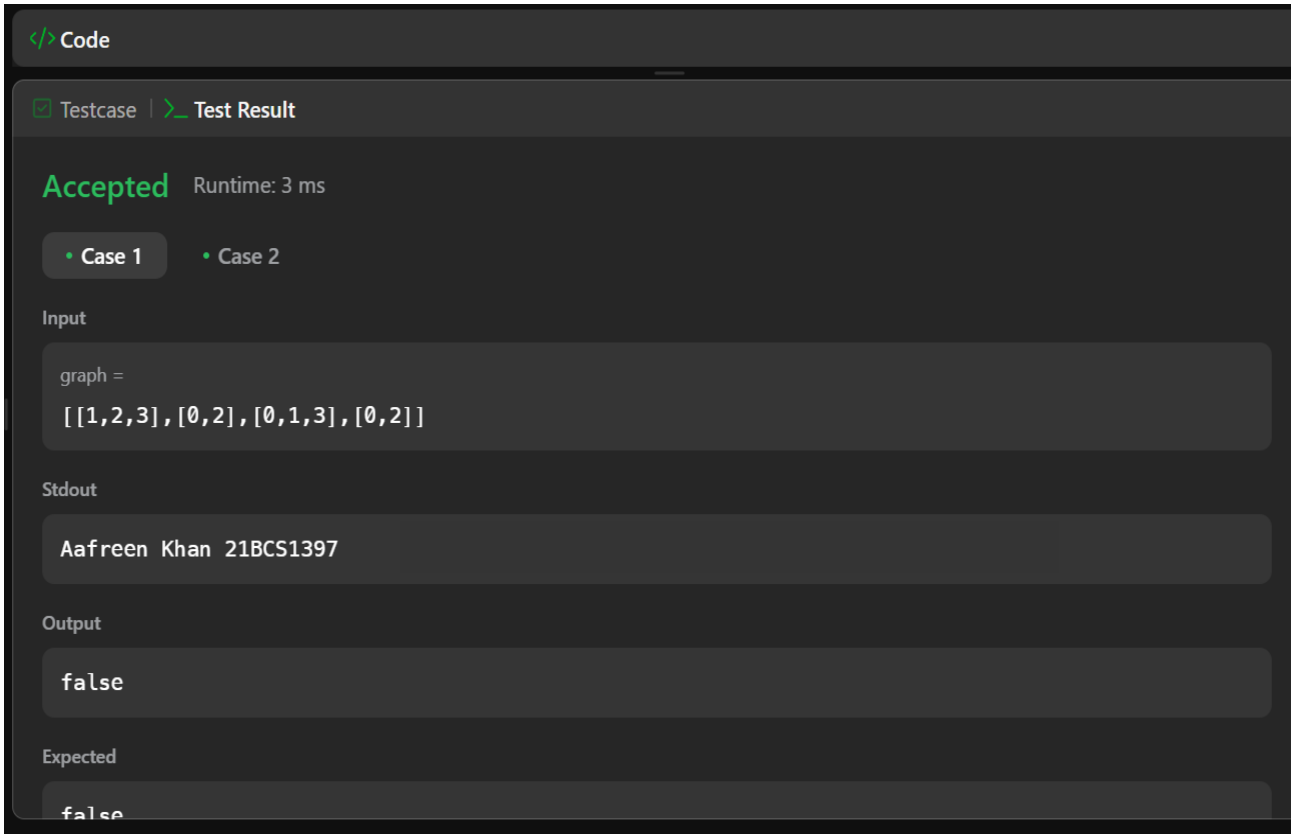
- The time complexity of this program is O(NlogN).
- The space complexity is O(N).

Problem 2:

Code:

```
class Solution {
public:
    bool isBipartite(vector<vector<int>>& graph) {
        int len = graph.size();
        stack<int> s;
        vector<int> vis(len);
        for (int i = 0; i < len; i++) {
            if (vis[i] > 0) continue;
            vis[i] = 1;
            s.push(i);
            while (s.size() > 0) {
                int curr = s.top();
                s.pop();
                vector<int> edges = graph[curr];
                for (int next:edges)
                    if (vis[next] == 0) {
                        vis[next] = vis[curr] ^ 3;
                        s.push(next);
                    } else if (vis[curr] == vis[next]) return false;
            }
        }
        return true;
    }
};
```

Output:



Input

```
graph =
[[1,2,3],[0,2],[0,1,3],[0,2]]
```

Stdout

```
Aafreen Khan 21BCS1397
```

Output

```
false
```

Expected

```
false
```

Time and Space complexity:

- The overall time complexity of the code O(V+E)
- Space complexity of the code is O(V).

Experiment 2.3

Student Name: Aafreen Khan	**UID:** 21BCS1397
Branch: CSE	**Section/Group:** 606 A
Semester: Sixth	**Date of Performance: 06.04.24**
Subject Name: AP Lab - 2	**Subject Code:** 21CSP-351

Aim: To demonstrate the concept of Divide and Conquer.

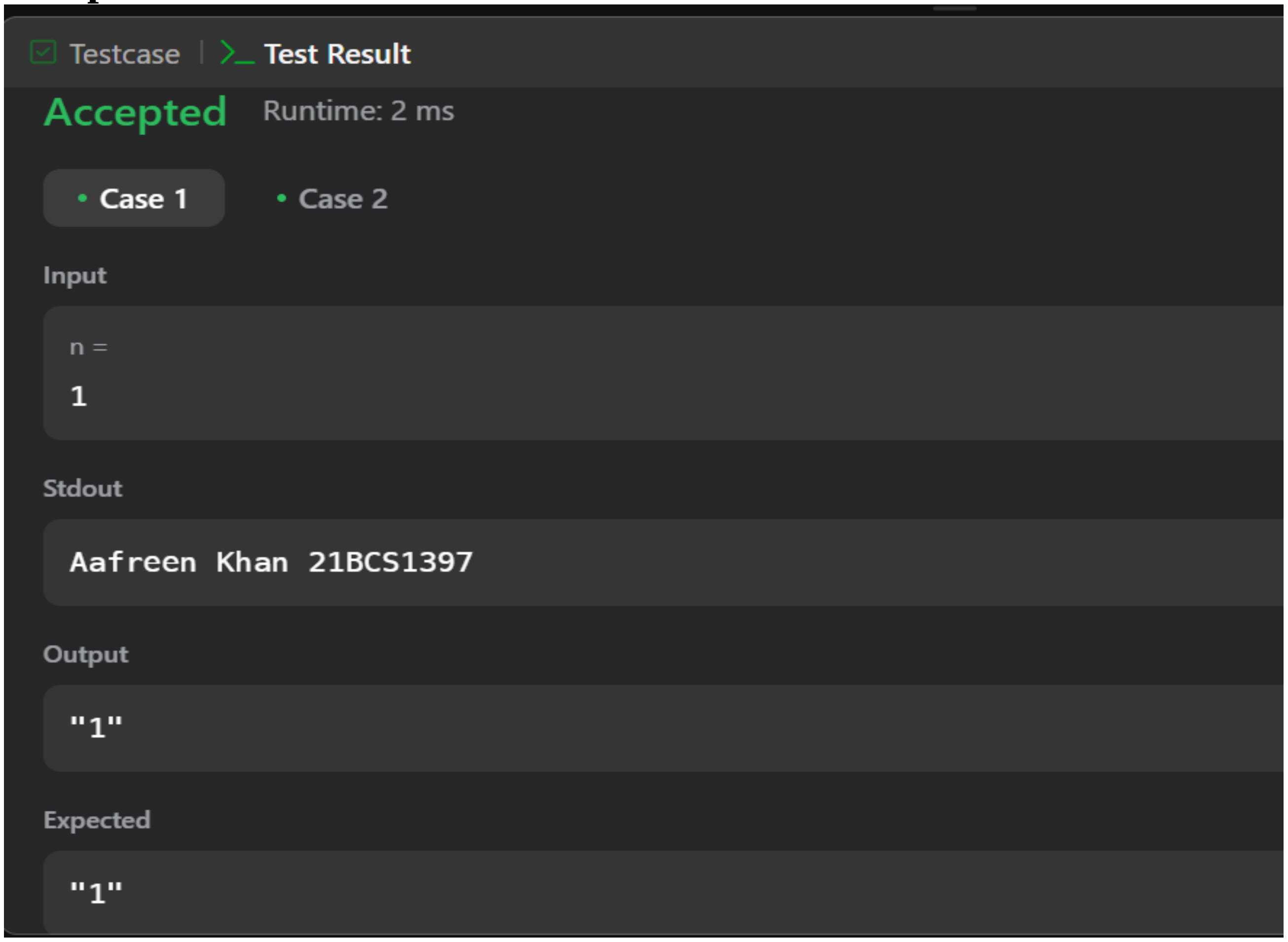
Objective:

Problem 1: Count&Say: The count-and-say sequence is a sequence of digit strings.

Code:

```
class Solution {
  public String countAndSay(int n) {
  cout<<"Aafreen Khan 21BCS1397";
  StringBuilder current = new StringBuilder("1");

    for (int i = 1; i < n; i++) {
      StringBuilder next = new StringBuilder();
      int count = 1;

      for (int j = 1; j < current.length(); j++) {
        if (current.charAt(j) == current.charAt(j - 1)) {
          count++;
        } else {
          next.append(count).append(current.charAt(j - 1));
          count = 1;
        }
      }

      next.append(count).append(current.charAt(current.length() - 1));
      current = next;
    }

    return current.toString();
  }
}
```

Output:



Testcase | >_ Test Result

Accepted Runtime: 2 ms

• Case 1 • Case 2

Input

```
n =
1
```

Stdout

```
Aafreen Khan 21BCS1397
```

Output

```
"1"
```

Expected

```
"1"
```

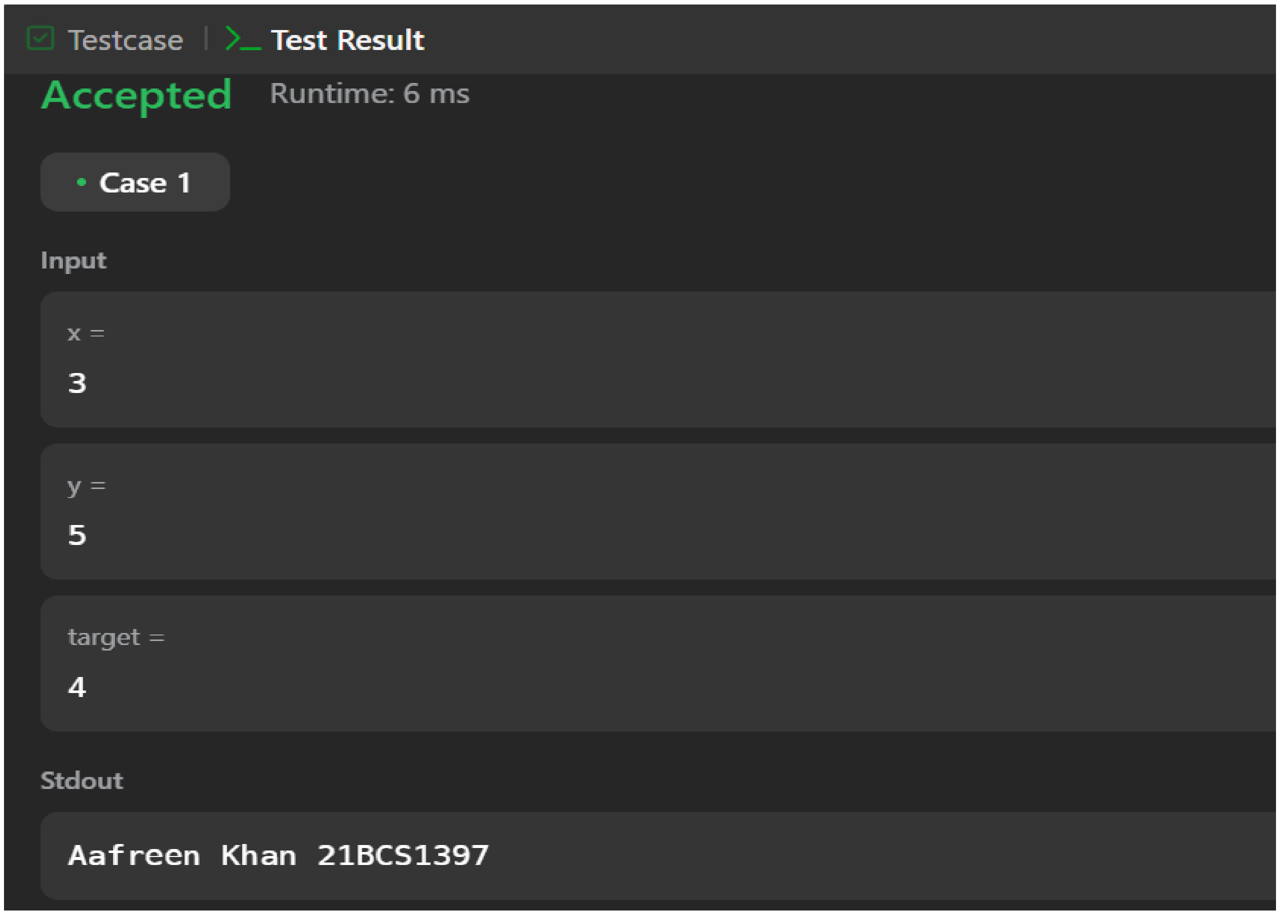
Problem 2: Longest Duplicate Substring: You are given two jugs with capacities jug 1 Capacity and jug 2 Capacity liters.
There is an infinite amount of water supply available.
Determine whether it is possible to measure exactly targetCapacity liters using these two jugs.

Code:

```
class Solution {
  public String longestDupSubstring(String s) {
  final int kMod = 1_000_000_007;
    final int n = s.length();
    int[] pows = new int[n];
    int bestStart = -1;
    int l = 1;
    int r = n;

    pows[0] = 1;
    for (int i = 1; i < n; ++i)
      pows[i] = (int) ((pows[i - 1] * 26L) % (long) kMod);

    while (l < r) {
      final int m = (l + r) / 2;
      final int start = getStart(s, m, pows, kMod);
      if (start == -1) {
        r = m;
      } else {
        bestStart = start;
        l = m + 1;
      }
    }

    if (bestStart == -1)
      return "";
    if (getStart(s, l, pows, kMod) == -1)
      return s.substring(bestStart, bestStart + l - 1);
      return s.substring(bestStart, bestStart + l);
  }
```

```
    private int getStart(final String s, int k, int[] pows, int kMod) {
        Map<Long, List<Integer>> hashToStarts = new HashMap<>();
        long h = 0;

        for (int i = 0; i < k; ++i)
            h = ((h * 26) % kMod + val(s.charAt(i))) % kMod;
        hashToStarts.put(h, new ArrayList<>());
        hashToStarts.get(h).add(0);

        for (int i = k; i < s.length(); ++i) {
            final int startIndex = i - k + 1;
            h = ((h - (long) (pows[k - 1]) * val(s.charAt(i - k))) % kMod + kMod) %
kMod;
            h = (h * 26 + val(s.charAt(i))) % kMod;
            if (hashToStarts.containsKey(h)) {
                final String currSub = s.substring(startIndex, startIndex + k);
                for (final int start : hashToStarts.get(h))
                    if (s.substring(start, start + k).equals(currSub))
                        return startIndex;
            }
            hashToStarts.put(h, new ArrayList<>());
            hashToStarts.get(h).add(startIndex);
        }

        return -1;
    }

    private int val(char c) {
        return c - 'a';
    }
    }
```

Output:



```
Accepted   Runtime: 6 ms

• Case 1

Input
x =
3
y =
5
target =
4

Stdout
Aafreen Khan 21BCS1397
```

Course Outcomes:

• Proficiency in applying Divide and Conquer methods to solve problems.
• Develop skills in algorithmic problem-solving using Divide and Conquer.
• Application of Divide-Conquer based DS for solving real-world problems.
• Enhancing ability to analyze and optimize time and space complexity through the application of Divide and Conquer.

Experiment 3.1

Student Name: Aafreen Khan **UID:** 21BCS1397
Branch: CSE **Section/Group:** 606 A
Semester: Sixth **Date of Performance:** 05.04.24
Subject Name: AP Lab - 2 **Subject Code:** 21CSP-351

Aim: To demonstrate the concept of Greedy approach.

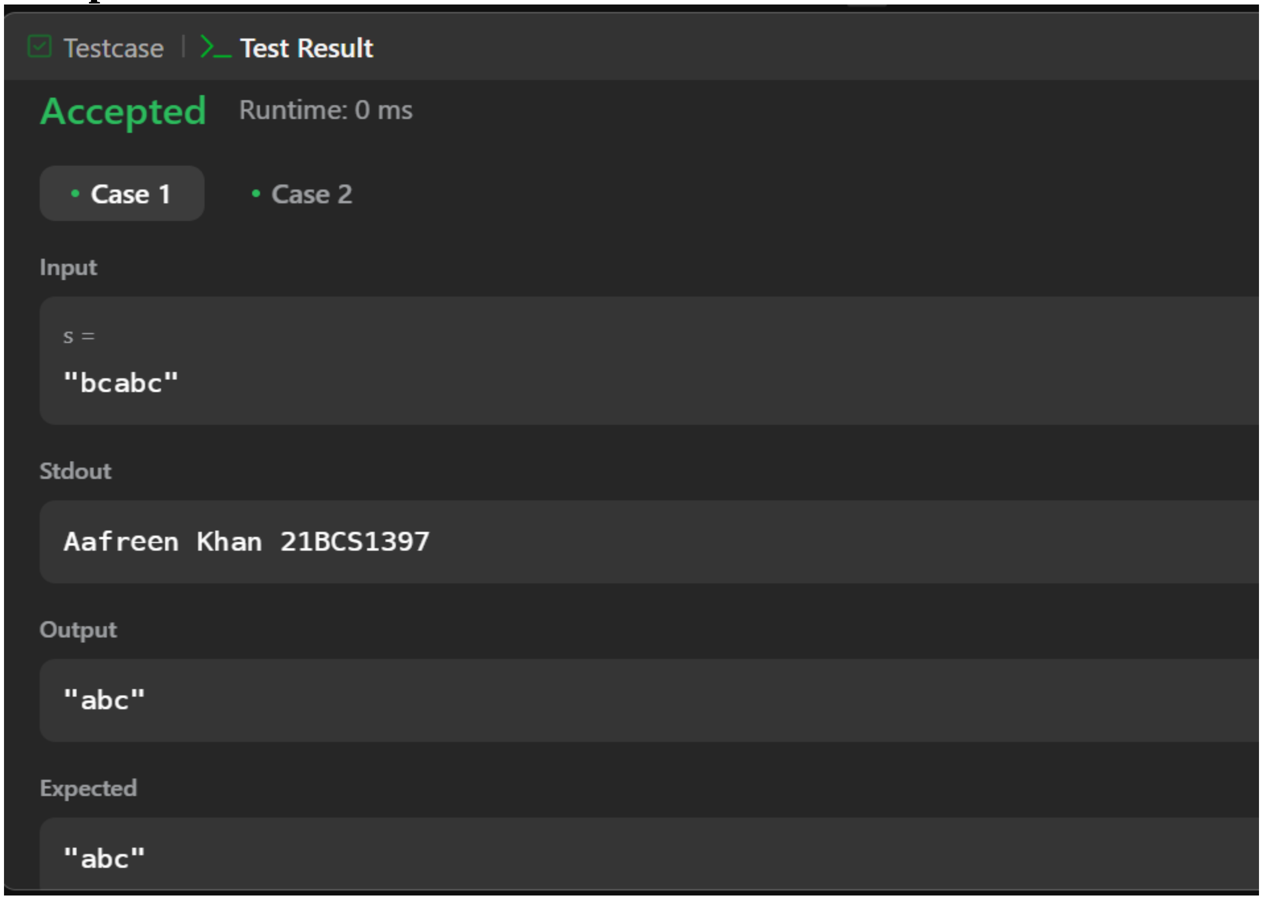
Objective:

Problem 1: RemoveDuplicateLetters: Given a string s, remove duplicate letters so that every letter appears once and only once. Youmust make sure your result is the smallest in lexicographical order among all possible results.

Code:

```
class Solution {
    public String removeDuplicateLetters(String s) {
    cout<<"Aafreen Khan 21BCS1397";
    StringBuilder sb = new StringBuilder();
        int[] count = new int[128];
        boolean[] used = new boolean[128];

        for (char c : s.toCharArray())
            ++count[c];

        for (char c : s.toCharArray()) {
            --count[c];
            if (used[c]) continue;
            while (sb.length() > 0 && sb.charAt(sb.length() - 1) > c &&
count[sb.charAt(sb.length() - 1)] > 0) {
                used[sb.charAt(sb.length() - 1)] = false;
                sb.setLength(sb.length() - 1);
            }
            used[c] = true;
            sb.append(c);
        }

        return sb.toString();
    }
}
```

Output:



Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
s =
"bcabc"
```

Stdout

```
Aafreen Khan 21BCS1397
```

Output

```
"abc"
```

Expected

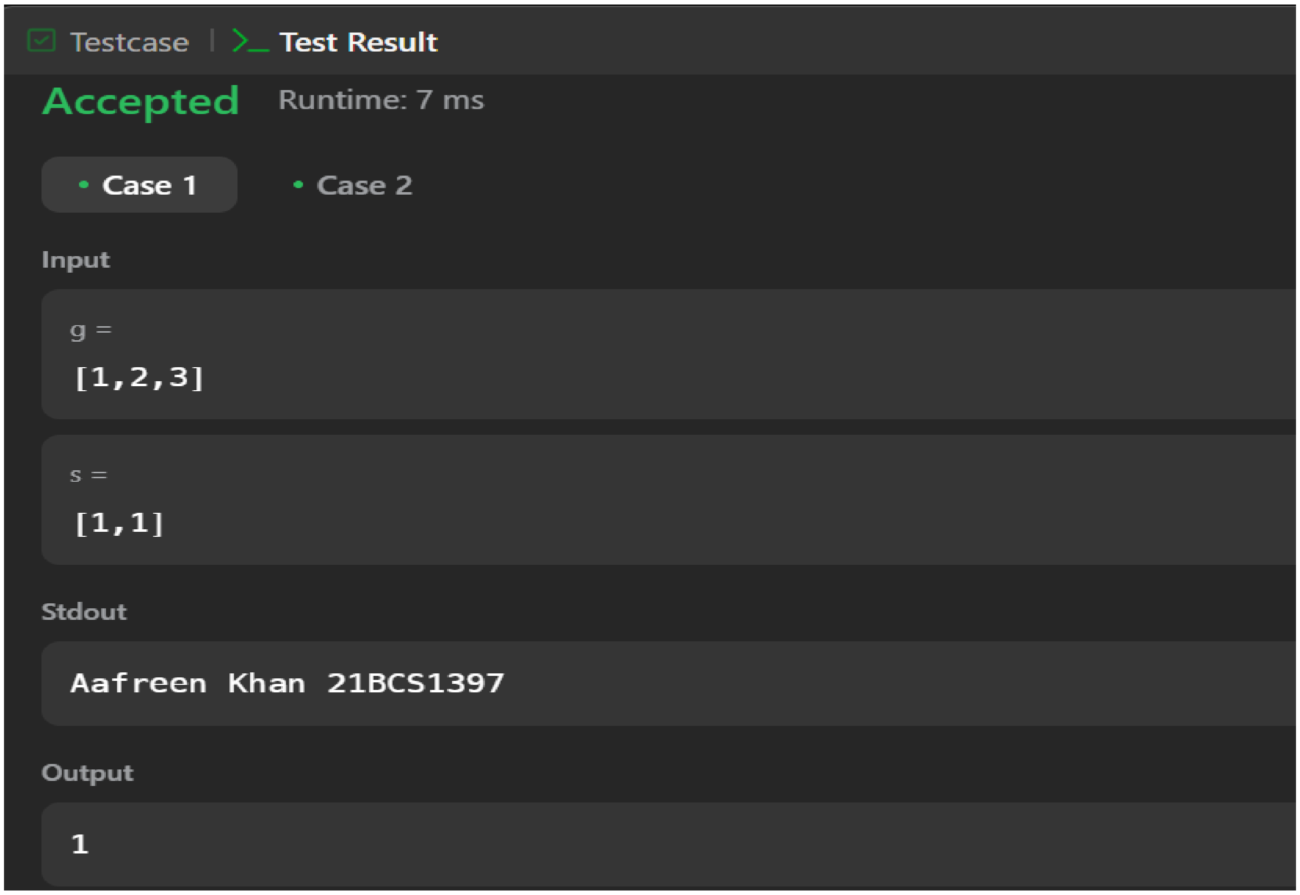
```
"abc"
```

Problem 2: Assign Cookies: Assume you are an awesome parent andwant to give your children some cookies. But, you should give each child at most one cookie. Each child i has a greed factor g[i], which is the minimum size of a cookie that the child will be content with; and each cookie j has a size s[j]. If s[j] >= g[i], we can assign the cookie j to the child i, and the child
i will be content. Your goal is to maximize the number of your contentchildren and output the maximum number.

Code:

```
class Solution {
  public int findContentChildren(int[] g, int[] s) {
    cout<<"Aafreen Khan 21BCS1397";
    Arrays.sort(g);
    Arrays.sort(s);

    int i = 0;
    for (int j = 0; i < g.length && j < s.length; ++j)
      if (g[i] <= s[j]) ++i;

    return i;
  }
}
```

Output:



```
Accepted    Runtime: 7 ms

Case 1    Case 2

Input
g =
[1,2,3]
s =
[1,1]

Stdout
Aafreen Khan 21BCS1397

Output
1
```

Experiment 3.2

Student Name: Aafreen Khan **UID: 21BCS1397**
Branch: BE-CSE **Section/Group:CC-606-A**
Semester: 6 **Date of Performance:06-04-2024**
Subject Name: Advance Programming lab **Subject Code:21CSP-251**

1. Aim:

- To Solve the binary watch
- To Solve the Sticker to spell word

2. Objective:

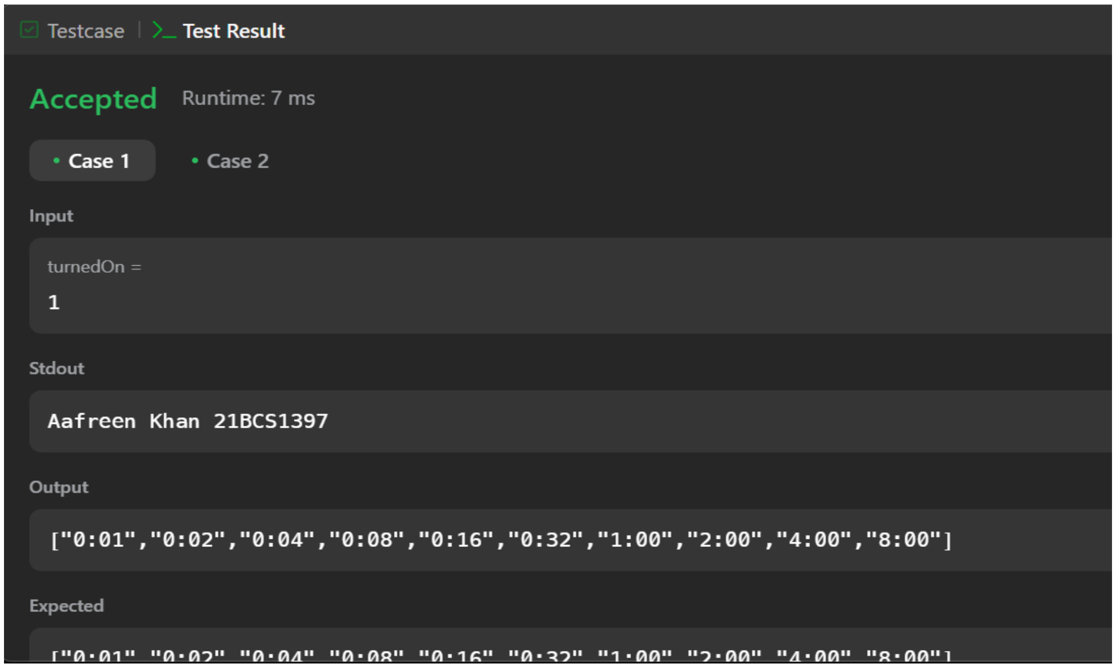
- A binary watch has 4 LEDs on the top to represent the hours (0-11), and 6 LEDs on the bottom to represent the minutes (0-59). Each LED represents a zero or one, with the least significant bit on the right.

- You would like to spell out the given string target by cutting individual letters from your collection of stickers and rearranging them. You can use each sticker more than once if you want, and you have infinite quantities of each sticker. Return the minimum number of stickers that you need to spell out target. If the task is impossible, return -1.

3. Algo. /Approach and output:

I)

```
class Solution {
public:
    vector<string> readBinaryWatch(int turnedOn) {
        vector<string> result;
        for(int h=0;h<12;h++){
            for(int m=0;m<60;m++){
                if((_builtin_popcount(h)+_builtin_popcount(m))==turnedOn){
                    result.push_back(to_string(h)+(m<10 ? ":0" : ":")+to_string(m));
                }
            }
        }
    return result;
    }
};
```



Testcase | Test Result

Accepted Runtime: 7 ms

Case 1 Case 2

Input

```
turnedOn =
1
```

Stdout

```
Aafreen Khan 21BCS1397
```

Output

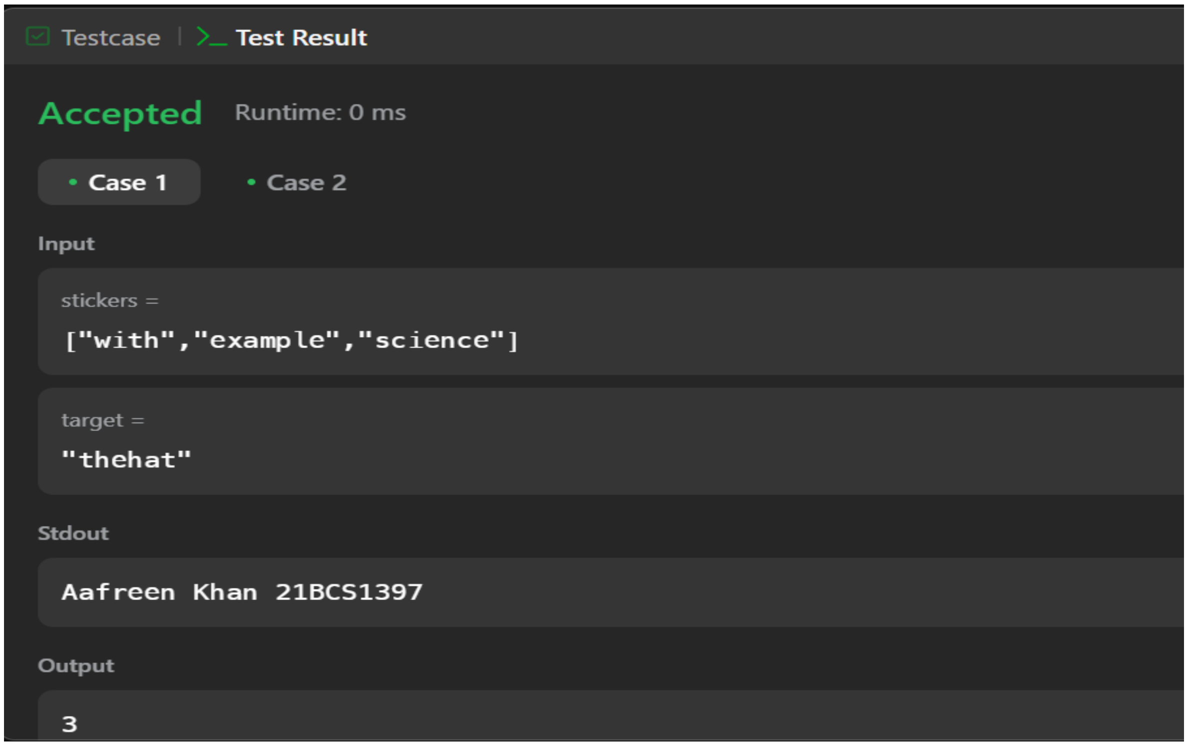
```
["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00"]
```

Expected

```
["0:01","0:02","0:04","0:08","0:16","0:32","1:00","2:00","4:00","8:00"]
```

II)

```
class Solution {
public:
    int minStickers(vector<string>& stickers, string& target) {
        int n = size(stickers);
        unordered_set<string> visited;
        vector<vector<int>> s_frequencies(n, vector<int>(26, 0));
        for (int i = 0; i < n; ++i)
            for (auto& c : stickers[i])
                ++s_frequencies[i][c - 'a'];
        vector<int> t_frequency(26, 0);
        for (auto& c : target)
            ++t_frequency[c - 'a'];
        queue<vector<int>> q;
        q.push(t_frequency);

        for (int res = 0; size(q); ++res) {
            for (int k = size(q); k > 0; --k) {
                auto t_freq = q.front(); q.pop();
                string t_str;
                for (int i = 0; i < 26; ++i)
                    if (t_freq[i] > 0)
                        t_str += string(t_freq[i], i);

                if (t_str == "") return res;

                if (visited.count(t_str)) continue;
                visited.insert(t_str);

                char seeking = t_str[0];
                for (auto& v : s_frequencies) {
                    if (v[seeking] > 0) {
                        q.push(t_freq); // Push first to copy t_freq
                        for (int i = 0; i < 26; ++i)
                            q.back()[i] -= v[i];
                    }
                }
            }
        }

        return -1;
    }
};
```



Accepted Runtime: 0 ms

Input

stickers =

```
["with","example","science"]
```

target =

```
"thehat"
```

Stdout

```
Aafreen Khan 21BCS1397
```

Output

```
3
```

Experiment 3.3

Student Name: Aafreen Khan **UID: 21BCS1397**
Branch: B.E-CSE **Section/Group: 606-A**
Semester: 6th **Date of Performance: 12/04/2024**
Subject Name: Advanced Programming Lab-II
Subject Code: 21CSP-351

1. **Aim:** To Demonstrate the Concept of Dynamic Programming.

2. **Objective:**

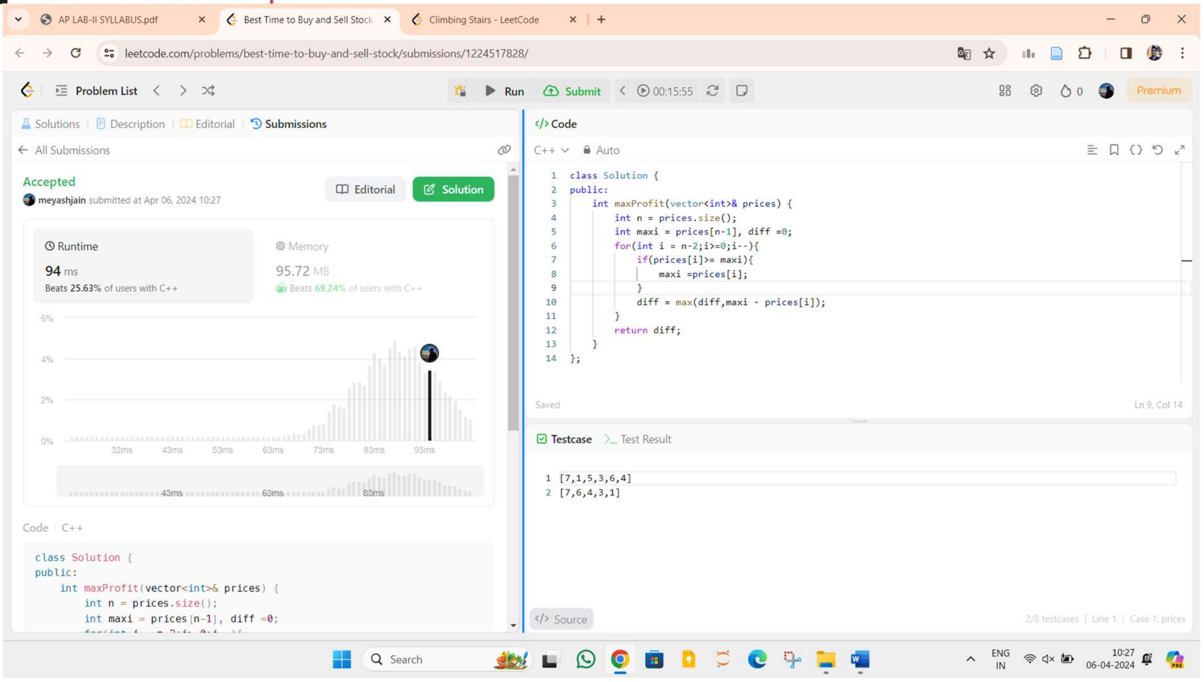
a) You are given an array prices here prices[i] is the price of a given stock on the ith day. You want to maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the maximum profit you can achieve from this transaction. If you cannot achieve any profit, return 0.

b) You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

3. **Code:**

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        int maxi = prices[n-1], diff =0;
        for(int i = n-2;i>=0;i--){
            if(prices[i]>= maxi){
                maxi =prices[i];
            }
            diff = max(diff,maxi - prices[i]);
        }
        return diff;
    }
};
```

OUTPUT:



leetcode.com/problems/best-time-to-buy-and-sell-stock/submissions/1224517828/

Accepted
meyashjain submitted at Apr 06, 2024 10:27

Runtime
94 ms
Beats 25.63% of users with C++

Memory
95.72 MB
Beats 69.24% of users with C++

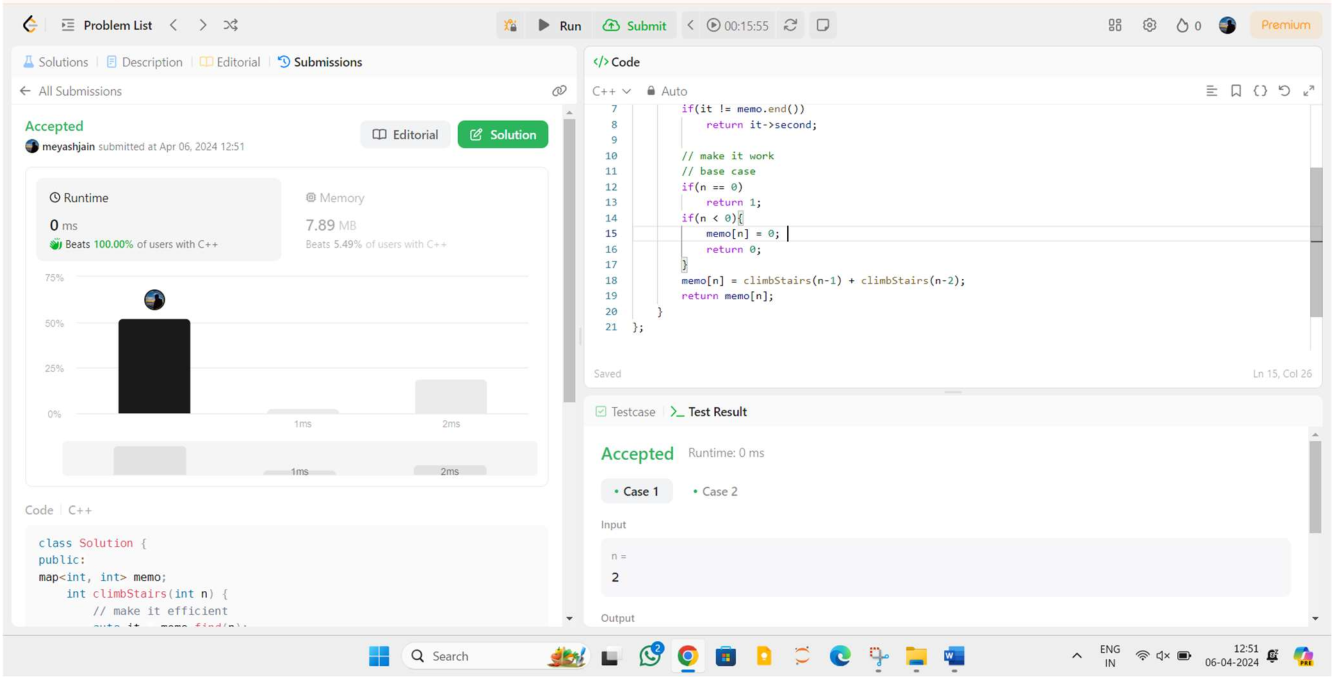
```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int n = prices.size();
        int maxi = prices[n-1], diff =0;
        for(int i = n-2;i>=0;i--){
            if(prices[i]>= maxi){
                maxi =prices[i];
            }
            diff = max(diff,maxi - prices[i]);
        }
        return diff;
    }
};
```

Testcase

```
[7,1,5,3,6,4]
[7,6,4,3,1]
```

```
class Solution {
public:
map<int, int> memo;
    int climbStairs(int n) {
        // make it efficient
        auto it = memo.find(n);
        if(it != memo.end())
            return it->second;

        // make it work
        // base case
        if(n == 0)
            return 1;
        if(n < 0){
            memo[n] = 0;
            return 0;
        }
        memo[n] = climbStairs(n-1) + climbStairs(n-2);
        return memo[n];
    }
};
```

OUTPUT:



Accepted
meyashjain submitted at Apr 06, 2024 12:51

Runtime
0 ms
Beats 100.00% of users with C++

Memory
7.89 MB
Beats 5.49% of users with C++

Code | C++

```
class Solution {
public:
map<int, int> memo;
    int climbStairs(int n) {
        // make it efficient
```

```
        if(it != memo.end())
            return it->second;

        // make it work
        // base case
        if(n == 0)
            return 1;
        if(n < 0){
            memo[n] = 0;
            return 0;
        }
        memo[n] = climbStairs(n-1) + climbStairs(n-2);
        return memo[n];
    }
};
```

Saved
Ln 15, Col 26

Accepted Runtime: 0 ms

Case 1 Case 2

Input

n =
2

Output

4. Learning Outcomes

- Learnt about dynamic programming.
- Learnt about memoization approach to visit all states.
- Learnt about Optimization problems.