Experiment 5

Student Name: Harsh Kumar **UID: 22BCS15183**
Branch: CSE **Section/Group: IOT_640 - B**
Semester: 6 **Date of Performance:24/02/25**
Subject Name: AP Lab-II **Subject Code: 22CSP-351**

1. Aim:

To solve Problems based on Searching and Sorting

2. Objective:

To apply searching and sorting algorithms to solve the problems of LeetCode questions based on that

3. Algorithm:

a) First bad version:

Set left = 1, right = n.

Use binary search:

- Find mid = (left + right) / 2.
- If mid is bad, search left (right = mid).
- Else, search right (left = mid + 1).

Return left (first bad version).
Time Complexity: O(log n).

b) Top K frequent elements:

```
Function topKFrequent(nums, k):
   freqMap = empty hashmap
   for each num in nums:
      freqMap[num] = freqMap[num] + 1
```

```
minHeap = empty priority queue
for each (num, freq) in freqMap:
   push (freq, num) into minHeap
   if size of minHeap > k:
      pop from minHeap

result = empty list
while minHeap is not empty:
   pop element from minHeap and add to result

    return result
```

4. Implementation/Code:

a) First bad version:

```
class Solution {
public:
        int firstBadVersion(int n) {
                int left = 1, right = n;
                while (left < right) {
                        int mid = left + ((right - left) >> 1);
                        if (isBadVersion(mid)) {
                                right = mid; // Reduce search space
                        } else {
                                left = mid + 1;
                        }
                  }
                return left; // First bad version
         }
};
```

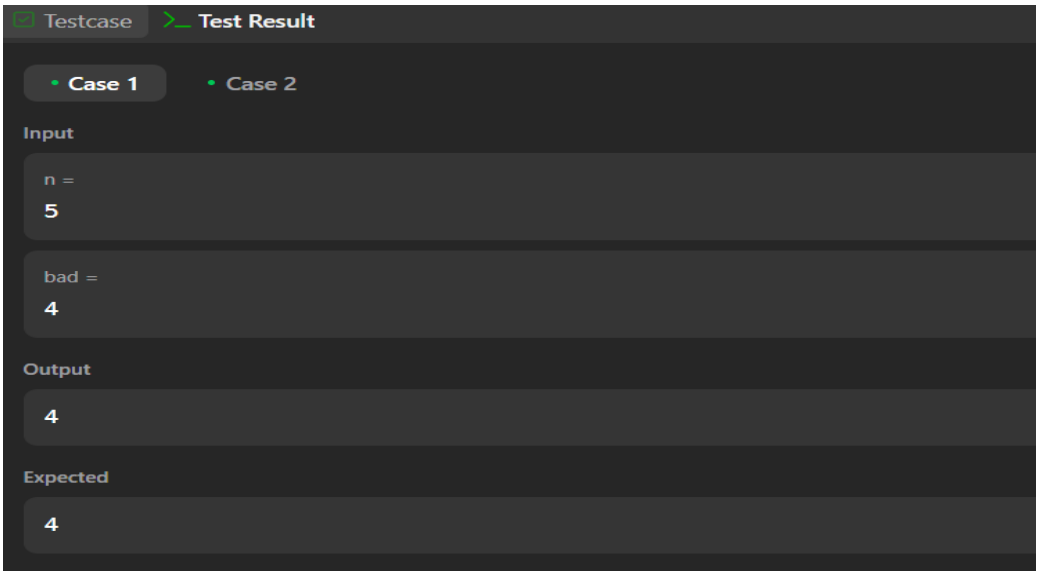
b) Top K frequent elements:

```
#include <vector>

#include <unordered_map>

#include <queue>

using namespace std;

class Solution {

public:

    vector<int> topKFrequent(vector<int>& nums, int k) {

        unordered_map<int, int> freqMap;

        for (int num : nums) {

            freqMap[num]++;

        }

        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
minHeap;

        for (auto& pair : freqMap) {

            minHeap.push({pair.second, pair.first});

            if (minHeap.size() > k) {

                minHeap.pop();

            }

        }

        vector<int> result;

        while (!minHeap.empty()) {

            result.push_back(minHeap.top().second);

            minHeap.pop();
```

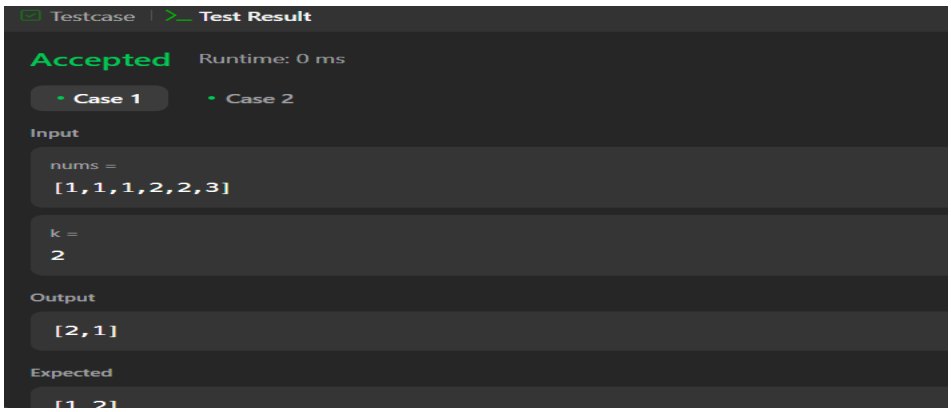
```
                    }

                    return result;

                }

            };
```

5. Output:

a) First bad version:



b) Top k frequent element:



6. Learning Outcome:

- **Efficiency** – Understanding how different algorithms optimize data retrieval and organization.
- **Complexity Analysis** – Learning to evaluate time and space complexity for better algorithm selection.
- **Real-World Applications** – Applying sorting and searching in databases, networking, and AI systems.
- **Problem-Solving Skills** – Enhancing logical thinking through algorithmic challenges and optimizations.