Experiment 5.1

Student Name: Tushal Anand **UID: 22BCS13570**

Branch: BE-CSE **Section/Group: IOT-641/A**

Semester:6th **Date of Performance: 25/02/2025**

Subject Name: AP II **Subject Code: 22CSH-351**

1. **Aim:**Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks.

   Objective: To implement an algorithm that finds the index of a peak element in a given 0-indexed integer array nums.

2. **Code:**

```
    class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int l=0, r=nums.size()-1;
        while(l<r){
            int mid=l+(r-l)/2;
            if(nums[mid]<nums[mid+1]){
                l=mid+1;
            }
            else{
                r=mid;
            }
        }
        return l;
    }
};
```

3. **Output:**

4. **Learning Outcome**s:

(i) Binary Search in unsorted arrays.
(ii) How to find local maxima in arrays.
(iii) Writing optimized code for logarithmic complexity problems.

Experiment 5.2

Aim: You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Objective: To implement a function that identifies the first bad version from a sequence of n product versions using the minimum number of API calls.

Code:

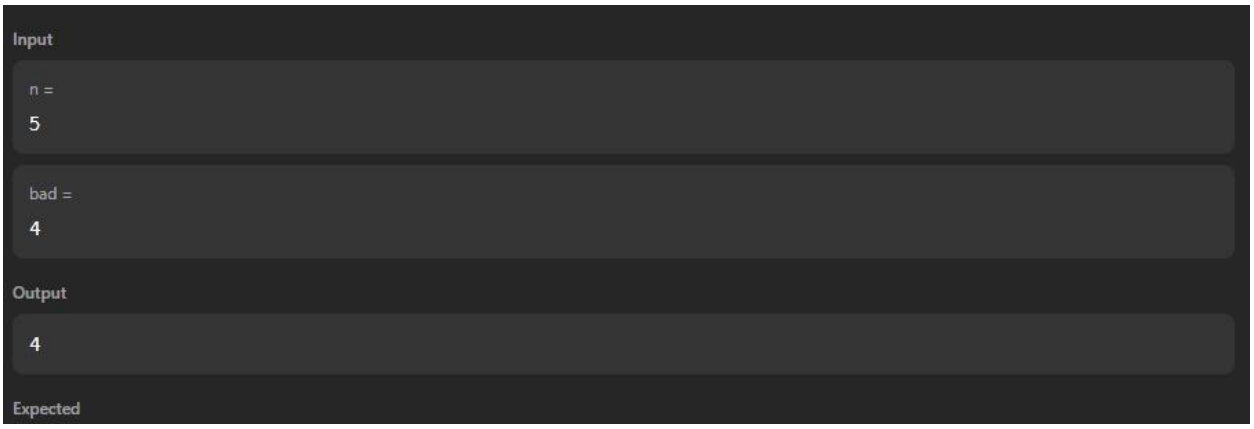
```
// The API isBadVersion is defined for you.
// bool isBadVersion(int version);

class Solution {
public:
    int firstBadVersion(int n) {
        int first = 1;
        int last = n;
```

```
        while (first < last) {
            int mid = first + (last - first) / 2;
```

```
            if (isBadVersion(mid)) {
                last = mid;
            } else {
                first = mid + 1;
            }
        }
```

```
        return first;
    }
};
```

Output:



Learning Outcomes:
(i) Efficient Binary Search implementation.
(ii) Reducing API calls using logarithmic search.
(iii) Writing clean and optimized code.
(iv) Handling corner cases effectively.

Experiment 5.3

Aim: Given an array nums with n objects colored red, white, or blue, sort them in-place so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Objective: To implement an in-place sorting algorithm that sorts an array nums[] consisting of integers 0, 1, and 2 representing three colors:

0 → Red
1 → White
2 → Blue

Code:

```
class Solution {
public:
    void sortColors(vector<int>& nums) {
        int low = 0, mid = 0, high = nums.size()-1;
        while(mid <= high){
            if(nums[mid] == 0){
                swap(nums[low], nums[mid]);
                low++;
                mid++;
            }
            else if(nums[mid] == 1){
                mid++;
            }
            else{
                swap(nums[mid], nums[high]);
                high--;
            }
        }
    }
};
```

Output:

Learning Outcomes:

(i) Mastering the Dutch National Flag Algorithm.
(ii) Writing in-place sorting without extra space.
(iii) How to efficiently sort arrays with only three unique values.
(iv) Optimizing time complexity from O(n log n) to O(n).