Experiment 5

Student Name: Deepak Agrawal **UID:22BCS12192**
Branch: CSE **Section/Group: NTPP-IOT603/B**
Semester: 6th **Date of Performance:20-3-25**
Subject Name: AP Lab-2 **Subject Code:** 22CSP-351

Problem -1

1. Aim: Merge Sorted Array

2. Objective:

- **Understand Merging of Sorted Arrays**: - The goal is to combine two sorted arrays into one sorted array. This helps in learning how to correctly place elements while maintaining order.
- **Efficient In-Place Merging**:-The merging should be done within nums1 without using extra space. This improves efficiency and helps in solving problems that require modifying arrays directly.
- **Using Two-Pointer Technique**: - The two-pointer approach helps merge arrays efficiently by placing larger elements first. This reduces unnecessary shifts and improves the merging process.
- **Handling Edge Cases**: - It is important to consider cases like an empty nums2 or extra zeros in nums1. This ensures that the algorithm works for all possible inputs.
- **Improving Problem-Solving Skills**:- Solving this problem enhances logical thinking and coding skills. It also prepares you for technical interviews that require optimizing solutions.

3. Implementation/Code:

```
class Solution {  public:
   void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
int i = m - 1, j = n - 1, k = m + n - 1;
      while (i >= 0 && j >= 0) {
if (nums1[i] > nums2[j]) {
            nums1[k]    =    nums1[i];
i--;
         } else {
            nums1[k] = nums2[j];
            j--
;          }
k--;
      }
      while (j >= 0) {
         nums1[k] = nums2[j];
```

```
            j--;
k--;
        }
    } };
```

4. Output:

Figure 1

5. Learning Outcome:

- **Ability to Merge Sorted Arrays**: - You will learn to merge two sorted arrays efficiently while maintaining their order in a single array. This helps in understanding how to correctly insert elements in the given space without using extra memory.
- **Understanding of Two-Pointer Approach**: - The two-pointer method allows efficient merging without extra space. This improves problem-solving skills and helps in solving other array-related problems.
- **Handling Edge Cases Confidently**: - You will understand how to manage cases like an empty nums2 or trailing zeros in nums1. This ensures that your solution is reliable and works in all scenarios.
- **Writing Optimized Code**: - Learning this method helps in writing optimized code with minimal time complexity. This makes your solutions more efficient and improves performance.
- **Problem-Solving for Interviews**: - This problem is commonly asked in coding interviews. Practicing it will improve your logical thinking and help you solve similar array-based problems quickly.

Problem-2

1. Aim: Sort Colors

2. Objectives:

- **Sorting Colors Without Sorting Function**: - The goal is to sort an array containing 0s, 1s, and 2s without using built-in sorting. This helps in learning efficient ways to organize data manually.
- **Using the Dutch National Flag Algorithm**: - The algorithm helps in sorting the array in a single pass. This improves understanding of how to arrange elements using multiple pointers.
- **Efficient In-Place Sorting**: - The sorting is done without extra space, modifying the array directly. This teaches how to optimize memory usage in coding problems.
- **Handling Different Cases Easily**: - The method ensures that all numbers are placed in the correct order. It helps in dealing with cases where numbers are shuffled randomly.
- **Improving Logical Thinking and Speed**: - Understanding this approach improves coding skills and speed. It is useful for solving interview questions and competitive programming problems.

3. Implementation/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--;
          }
       }
   }
};
```

4. Output:

Figure 2

5. Learning Outcomes:

- **Sorting Arrays Without Extra Space**:-You will learn how to sort an array without using extra memory. This helps in understanding space-efficient solutions.
- **Mastering the Two-Pointer Approach**: - The two-pointer method helps in arranging elements quickly. It makes solving similar sorting problems easier.
- **Handling Complex Sorting Problems**: - You will gain confidence in solving sorting problems efficiently. This improves your problem-solving ability in technical interviews.
- **Writing Optimized Code**: - The approach ensures sorting is done in one pass. This makes the code faster and reduces unnecessary computations.
- **Better Preparation for Interviews**: - This problem is commonly asked in coding interviews. Practicing it will strengthen your ability to solve sorting-based challenges.

Problem – 3

1. **Aim:** Find Peak Element
2. **Objectives:**

- **Understanding Peak Elements**: - The objective is to identify a peak element in an array where each element is compared with its neighbors. This helps in recognizing patterns within arrays and solving related problems.
- **Implementing Binary Search**: - The goal is to apply the binary search approach to find a peak element efficiently. This ensures that the solution runs in O(log n) time instead of a linear scan.

- **Optimizing Problem-Solving Skills**: - This problem helps in improving logical reasoning by teaching how to make decisions based on comparisons. Understanding how to narrow down the search space efficiently is a key takeaway.
- **Handling Special Cases**: - The objective is to handle various scenarios, such as peaks appearing at the start, middle, or end of the array. This ensures a complete and robust solution.
- **Enhancing Algorithmic Thinking**: - By working on this problem, you will strengthen your ability to design and implement optimized algorithms. This contributes to developing better coding and debugging skills.

3. Implementation/Code:

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

4. Output:

Figure 3

5. Learning Outcomes

- **Efficient Peak Finding**: - You will learn how to locate a peak element without scanning the entire array, using a smarter approach with binary search.
- **Mastering Binary Search Variations**: - You will understand how binary search can be adapted for different problems beyond simple number searching.
- **Developing a Logical Approach**: - You will improve your ability to break down problems logically, making it easier to apply efficient solutions in coding interviews and real-world tasks.
- **Understanding Search Space Reduction**: - You will gain insights into how reducing the search space step by step can lead to significant performance improvements.
- **Building Optimized and Scalable Solutions**:- You will develop the skills to write code that is both time-efficient and scalable, a crucial requirement for competitive programming and software development.