Experiment Title: Implementation of Programs on Divide and Conquer Problems.

Aim/Objective: To understand the concept and implementation of Basic programs on Divide and Conquer Problems.

Description: The students will understand and able to implement programs on Divide and Conquer Problems.

Pre-Requisites:

Knowledge: Arrays, Sorting, Divide and Conquer in C/C++/Python Tools: Code Blocks/Eclipse IDE

Pre-Lab:

Write a program to sort an array using the Merge Sort algorithm.

Input: An unsorted array of integers.

Output: The sorted array.

Example: Input: [5, 2, 4, 6, 1, 3]

Output: [1, 2, 3, 4, 5, 6]

- **Procedure/Program:**

```
#include <stdio.h>
void merge(int arr[], int left, int mid, int right) {
  int n1 = mid - left + 1;
  int n2 = right - mid;

  int leftArr[n1], rightArr[n2];

  for (int i = 0; i < n1; i++)
    leftArr[i] = arr[left + i];
  for (int j = 0; j < n2; j++)
    rightArr[j] = arr[mid + 1 + j];
```

```
 int i = 0, j = 0, k = left;
  while (i < n1 && j < n2) {
    if (leftArr[i] <= rightArr[j]) {
      arr[k] = leftArr[i];
      i++;
    } else {
      arr[k] = rightArr[j];
      j++;
    }
    k++;
  }

  while (i < n1) {
    arr[k] = leftArr[i];
    i++;
    k++;
  }

  while (j < n2) {
    arr[k] = rightArr[j];
    j++;
    k++;
  }
}
```

```
void mergeSort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        merge(arr, left, mid, right);
    }
}

int main() {
    int arr[] = {5, 2, 4, 6, 1, 3};
    int size = sizeof(arr) / sizeof(arr[0]);

    mergeSort(arr, 0, size - 1);

    printf("Sorted array: ");
    for (int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

- **Data and Results**:

Data

Input array: {5, 2, 4, 6, 1, 3}

Array size: 6

Result

Sorted array: 1 2 3 4 5 6

- **Analysis and Inferences**:

Analysis

Merge Sort uses divide and conquer for efficient sorting.

Inferences

Merge Sort has O(n log n) time complexity efficiency.

In-Lab:

Find the majority element in an array (an element that appears more than n/2 times) using Divide and Conquer.

Input: An array of integers.

Output: The majority element, or -1 if none exists.

Example:

Input: [3, 3, 4, 2, 4, 4, 2, 4, 4]

Output: 4

- **Procedure/Program:**

```
#include <stdio.h>

int findMajorityElement(int arr[], int left, int right) {
    if (left == right)
        return arr[left];

    int mid = left + (right - left) / 2;

    int leftMajority = findMajorityElement(arr, left, mid);
    int rightMajority = findMajorityElement(arr, mid + 1, right);

    if (leftMajority == rightMajority)
        return leftMajority;

    int leftCount = 0, rightCount = 0;

    for (int i = left; i <= right; i++) {
        if (arr[i] == leftMajority)
            leftCount++;

        if (arr[i] == rightMajority)
            rightCount++;
    }
```

```
  if (leftCount > (right - left + 1) / 2)
      return leftMajority;

   if (rightCount > (right - left + 1) / 2)
      return rightMajority;

   return -1;
}

int main() {
   int arr[] = {3, 3, 4, 2, 4, 4, 2, 4, 4};
   int n = sizeof(arr) / sizeof(arr[0]);

   int result = findMajorityElement(arr, 0, n - 1);

   printf("Output: %d\n", result);

   return 0;
}
```

- **Data and Results:**

Data

Input array: {3, 3, 4, 2, 4, 4, 2, 4, 4}

Result

Majority element: 4

- **Analysis and Inferences:**

Analysis

Divide and conquer identifies majority in O(n log n).

Inferences

Majority exists if count exceeds n/2 in array.

Post-Lab:

Find a peak element in an array using a divide-and-conquer approach. A peak element is one that is greater than or equal to its neighbors.

Input: An array of integers.

Output: A peak element.

Example:

Input: [1, 3, 20, 4, 1, 0]

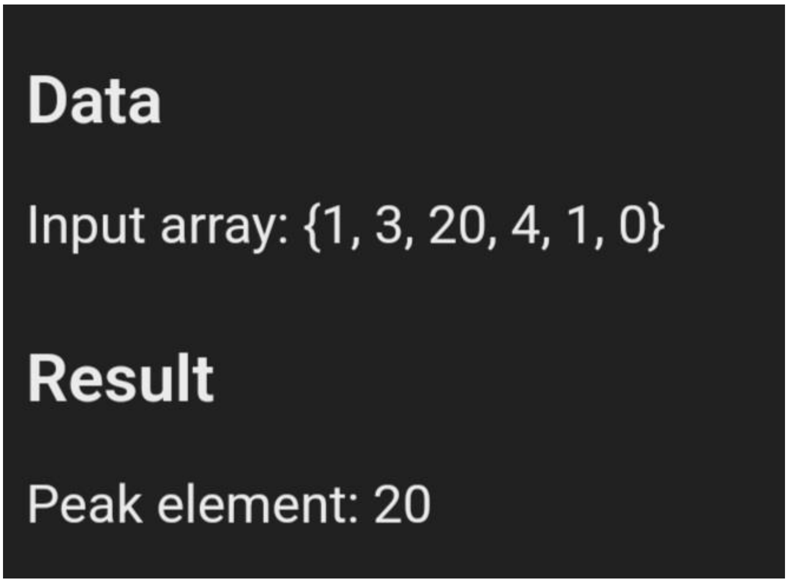
Output: 20

- **Procedure/Program:**

```
#include <stdio.h>


int main() {
   int arr[] = {1, 3, 20, 4, 1, 0};
   int n = sizeof(arr) / sizeof(arr[0]);


   int findPeak(int arr[], int left, int right) {
      int mid = left + (right - left) / 2;


      if ((mid == 0 || arr[mid - 1] <= arr[mid]) && (mid == n - 1 || arr[mid + 1] <=
arr[mid])) {
         return arr[mid];
      }
      else if (mid > 0 && arr[mid - 1] > arr[mid]) {
         return findPeak(arr, left, mid - 1);
      }
      else {
         return findPeak(arr, mid + 1, right);
```

```
    }
  }

  int peak = findPeak(arr, 0, n - 1);
  printf("Output: %d\n", peak);

  return 0;
}
```

- **Data and Results:**



- **Analysis and Inferences:**

Analysis

Divide and conquer finds peak in logarithmic time complexity.

Inferences

Peak exists if it is greater than neighbors.

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What are some common applications of divide and conquer algorithms?

- Sorting (Merge Sort, Quick Sort), Searching (Binary Search), and Matrix Multiplication (Strassen's Algorithm).

2. What are the limitations or challenges of using the divide and conquer approach

- Overhead in recursive calls, difficulty in parallelization, and increased memory usage for temporary storage.

3. Define the merge sort algorithm and its time complexity.

- Merge Sort splits the array, sorts halves, and merges them. Time complexity: O(n log n).

4. What are some alternative problem-solving approaches to divide and conquer?

- Greedy algorithms, dynamic programming, brute force, and backtracking.

5. How can parallelization be applied to Divide and Conquer algorithms to improve performance?

- Independent subproblems can be executed concurrently on multiple processors.

Evaluator Remark (if Any):	**Marks Secured___ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.