Assignment - 2 :

a) WAP to implement QUICK sort using Divide and Conquer Strategy.

b) WAP to implement MERGE sort using Divide and Conquer Strategy.

(a) Quick Sort –
Algorithm –

Quick Sort Algorithm –

Procedure: quickSort(A, low, high)
Input: A – An array containing elements to be sorted.
low – Lower bound of array A.
high – Upper bound of array A.
Output: A – Sorted Array

1. if low < high then
2. partitionIndex ← partition(A, low, high).
3. call quickSort(A , low , partitionIndex-1).
4. call quickSort(A , partitionIndex + 1, high).
5. end if.
6. return.

Partition Algorithm –

Procedure: partition(A, low, high)
Input: A – An array A that is to be partitioned into two subarrays.
low – Lower bound of array A.
high – Upper bound of array A.
Output: j – New location of pivot element of array A.

1. i ← low+1.
2. j ← high.
3. pivot ← A[low].
4. while i < j do
5. while A[i] ≤ pivot do
6. i++.
7. end while.
8. while A[j] > pivot do
9. j--.
10. end while.
11. if i < j

```
12.             temp = A[i].
13.             A[i] = A[j].
14.             A[j] = temp.
15.     end if.
16. end while.
17. temp = A[low].
18. A[low] = A[j].
19. A[j] = temp.
20. return j.
```

Program –

```
#include <stdio.h>

void printArray(int A[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", A[i]);
    }
    printf("\n");
}

int partition(int A[], int low, int high)
{
    int i = low + 1, j = high, pivot = A[low], temp;

    while (i < j)
    {
        while (A[i] <= pivot)
        {
            i++;
        }
        while (A[j] > pivot)
        {
            j--;
        }
        if (i < j)
        {
            temp = A[i];
            A[i] = A[j];
            A[j] = temp;
        }
    }

    // Swap A[low] and A[j]
    temp = A[low];
```

```
    A[low] = A[j];
    A[j] = temp;

    return j;
}

void quickSort(int A[], int low, int high)
{
    int partitionIndex; // index of pivot after partition

    if (low < high)
    {
        partitionIndex = partition(A, low, high);
        quickSort(A, low, partitionIndex - 1);  // sort
left subarray
        quickSort(A, partitionIndex + 1, high); // sort
right subarray
    }
    return;
}

int main()
{
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int A[n];
    for (int i = 0; i < n; i++)
    {
        printf("Enter array element (%d): ", i + 1);
        scanf("%d", &A[i]);
    }

    printf("Unsorted array: ");
    printArray(A, n);
    quickSort(A, 0, n - 1);
    printf("Sorted array: ");
    printArray(A, n);

    return 0;
}
```

Output –

```
Falguni Sarkar@MELOPHILE  G:\Semester~4\Design & Analysis of Algorithm\La
b\Assignment 2
  [13:23]
❯ cd "g:\Semester~4\Design & Analysis of Algorithm\Lab\Assignment 2\" ; if
 ($?) { gcc Quick_Sort.c -o Quick_Sort } ; if ($?) { .\Quick_Sort }
Enter the size of the array: 5
Enter array element (1): 9
Enter array element (2): 7
Enter array element (3): 5
Enter array element (4): 3
Enter array element (5): 1
Unsorted array: 9 7 5 3 1
Sorted array: 1 5 3 7 9
Falguni Sarkar@MELOPHILE  G:\Semester~4\Design & Analysis of Algorithm\La
b\Assignment 2
  [13:24]
❯ cd "g:\Semester~4\Design & Analysis of Algorithm\Lab\Assignment 2\" ; if
 ($?) { gcc Quick_Sort.c -o Quick_Sort } ; if ($?) { .\Quick_Sort }
Enter the size of the array: 7
Enter array element (1): 38
Enter array element (2): 27
Enter array element (3): 43
Enter array element (4): 3
Enter array element (5): 9
Enter array element (6): 82
Enter array element (7): 10
Unsorted array: 38 27 43 3 9 82 10
Sorted array: 3 9 27 10 38 43 82
```

(b) Merge Sort –

Algorithm –

Merge Sort Algorithm –

Procedure: mergeSort(A, low, high)
Input: A – An array containing elements to be sorted.
low – Lower bound of array A.
high – Upper bound of array A.
Output: A – Sorted Array

1. if low < high then
2. mid ← ⌊ (low+high) / 2 ⌋
3. call mergeSort(A , low , mid).
4. call mergeSort(A , mid + 1, high).
5. call merge (A, mid, low, high).
6. end if.
7. return.

Merge Algorithm –

Procedure: merge(A, mid, low, high)
Input: A – An array containing sorted left and right sub-arrays.
mid – Upper bound of sorted left sub-array A.
low – Lower bound of sorted left sub-array A.
high – Upper bound of sorted right sub-array A.
Output: A – Sorted Array

1. i ← low.
2. j ← mid + 1.
3. k ← low.
4. while i ≤ mid and j ≤ high do
5. if A[i] < A[j] then
6. B[k] = A[i].
7. i++.
8. else
9. B[k] = A[j].
10. j++.
11. end if.
12. k++.
13. end while.
14. while i ≤ mid do
15. B[k] = A[i]
16. k++.
17. i++.
18. end while.
19. while j ≤ high do
20. B[k] = A[j]
21. k++.
22. j++.
23. end while.
24. for i ← low to high do
25. A[i] = B[i].
26. end for.
27. return.

Program –

```
#include <stdio.h>

void printArray(int A[], int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("%d ", A[i]);
```

```
    }
    printf("\n");
}

void merge(int A[], int mid, int low, int high)
{
    int i, j, k, B[high+1];
    i = low;
    j = mid + 1;
    k = low;

    while (i <= mid && j <= high)
    {
        if (A[i] < A[j])
        {
            B[k] = A[i];
            i++;
        }
        else
        {
            B[k] = A[j];
            j++;
        }
        k++;
    }
    while (i <= mid)
    {
        B[k] = A[i];
        k++;
        i++;
    }
    while (j <= high)
    {
        B[k] = A[j];
        k++;
        j++;
    }
    for (int i = low; i <= high; i++)
    {
        A[i] = B[i];
    }
    return;
}

void mergeSort(int A[], int low, int high)
{
```

```
    int mid;
    if (low < high)
    {
        mid = (low + high) / 2;
        mergeSort(A, low, mid);
        mergeSort(A, mid + 1, high);
        merge(A, mid, low, high);
    }
    return;
}

int main()
{
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);
    int A[n];
    for (int i = 0; i < n; i++)
    {
        printf("Enter array element (%d): ", i + 1);
        scanf("%d", &A[i]);
    }

    printf("Unsorted array: ");
    printArray(A, n);
    mergeSort(A, 0, n - 1);
    printf("Sorted array: ");
    printArray(A, n);

    return 0;
}
```

Output –

```
Falguni Sarkar@MELOPHILE  G:\Semester~4\Design & Analysis of Algorithm\Lab\Assignment 2                [13:17]
❯ cd "g:\Semester~4\Design & Analysis of Algorithm\Lab\Assignment 2\" ; if ($?) { gcc Merge_Sort.c -o Merge_Sort } ; if ($?) { .\Merge_Sort }
Enter the size of the array: 7
Enter array element (1): 38
Enter array element (2): 27
Enter array element (3): 43
Enter array element (4): 3
Enter array element (5): 9
Enter array element (6): 82
Enter array element (7): 10
Unsorted array: 38 27 43 3 9 82 10
Sorted array: 3 9 10 27 38 43 82
Falguni Sarkar@MELOPHILE  G:\Semester~4\Design & Analysis of Algorithm\Lab\Assignment 2                [13:17]
❯ cd "g:\Semester~4\Design & Analysis of Algorithm\Lab\Assignment 2\" ; if ($?) { gcc Merge_Sort.c -o Merge_Sort } ; if ($?) { .\Merge_Sort }
Enter the size of the array: 7
Enter array element (1): 12
Enter array element (2): 11
Enter array element (3): 13
Enter array element (4): 5
Enter array element (5): 6
Enter array element (6): 7
Enter array element (7): 4
Unsorted array: 12 11 13 5 6 7 4
Sorted array: 4 5 6 7 11 12 13
```