PRACTICAL 3:

AIM: Implementation and Time analysis of Merge Sort algorithms for Best case, Average case & Worst-case using Divide and Conquer.

Algorithm:

1. MERGE_SORT(arr, beg, end)
2. **if** beg < end
3. set mid = (beg + end)/2
4. MERGE_SORT(arr, beg, mid)
5. MERGE_SORT(arr, mid + 1, end)
6. MERGE (arr, beg, mid, end)
7. end of **if**
8. END MERGE_SORT

Code:

```
#include <stdio.h>
#include <stdlib.h>
void merge(int arr[], int l, int m, int r)
{
        int i, j, k;
        int n1 = m - l + 1;
        int n2 = r - m;
        int L[n1], R[n2];
        for (i = 0; i < n1; i++)
                L[i] = arr[l + i];
        for (j = 0; j < n2; j++)
                R[j] = arr[m + 1 + j];
        i = 0;
        j = 0;
        k = l;
        while (i < n1 && j < n2) {
                if (L[i] <= R[j])
                {       arr[k] = L[i];
                        i++;            }
                else {  arr[k] = R[j];
                        j++;            }
                k++;    }
        while (i < n1) {
                arr[k] = L[i];
                i++;
```

```
        k++;    }
while (j < n2)
{       arr[k] = R[j];
        j++;
        k++;    }
}
void mergeSort(int arr[], int l, int r)
{
        if (l < r)
        {
                int m = l + (r - l) / 2;
                mergeSort(arr, l, m);
                mergeSort(arr, m + 1, r);
                merge(arr, l, m, r);
        }   }
int main()
{       int u,i=0;
        printf("Enter Element size: ");
        scanf("%d",&u);
        int A[u];
        for(i=0;i<u;i++)
         {       printf("Enter Element: ");
                 scanf("%d",&A[i]);              }
        mergeSort(A, 0, u-1);
        printf("\nSorted array is \n");
   i=0;
         for(i=0;i<u;i++)
        {       printf("%d\t",A[i]);          }
   return 0;
}
```

Output:

Best Case Complexity: O(n*logn)

```
PS C:\Users\raj> cd "f:\pu_it_pratical\DAA\" ; if ($?) { gc
Enter Element size: 4
Enter Element: 1
Enter Element: 2
Enter Element: 3
Enter Element: 4

Sorted array is
1       2       3       4
PS F:\pu_it_pratical\DAA>
```

Average Case Complexity: O(n*logn):

```
PS C:\Users\raj> cd "f:\pu_it_pratical
Enter Element size: 4
Enter Element: 1
Enter Element: 2
Enter Element: 4
Enter Element: 3

Sorted array is
1       2       3       4
PS F:\pu_it_pratical\DAA>
```

Worst Case Complexity: O(n*logn):

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pr
Enter Element size: 5
Enter Element: 9
Enter Element: 6
Enter Element: 5
Enter Element: 4
Enter Element: 2

Sorted array is
2       4       5       6       9
PS F:\pu_it_pratical\DAA>
```

PRACTICAL 4:

AIM :Implementation and Time analysis of Quick-Sort algorithms for Best case, Average case & Worst-case using Divide and Conquer.

Algorithm:

Pivot value is Front in quick sort

```
1.  QUICKSORT (array A, start, end)
2.  {
3.   if (start < end)
4.   {
5.   p = partition(A, start, end)
6.   QUICKSORT (A, start, p - 1)
7.   QUICKSORT (A, p + 1, end)
8.   }
9.  }
10. PARTITION (array A, start, end)
11. {
12.  pivot ? A[end]
13.  i ? start-1
14.  for j ? start to end -1 {
15.  do if (A[j] < pivot) {
16.  then i = i + 1
17.  swap A[i] with A[j]
18.   }}
19.  swap A[i+1] with A[end]
20.  return i+1
21. }
```

Code(Pivot Value from Start):

```
#include<stdio.h>
void quicksort(int num[],int front,int l)
{  int i,j,pivot,temp;
   if(front<l)
   {  pivot=front;
      i=front;
      j=l;
      while(i<j)
      {
         while(num[i]<=num[pivot]&&i<l)
         i++;
         while(num[j]>num[pivot])
         j--;
         if(i<j)
```

```
        {  temp=num[i];
           num[i]=num[j];
           num[j]=temp;
        }       }
      temp=num[pivot];
      num[pivot]=num[j];
      num[j]=temp;
      quicksort(num,front,j-1);
      quicksort(num,j+1,l);
   }
}
int main()
{
   int i,count;
   printf("Enter element Size:");
   scanf("%d",&count);
   int num[count];
   printf("Enter %d elements:",count);
   for (i=0;i<count;i++)
   { scanf("%d",&num[i]); }
   quicksort(num,0,count-1);
   printf("Sorted elements:");
   for(i=0;i<count;i++)
   { printf("%d\t",num[i]); }
   return 0;

}
```

Code(Pivot Value from end):

```
#include <stdio.h>
void swap(int *a, int *b)
{    int temp = *a;
   *a = *b;
   *b = temp;    }
int partition(int a[], int start, int end)
{   int pivot = a[end];
   int i = (start - 1);
   for (int j = start; j <= end - 1; j++)
   {      if (a[j] < pivot)
          {             i++;
                        int t = a[i];
                        a[i] = a[j];
                        a[j] = t;          }      }
   int t = a[i+1];
```

```
    a[i+1] = a[end];
    a[end] = t;
    return (i + 1);
}
void quicksort(int a[], int start, int end)
{      if (start < end)
       {              int p = partition(a, start, end);
                      quicksort(a, start, p - 1);
                      quicksort(a, p + 1, end);            }     }
int main()
{   int u,i=0;
    printf("Enter Element size: ");
    scanf("%d",&u);
    int A[u];
    for(i=0;i<u;i++)
    {        printf("Enter Element: ");
            scanf("%d",&A[i]);              }
     quicksort(A, 0, u - 1);
    printf("\nSorted array:\n");
    i=0;
     for(i=0;i<u;i++)
    {        printf("%d\t",A[i]);        }
     return 0;         }
```

Output:

Front:

Best Case Complexity: O(n*logn)

```
Enter element Size:9
Enter 9 elements:13
19
17
15
12
16
18
4
11
Sorted elements:4       11      12      13      15      16      17      18      19
PS F:\pu_it_pratical\DAA>
```

Average Case Complexity: O(n*logn)

```
PS F:\pu_it_pratical\DAA> cd "f:\pu_it_pratical\DAA\" ; if
Enter element Size:4
Enter 4 elements:1
3
2
5
Sorted elements:1       2       3       5
```

Worst Case Complexity: O(n^2):

```
Enter Element size: 4
Enter Element: 1
Enter Element: 2
Enter Element: 5
Enter Element: 4

Sorted array:
1       2       4       5
PS F:\pu_it_pratical\DAA>
```

End:

Best Case Complexity: O(n*logn)

```
Enter Element size: 9
Enter Element: 19
Enter Element: 17
Enter Element: 15
Enter Element: 12
Enter Element: 16
Enter Element: 18
Enter Element: 4
Enter Element: 11
Enter Element: 13

Sorted array:
4       11      12      13      15      16      17      18      19
```

Average Case Complexity: O(n*logn)

```
Enter Element size: 4
Enter Element: 1
Enter Element: 2
Enter Element: 5
Enter Element: 4

Sorted array:
1       2       4       5
```

Worst Case Complexity: O(n^2):

```
Enter Element size: 5
Enter Element: 9
Enter Element: 7
Enter Element: 6
Enter Element: 5
Enter Element: 1

Sorted array:
1       5       6       7       9
```