Subject: DAOA **Class/ Sem: T. Y B. Tech/ Sem-V** **A.Y: 2025-26 (Odd)**

EXPERIMENT NO. 1

MERGE SORT

Name : Soham Kishor Walam **Roll No : D102**

AIM: Write a program to implement merge sort and analyze its time complexity.

THEORY:

Merge sort algorithms are based on a divide and conquer strategy. First, the sequence to be sorted is decomposed into two halves (Divide). Each half is sorted independently (Conquer). Then the two sorted halves are merged to a sorted sequence (Combine). Its worst-case running time has a lower order of growth than insertion sort. Since we are dealing with subproblems, we state each subproblem as sorting a subarray $A[p .. r]$. Initially, $p = 1$ and $r = n$, but these values change as we recurse through sub problems.

To sort $A[p .. r]$:

1. Divide Step

If a given array A has zero or one element, simply return; it is already sorted. Otherwise, split $A[p .. r]$ into two sub arrays $A[p.. q]$ and $A[q + 1 .. r]$, each containing about half of the elements of $A[p .. r]$. That is, q is the halfway point of $A[p .. r]$.

2. Conquer Step

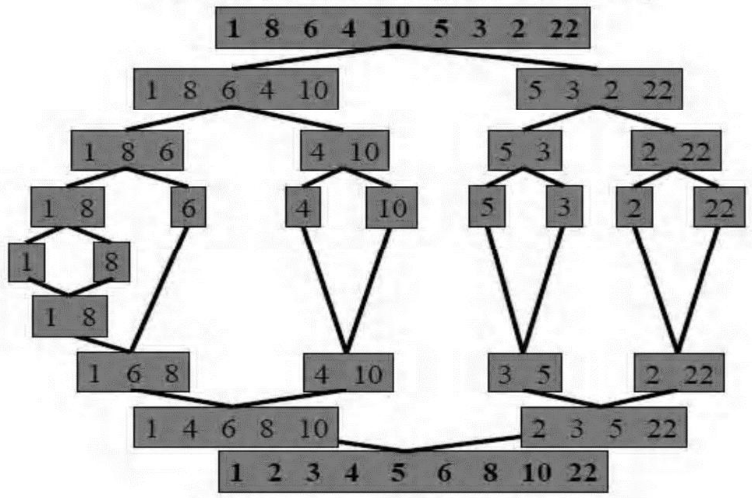
Conquer by recursively sorting the two sub arrays $A[p .. q]$ and $A[q + 1 .. r]$.

3. Combine Step

Combine the elements back in $A[p .. r]$ by merging the two sorted sub arrays $A[p .. q]$ and $A[q + 1 .. r]$ into a sorted sequence. To accomplish this step, we will define a procedure MERGE (A, p, q, r).

Example:

The array to be sorted : [1 8 6 4 10 5 3 2 22]



ALGORITHM:

MergeSort(int A[0....n-1,low,high) :

```
if (low<high) then
    {
        mid ← (low+high)/2
        MergeSort (A,low,mid)
        MergeSort (A,mid+1, high)
        Combine(A,low,mid,high)
    }
```

Combine(int A[0....n-1,low,high) :

```
1. k ← low
2. i ← low
3. j ← mid+1
4. while(i<= mid && j<=high) do
    {
        if (A[i]<=A[j]) then
            {
                temp[k] ← A[i]
                i←i+1
                k←k+1
```

```
            }
            else
              {
                temp[k] ← A[j]
                j←j+1
                k←k+1
              }
          }
5.  while (i<=mid) do
      {
         temp[k] ← A[i]
         i←i+1
         k←k+1
      }
6.  while (j<=high) do
      {
         temp[k] ← A[j]
         j←j+1
         k←k+1
      }
```

CODE:

```c
#include <stdio.h>

void mergeSort(int arr[], int left, int right);
void merge(int arr[], int left, int mid, int right);

int main() {
    int n, i;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    int arr[n];

    printf("Enter %d integers:\n", n);
    for (i = 0; i < n; i++) {
        printf("Element %d: ", i + 1);
        scanf("%d", &arr[i]);
```

```c
    }

    printf("\nBefore sorting:\n");
    for (i = 0; i < n; i++) {
        printf("%d\t", arr[i]);
    }
    printf("\n");

    mergeSort(arr, 0, n - 1);

    printf("\nAfter sorting (ascending order):\n");
    for (i = 0; i < n; i++) {
        printf("%d\t", arr[i]);
    }
    printf("\n");

    return 0;
}

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);
    }
}

void merge(int arr[], int left, int mid, int right) {
    int i, j, k;
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1];  // left sub array
    int R[n2];  // right sub array

    for (i = 0; i < n1; i++)
        L[i] = arr[left + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[mid + 1 + j];

    i = 0; j = 0; k = left;

    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k++] = L[i++];
```

```
        } else {
            arr[k++] = R[j++];
        }
    }

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

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

OUTPUT:

```
PS D:\DAOA> gcc mergesort.c -o mergesort
>>
PS D:\DAOA> ./mergesort
Enter the number of elements: 4
Enter 4 integers:
Element 1: 1
Element 2: 7
Element 3: 0
Element 4: 9

Before sorting:
1       7       0       9

After sorting (ascending order):
0       1       7       9
```

TIME COMPLEXITY ANALYSIS:

Analyze the time complexity of Merge Sort using both the Master Theorem and the Substitution Method.

1) Master Method

MERGE SORT

$\Rightarrow T(n) = 2\,T(n/2) + n.$

By Master Method comparing,

$T(n) = a\,T(n/b) + f(n)$

$\therefore a = 2 \quad b = 2 \quad f(n) = n.$

i) $O(n^{\log_b a}) = O(n^{\log_2 2}) = O(n^1) = n.$

ii) $\Theta(n^{\log_b a} \log n) = O(n^{\log_2 2} \log n) = O(n \log n)$

iii) $n^{\log_b a} = f(n)$

$n^{\log_2 2} = f(n)$

$\therefore \boxed{n = n}$

2) Substitution Method.

$T(n) = 2\,T(n/2) + n.$

$\therefore T(1) = k$

Substituting n by $n/2$

$T(n) = 2^k\,T\left(\frac{n}{2^k}\right) + k\,cn.$

$\therefore T(n) = 2\,T\left(\frac{n}{2}\right) + cn \quad -(i)$

$T(n/2) = 2T\left(\frac{n}{4}\right) + c\frac{n}{2}$

$\therefore T(n) = 2\left[2\,T\left(\frac{n}{4}\right) + c\frac{n}{2}\right] + cn$

$\therefore T(n) = 4\,T\left(\frac{n}{4}\right) + 2cn \quad -(ii)$

$n = n/4 \quad -$ in (i)

$T(n/4) = 2\,T\left(\frac{n}{8}\right) + c\left(\frac{n}{4}\right) \quad -$

$\therefore T(n) = 4\left[2\,T\left(\frac{n}{8}\right) + c\left(\frac{n}{4}\right)\right] + cn$

$T(n) = 8\,T\left(\frac{n}{8}\right) + 3cn. \quad -(iii)$

$\boxed{\therefore T(n) = 2^k\,T\left(\frac{n}{2^k}\right) + k\,cn}$

$\therefore \frac{n}{2^k} = 1$

$\therefore n = 2^k$

$\log_2 n = k$

$\therefore T(n) = 2^{\log_2 n}\,T(1) + cn\log_2 n$

$= nd + cn\log_2 n$

$\boxed{\therefore T(n) = O(n \log n)}$

CONCLUSION:

Merge Sort does not exhibit distinct best or worst cases, as it performs consistently regardless of input order. The algorithm makes log n levels of division until subarrays of size 1 are

reached. At each level, merging takes O(n) time, resulting in an overall time complexity of O(n logn). However, it requires auxiliary space equal to the size of the original array.