**Experiment No. 2**

**Aim:** To implement the Merge sort.

**Theory:**

### Basically, Merge sort is a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.

**Merge sort can be explained simply as splitting the array into two halves, sorting each half, and then merging the sorted halves back together. This process is repeated until the entire array is sorted.** **One of the main advantages of merge sort is that it has a time complexity of O(n log n), which means it can sort large arrays relatively quickly. It is also a stable sort, which means that the order of elements with equal values is preserved during the sort.**

**Complexity of Insertion sort**

* Average Time complexity - O(N logn)
* Best case - O(N logn)
* Worst case - O(N logn)
* Space complexity - O(N)

## What is Merge Sort Algorithm?

#### The concept of Divide and Conquer involves three steps:

#### Divide the problem into multiple subproblems.

#### Solve the Sub Problems. The idea is to break down the problem into atomic subproblems, where they are actually solved.

#### Combine the solutions of the subproblems to find the solution of the actual problem.

#### So, the Merge sort working rule involves the following steps:

#### Divide the unsorted array into subarray, each containing a single element.

#### Take adjacent pairs of two single-element array and merge them to form an array of 2 elements.

#### Repeat the process till a single sorted array is obtained.

Algorithm for Merge Sort :

Merge\_Sort( low, high)

{

If( low < high ){

Mid = [(low + high)/2]

Merge\_sort(mid+1, high)

Merge\_sort(low, mid)

Merge(low, mid, high)

}

}

## 

**Code:**

import java.util.\*;

public class Main

{

public static void merge(int A[], int mid, int low, int high){

int i = low, k= low, j = mid+1;

System.out.println("\nlow is "+low+", mid is "+mid+", high is "+high);

int B[] = new int[5];

while(i <= mid && j <= high){

if (A[i] < A[j]) {

B[k] = A[i];

i++; k++;

}

else{

B[k] = A[j];

j++; k++;

}

}

// Emptying all the remaining elements in the array

while(i <= mid){

B[k] = A[i];

i++; k++;

}

while(j <= high){

B[k] = A[j];

j++; k++;

}

// Copying elements of array B into A

for(int x = low; x <= high; x++){

A[x] = B[x];

}

//Print test

for(int m : B){

System.out.print(m+" ");

}

}

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

if(low < high){

int mid = (int)Math.floor((low + high)/2);

// System.out.println("\n mid is = "+mid);

mergeSort(A, low, mid);

mergeSort(A, mid+1, high);

merge(A, mid, low, high);

}

}

public static void printArray(int []arr){

for(int i : arr){

System.out.print(i+" ");

}

System.out.println();

return;

}

public static void main(String[] args) {

// int []arr = {2,1,7,3,8};

Scanner sc = new Scanner(System.in);

int []arr = new int[5];

System.out.println("Enter array elements");

for(int i = 0; i<5; i++){

arr[i] = sc.nextInt();

}

System.out.print("Original array before Merge sort: ");

printArray(arr);

mergeSort(arr, 0, 4);

System.out.println();

System.out.print("Merge Sort in ascending order: ");

printArray(arr);

}

}

**Output:**

Enter array elements

2

1

7

3

8

Original array before Merge sort: 2 1 7 3 8

low is 0, mid is 0, high is 1

1 2 0 0 0

low is 0, mid is 1, high is 2

1 2 7 0 0

low is 3, mid is 3, high is 4

0 0 0 3 8

low is 0, mid is 2, high is 4

1 2 3 7 8

Merge Sort in ascending order: 1 2 3 7 8

\*\* Process exited - Return Code: 0 \*\*

**Conclusion:**

The purpose of this experiment was to demonstrate how Merge Sort works and how to implement them. In addition, we learned about their time and space complexity.