-Experiment No: 2

MERGE SORT

**Aim:**

To implement the merge sort using divide and conquer approach

**Theory:**

Merge Sort:

Merge sort is one of the most efficient sorting algorithms. It works on the principle of Divide and Conquer. Merge sort repeatedly breaks down a list into several sub lists until each sub list consists of a single element and merging those sub lists in a manner that results into a sorted list.

It follows the divide and conquer approach

1. Divide the unsorted list into N sub lists, each containing 1 element.

2. Take adjacent pairs of two singleton lists and merge them to form a list of 2 elements. N will now convert into N/2 lists of size 2.

3. Repeat the process till a single sorted list of obtained.

While comparing two sub lists for merging, the first element of both lists is taken into consideration. While sorting in ascending order, the element that is of a lesser value becomes a new element of the sorted list. This procedure is repeated until both the smaller sub lists are empty and the new combined sub list comprises all the elements of both the sub lists

**ALGORITHM**

**Algorithm**  merge(int Arr[], int start, int mid, int end)

// Problem description: Merge two sorted arrays where first array starts from start to mid and the

// second starts from mid + 1 to end

//Input: Arr array is sorted from start to mid and Arr is sorted from index position mid+1 to high

//Output: Arr array is a sorted from start to end

// create a temp array

int temp[end - start + 1];

i = start, j = mid+1, k = 0;

// traverse both arrays and in each iteration add smaller of both elements in temp

while(i <= mid && j <= end)

{

if(Arr[i] <= Arr[j])

{

temp[k] = Arr[i];

k += 1;

i += 1;

}

else

{

temp[k] = Arr[j];

k += 1;

j += 1;

}

}

// add elements left in the first interval

while(i <= mid)

{

temp[k] = Arr[i];

k += 1;

i += 1;

}

// add elements left in the second interval

while(j <= end)

{

temp[k] = Arr[j];

k += 1;

j += 1;

}

// copy temp to original interval

for(i = start; i <= end; i += 1)

{

Arr[i] = temp[i - start]

}

**Algorithm mergeSort(Arr, start, end)**

// Problem description: Sort the elements of the array between the lower bound and upper bound

//Input: // Arr is an array of integer type

// start and end are the starting and ending index of current interval of Arr

//Arr array is unsorted within start and end as lower bound and upper bound

//Output: Arr array is a sorted from start to end

if(start < end)

{

int mid = (start + end) / 2;

mergeSort(Arr, start, mid);

mergeSort(Arr, mid+1, end);

merge(Arr, start, mid, end);

}

}

**Conclusion**: Merge sort using divide and conquer approach was studied and implemented successfully.