**EXPERIMENT-4**

**AIM:** Merge Sort with its time complexity.

**CODE:**

#include <iostream>

using namespace std;

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

    int i,j,k;

    int B[50];

    i=k=low;

    j= mid+1;

    while(i<=mid && j<=high){

        if(A[i]<=A[j]){

            B[k++] = A[i++];

        }

        else{

            B[k++] = A[j++];

        }

    }

    for(;i<=mid; i++){

        B[k++] =A[i];

    }

    for(; j<=high; j++){

        B[k++] = A[j];

    }

    for(i=low;i<=high;i++){

        A[i] = B[i];

    }

}

void Iterative\_merge\_sort(int A[], int size){

    int p,i,low,mid,high;

    for(p=2; p<=size; p=p\*2){

        for(i = 0; i+p-1<size; i=i+p){

            low = i;

            high = i+p-1;

            mid = (low+high) / 2;

            merge(A,low,mid,high);

        }

    }

    if((p/2) < size){

        merge(A,0,(p/2)-1,size);

    }

}

void display\_elements\_arr(int A[], int x)

{

    cout << "| ";

    for (int i = 0; i < x; i++)

    {

        cout << A[i] << " | ";

    }

    cout << endl;

}

void fill\_array(int A[], int size){

    int i;

    for(int i =0; i<size; i++){

        cout<<"Enter data at position "<<i<<" : ";

        cin>>A[i];

    }

}

int main(){

    int size;

    cout<<"Enter size: ";

    cin>>size;

    int \*A = new int(size);

    fill\_array(A, size);

    display\_elements\_arr(A, size);

    Iterative\_merge\_sort(A,size);

    display\_elements\_arr(A, size);

    return 0;

}

**Time complexity of this program is O(n\*log n).**

If there are n elements in the array, then the Iteration\_merge\_sort function will call merge function log n times and every time the merge function will have to execute n number of comparison.