**Experiment No:** 2  **Date:** 02/04/2021

**Aim:** Implementation of Merge Sort Using

Divide and Conquer and obtain its step count

**Theory:**

**Merge Sort**

* Merge Sort is one of the most popular sorting algorithms that is based on the principle of Divide and Conquer Algorithm.
* Merge Sort is a Divide and Conquer algorithm.
* It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.
* The merge() function is used for merging two halves.
* The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.

**Algorithm**

**Algorithm MergeSort(low, high)**

//a[low: high] is a global array to be sorted. Small(P) is true if there is only one element to sort.

//In this case the list is already sorted

{

If ( low < high ) then //if there are more than one element

{

//divide P into subproblems. Find where to split the set

mid := [ (low+high)/2];

//solve the subproblems.

MergeSort (low, mid);

MergeSort(mid+1,high);

//combine the solutions

Merge(low,mid,high);

}

}

**Algorithm Merge(low,mid,high)**

//a[low:high] is a global array containing two sorted subsets in a a[low:mid] and in a [mid+1: high].

//the goal is to merge these two sets into a single set residing in a[low:high].

//b[] is an auxiliary global array

{

h :=low;I :=low;j := mid+1;

while ((h≤mid) and (j≤high)) do

{

If (a[h]≤a[j]) then

{

b[i] := a[h];h :=h+1;

}

else

{

b[i] := a[j]; j := j+1;

}

i := i+1;

}

If (h>mid) then

for k := j to high do

{

b[i] := a[k];i :=i+1;

}

else

for k := h to mid do

{

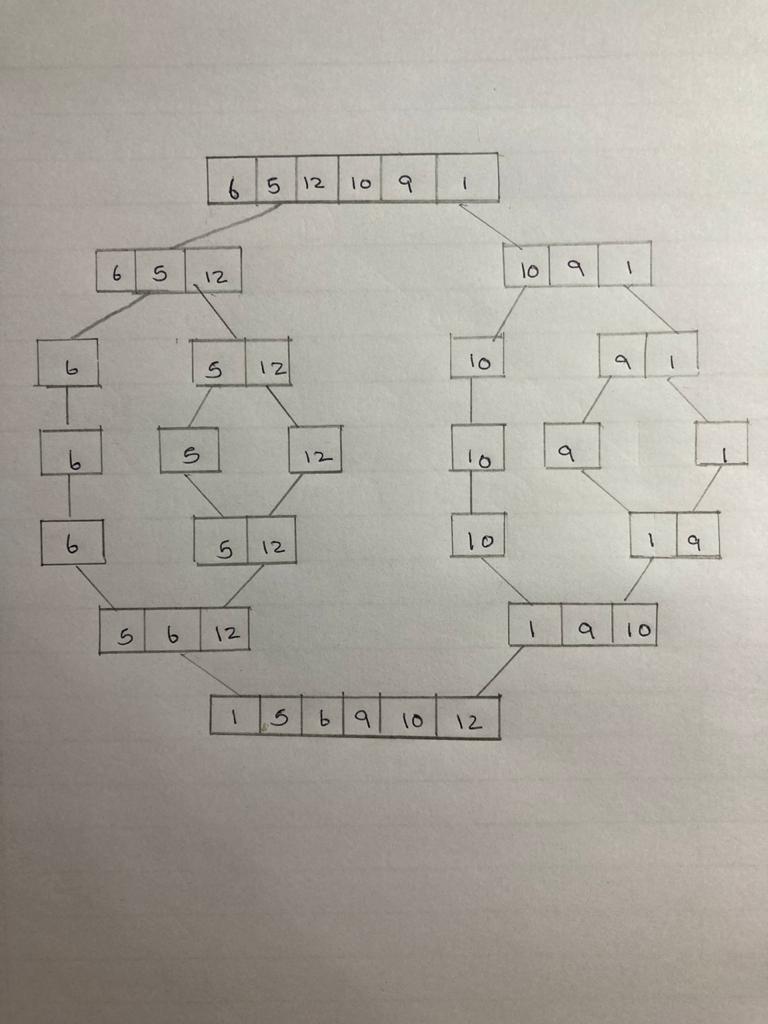
b[i] := a[k];i :=i+1;

}

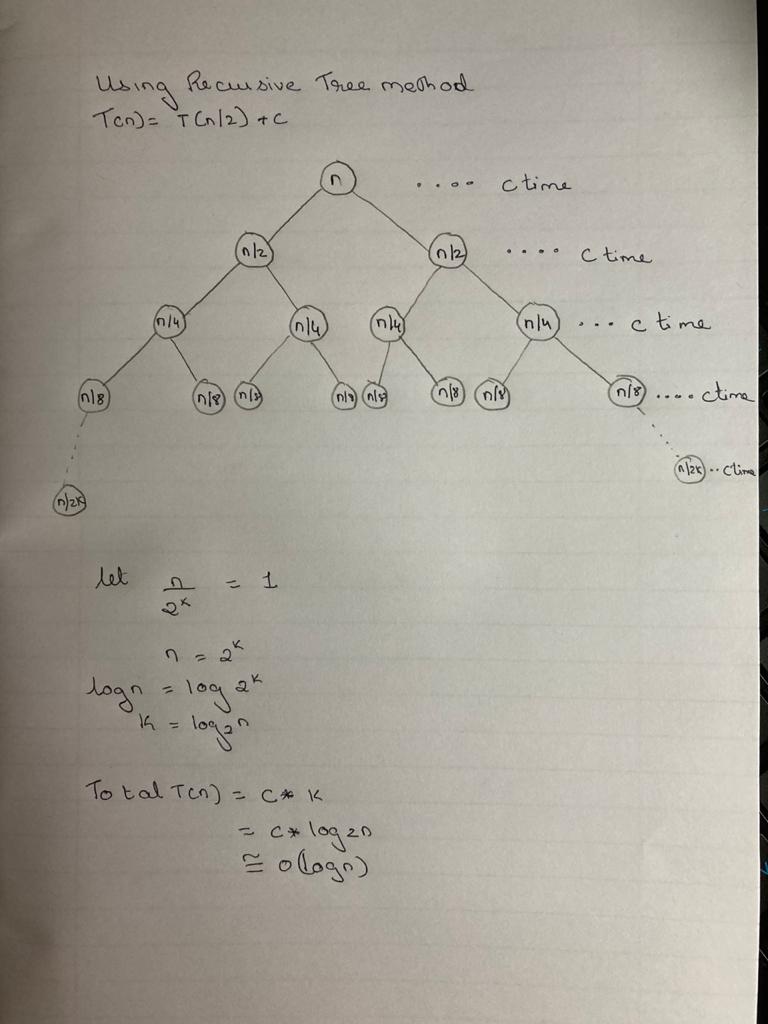
for k := low to high do a[k] := b[k];

}

**Tracing with Examples**



**Using Recursive Tree Method**



**Recurrence Relation Formulation and Solving**

* Time Complexity:
  + Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

**T(n) = 2T(n/2) + cn**

* + The above recurrence can be solved either using the Recurrence Tree method or the Master method.
  + It falls in case II of Master Method and the solution of the recurrence is θ(nLogn).
  + Master’s theorem:

**a=2 b=2 f(n)=cn**

**n(log(base b)a)**

**n(log2(2))**

**n1=n**

**T(n) =n\*U(n)**

**u(n) => h(n) = f(n)/n=cn/n=c=>r=0=>i=0**

**(Log(n))(i+1)/(i+1)**

**=log(n)**

**T(n) = n\*log(n)**

**=O(nlog(n))**

* + Time complexity of Merge Sort is O(nLogn) in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.
* **Auxiliary Space:** O(n)
* **Algorithmic Paradigm:** Divide and Conquer
* **Sorting in Place:** No in a typical implementation
* **Stable:** Yes

**PROGRAM**

#include<iostream>

using namespace std;

int count = 0;

int a[10]={1,2020,28,852,852,8,85,8,50,25};

int b[10];

void Merge(int low,int mid,int high)

{

count++;

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

while((h<=mid)&&(j<=high))

{

count++;

if(a[h]<=a[j])

{

b[i]=a[h];

h=h+1;

}

else

{

b[i]=a[j];

j++;

}

i=i+1;

}

count++;

if(h>mid)

{

count++;

for(int k=j;k<=high;k++)

{

b[i]=a[k];

i++;

}

}

else

for(int k=h;k<=mid;k++)

{

b[i]=a[k];

i++;

}

for(int k=low;k<=high;k++)

{

a[k]=b[k];

}

}

void mergesort(int low,int high)

{

if(low<high)

{

count++;

int mid = (low+high)/2;

mergesort(low,mid);

mergesort(mid+1,high);

Merge(low,mid,high);

}

}

int main()

{

int size;

cout<<"-----------------"<<endl;

cout << "BEFORE MERG SORT "<<endl;

cout<<"-----------------"<<endl;

count++;

for(int i=0;i<10;i++)

cout << a[i]<<" ";

cout<<endl<<endl;

cout<<"-----------------"<<endl;

cout << "AFTER MERG SORT "<<endl;

cout<<"-----------------"<<endl;

mergesort(0,9);

count++;

for(int i=0;i<10;i++)

cout << a[i]<<" ";

cout<<endl<<endl;

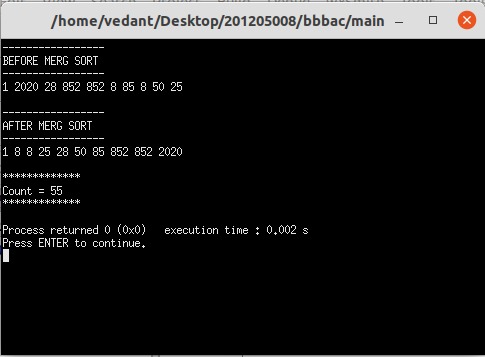
cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Count = "<<count<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

**OUTPUT**



**CONCLUSION**

* Detailed concept of Merge Sort using Divide and Conquer methods was studied successfully.
* Merge sort program were executed successfully.