**NAME:** SHUBHAM TAKANKHAR

**CLASS:** SY MCA

**ROLL NO:** 54

**ASSIGNMENT 1:** Merge Sort implementations using divide and conquer approach and time complexity analysis.

**SOURCE CODE:**

**<<Merge Sort Implementation>>**

import java.util.Scanner;

import java.util.Random;

public class mergeSorts {

    mergeSorts(int arr[], int start, int end) {

        divide(arr, start, end);

    }

    void divide(int arr[], int start, int end) {

        if (start < end) {

            int mid = ((start + end) / 2);

            divide(arr, start, mid); n\*(log n)

            divide(arr, mid + 1, end); (log n)

            conquer(arr, start, mid, end); (log n)

        }

    }

    void conquer(int[] arr, int start, int mid, int end) {

        int[] arr2 = new int[arr.length];

        int i = start;

        int j = mid + 1;

        int k = start;

        while (i <= mid & j <= end) {

            if (arr[i] < arr[j]) {

                arr2[k] = arr[i];

                i++;

            } else {

                arr2[k] = arr[j];

                j++;

            }

            k++;

        }

        while (i <= mid) {

            arr2[k] = arr[i];

            k++;

            i++;

        }

        while (j <= end) {

            arr2[k] = arr[j];

            k++;

            j++;

        }

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

            arr[i] = arr2[i];

        }

    }

    public static void main(String[] args) {

        Random rd = new Random();

        System.out.println("Enter Size of Array");

        Scanner input = new Scanner(System.in);

        int n = input.nextInt();

        input.close();

        int arr[] = new int[n];

        // generating random array

        for (int i = 0; i < arr.length; i++) {

            arr[i] = rd.nextInt(50);

        }

        System.out.println("Before Sorting:");

        for (int items : arr) {

            System.out.print(items + ",");

        }

        new mergeSorts(arr, 0, arr.length - 1);

        System.out.println("\nAfter Sorting:");

        for (int items : arr) {

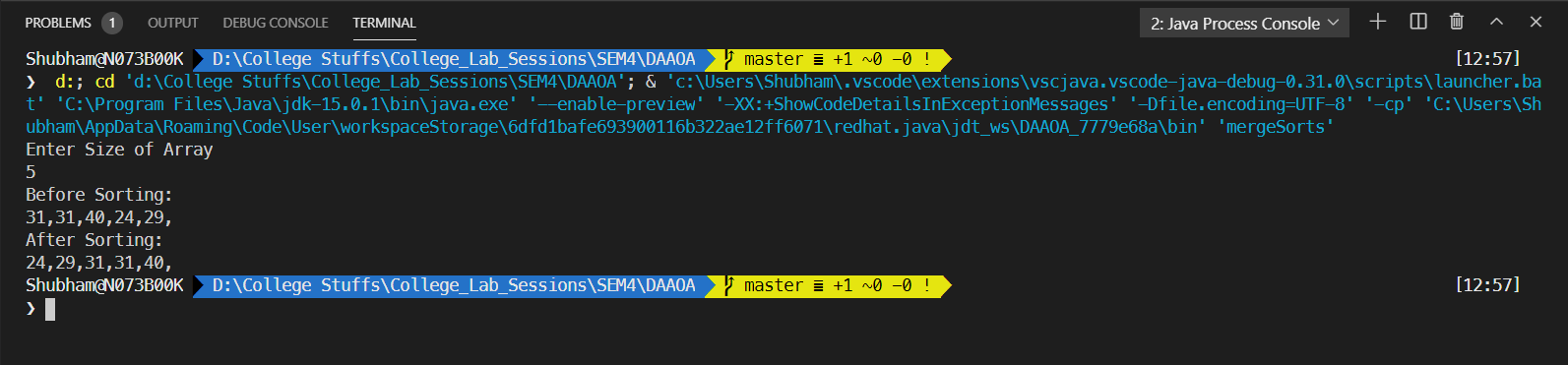
            System.out.print(items + ",");

        }

    }

}

**OUTPUT:**



**Time complexity measure of Merge sort**

In merge sort we divide the array into two so we have:

T() + T = 2T

*Where,*

*NL = left half array*

*NR= right half*

Finally we merge these two sub arrays using merge procedure which takes Θ(n) time

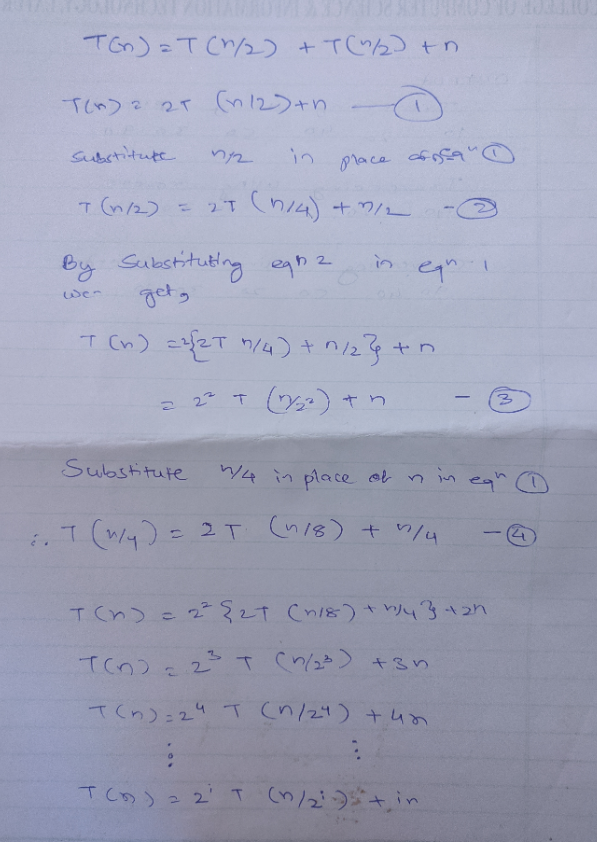
Merge Sort is a recursive algorithm and time complexity can be expressed as:

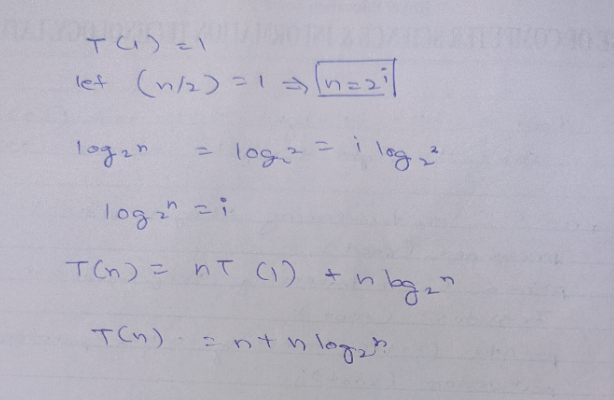
**T(n) = 2T + θ(n)**

*Where,*

*n = time taken to merge these two sub arrays*

**Analysis:**

****

****

Therefore,

Worst Case Time Complexity [ Big-O ]: O(n\*log n)

Best Case Time Complexity [Big-omega]: O(n\*log n)

Average Time Complexity [Big-theta]: O(n\*log n)

**Binary search time complexity computation**

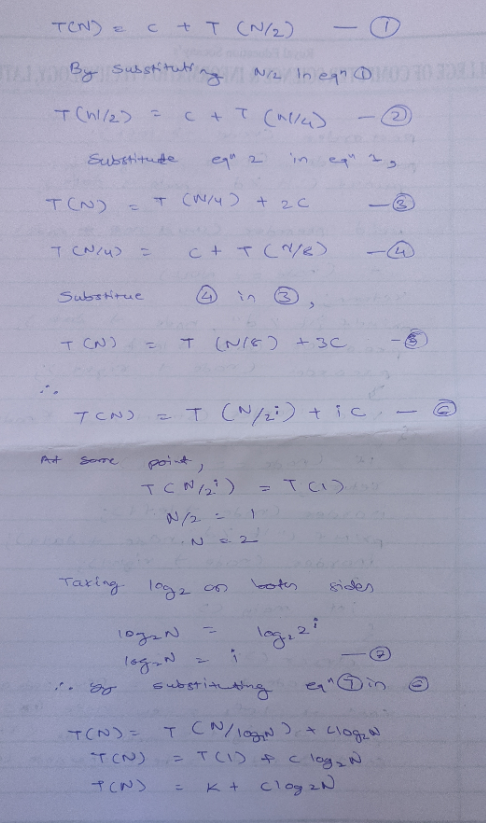
Binary search cuts down the list of elements into half So we have:

**T(N) =C + T**

*Where,*

*T = time taken to search in half of the array*

*C = constant time to compare midpoint and divide the list*

**

Therefore,

Worst Case Time Complexity [ Big-O ]: O(log n)

Best Case Time Complexity [Big-omega]: O(1)

Average Time Complexity [Big-theta]: O(log n)