LAB

REPORT

CSE 114 : Data Structure and Algorithms Sessional

|  |  |
| --- | --- |
| PREPARED BY  Mehrin Farzana  ID: 2101013  Session: 2021-2022  Date: 30/07/2023 | SUPERVISED BY  Suman Saha  Lecturer  Department of IRE, BDU |

|  |  |
| --- | --- |
| logo1 | BANGABANDHU SHEIKH MUJIBUR RAHMAN DIGITAL UNIVERSITY  (BDU) |
|  |  |

**List of Problems**

1. Given an array arr[], its starting position l and its ending position r. Sort the array using merge sort algorithm.

Constraints: 1 <= N <= 10 5 1 <= arr[i] <= 10 5 .

1. Analyze time complexity and space complexity of the Merge Sort Algorithm.

**Problem No.:** 01

**Problem Statement:**

Write down a program that implements three sorting algorithms (bubble sort, selection sort, and insertion sort). Using a switch statement to choose the desired algorithm.

**Code:**

#include <stdio.h>

void merge\_sort(long int arr[], long int l, long int r)

{

if (l < r) {

long int m = l + (r - l) / 2;

merge\_sort(arr, l, m);

merge\_sort(arr, m + 1, r);

long int i, j, k;

long int n1 = m - l + 1;

long int n2 = r - m;

long int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

}

int main()

{

long int n;

scanf("%ld", &n);

long int a[n];

for(long int i=0; i<n; i++){

scanf("%ld", &a[i]);

}

merge\_sort(a, 0, n - 1);

for(long int i=0; i<n; i++){

printf("%ld ", a[i]);

}

return 0;

}

**Output:**

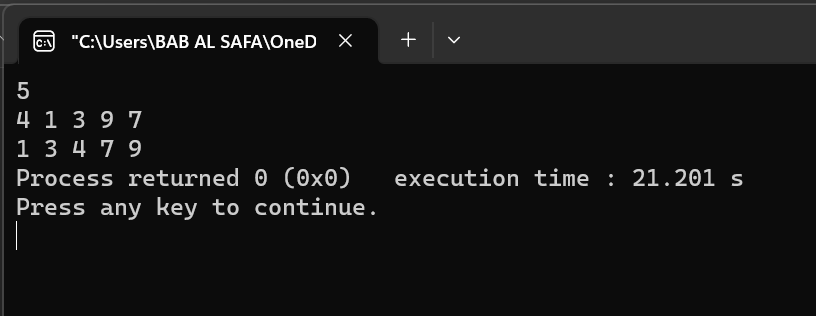


Fig 1.1: Output on console for case 1.

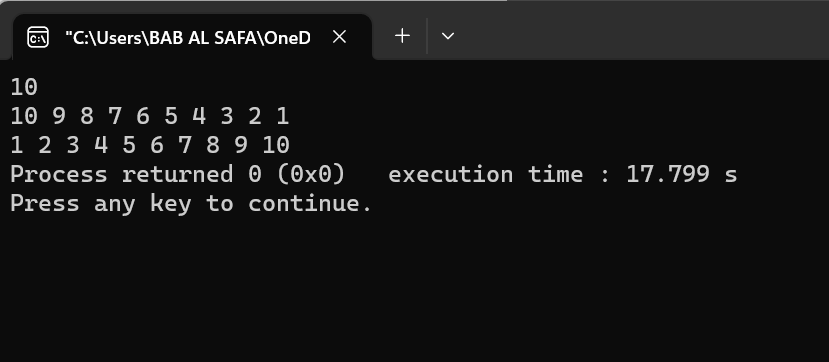


Fig 1.2: Output on console for case 2.

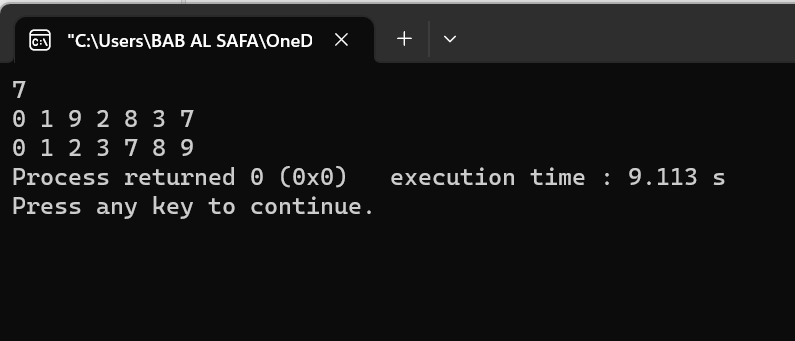


Fig 1.3: Output on console for case 3.

**Problem No.:** 02

**Problem Statement:**

Analyze time complexity and space complexity of the Merge Sort Algorithm.

**Code:**

#include <stdio.h>

void merge\_sort(long int arr[], long int l, long int r)

{

if (l < r) {

long int m = l + (r - l) / 2;

merge\_sort(arr, l, m);

merge\_sort(arr, m + 1, r);

long int i, j, k;

long int n1 = m - l + 1;

long int n2 = r - m;

long int L[n1], R[n2];

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

i = 0;

j = 0;

k = l;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

}

}

int main()

{

long int n;

scanf("%ld", &n);

long int a[n];

for(long int i=0; i<n; i++){

scanf("%ld", &a[i]);

}

merge\_sort(a, 0, n - 1);

for(long int i=0; i<n; i++){

printf("%ld ", a[i]);

}

return 0;

}

**Analysis:**

The time complexity of merge sort for best case, average case and worst case is **O(N \* logN)**. The space complexity of merge sort algorithm is **O(N)** .

 Merge Sort is a sorting algorithm that uses the divide and conquer approach. It keeps on dividing the array into two parts until they cannot be divided further and then they are sorted and merged back together into sorted arrays.

Thus, The time complexity of merge sort for best case, average case and worst case is **O(N \* logN)**.

In case of merge sort, we need to build a temporary array to merge the sorted arrays. So the auxiliary space requirement is **O(N)**.

Thus, the total space required is **O(N).**