Data Structures and Algorithms

Lab Report

Lab09

Group Members Name & Reg #:	**Muhammad Haris Irfan** **(FA18-BCE-090)**
Class	Data Structures and Algorithms CSC211 (**BCE-3B**)
Instructor's Name	Dilshad Sabir

In Lab Tasks

Task:1

Complete the merge () function for Merge Sort.

Solution:

The code is shown below,

```
void merge(int * ptr_array, int l, int m, int r)
{
     /*** Complete this function ***/

    int i, j, k;

    int n1 = m - l + 1;
    int n2 =  r - m;
    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = *(ptr_array+l + i);
    for (j = 0; j < n2; j++)
        R[j] = *(ptr_array+m + 1+ j);
/*********************************************/
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
           *(ptr_array+k) = L[i];
            i++;
        }
        else
        {
            *(ptr_array+k) = R[j];
            j++;
        }
        k++;
    }
    while (i < n1)
    {
        *(ptr_array+k) = L[i];
```

```
        i++;
        k++;
    }


    while (j < n2)
    {
        *(ptr_array+k) = R[j];
        j++;
        k++;
    }
}
```

The Result of the following code is attached below:

```
C:\Users\Hp\Documents\CodeBlocks\C\DataStructures\Lab09\bin\Debug\Lab09.exe

How do you want to sort the Array?
1-Merge Sort?
2-Quick Sort?
3-EXIT

Enter your choice? 1-3
1
Unsorted array:
743     204     491     738     348     150     563      99     855     524     923     388     910     334     722
        153

Sorted array:
 99     150     153     204     334     348     388     491     524     563     722     738     743     855     910
        923



Time elpased is 3.022000 seconds
Process returned 0 (0x0)   execution time : 3.281 s
Press any key to continue.
```

Task:2

Complete the partition () function Quick Sort.

Solution:

The code is shown below,

```
int partition(int * ptr_array, int start_idx, int end_idx)
{

    /** To Complete **/
int pivot =*(ptr_array+end_idx);
    int i = (start_idx - 1);

    for (int j = start_idx; j <= end_idx- 1; j++)
    {

        if (*(ptr_array+j) < pivot)
        {
            i++;
            swap((ptr_array+i), (ptr_array+j));
        }
    }
    swap((ptr_array+i+1), (ptr_array+end_idx));
    return (i + 1);
}
```

The Result of the following code is attached below:

```
C:\Users\Hp\Documents\CodeBlocks\C\DataStructures\Lab09\bin\Debug\Lab09.exe

How do you want to sort the Array?
1-Merge Sort?
2-Quick Sort?
3-EXIT

Enter your choice? 1-3
2
Unsorted array:
841     976     360     742     873     880     632     585     683     690     486     709     716     204     127
          8

Sorted array:
  8     127     204     360     486     585     632     683     690     709     716     742     841     873     880
        976



Time elpased is 1.059000 seconds
Process returned 0 (0x0)   execution time : 1.070 s
Press any key to continue.
```

Post Lab Task.

Task 3:

Study and perform comparative analysis between different sorting algorithms we have implemented in current and previous Lab.

Solution

In last lab we determined the time of each type of sort, and we reached the conclusion that Merge Sort was indeed the best type of sort as it took least time.

Data Size	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort
16	1.14s	1.74s	0.90s	0.74s
128	0.90s	0.84s	0.58s	0.47s
1024	1.26s	1.46s	1.10s	0.64s
16384	2.86s	2.55s	2.43s	1.78s
131072	55.18s	29.88s	21.12s	11.7s

But in this lab we learnt about quick sort, it is similar to merge sort but it uses inplace sorting due to which is is abit faster then even merge sort, for the worst case, it has a quadratic running time given as **O(n2)**, however its efficient implementation can minimize the probability of requiring quadratic time.

Quick sort does not require additional memory space to store data, whereas Merge Sort algorithm requires **O(n) additional memory** for sorting in **O(nlogn)** time, hence we can safely conclude that Quick Sort is the best type of sorting keeping time barrier in mind.

THE END