Green University of Bangladesh
Department of Computer Science and Engineering(CSE)

Faculty of Sciences and Engineering

Semester: (Spring, Year:2021), B.Sc. in CSE (Day)

LAB REPORT NO 03

Course Code:206 Section:DB

Lab Experiment Name: Implementation of Quick sort and Merge sort

Student Details

	Name	ID
1.	Shamim Ahmed	201902067

Course Teacher's Name : Monoshi Kumar Roy

[For Teachers use only: Don't Write Anything inside this box]

Lab Report Status Marks: ……

Signature:……

.

TITLE OF THE LAB EXPERIMENT

Implementation of Quick sort and Merge sort for 5000 numbers.

OBJECTIVES/AIM

From this lab we will learn about Divide and Conquer algorithm and after that lab we would be able to sort numbers by quick sort and merge sort.

PROCEDURE / ANALYSIS / DESIGN

Quick sort : QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

Quick sort Algorithm:

Step 1 - Consider the first element of the list as pivot (i.e., Element at first position in the list).

Step 2 - Define two variables i and j. Set i and j to first and last elements of the list respectively.

Step 3 - Increment i until list[i] > pivot then stop.

Step 4 - Decrement j until list[j] < pivot then stop.

Step 5 - If i < j then exchange list[i] and list[j].

Step 6 - Repeat steps 3,4 & 5 until i > j.

Step 7 - Exchange the pivot element with list[j] element.

Quick sort Pseudocode:

```
quickSort(arr[], low, high)
{
   if (low < high)
   {
      /* pi is partitioning index, arr[p] is now
         at right place */
      pi = partition(arr, low, high);

      quickSort(arr, low, pi - 1);  // Before pi
      quickSort(arr, pi + 1, high); // After pi
   }
}
```

Marge sort : 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.

Marge sort algorithm :

Step 1 − if it is only one element in the list it is already sorted, return.

Step 2 − divide the list recursively into two halves until it can no more be divided.
Step 3 − merge the smaller lists into new list in sorted order.

Marge sort procedure:

```
procedure mergesort( var a as array )
if ( n == 1 ) return a
        var l1 as array = a[0] ... a[n/2]
        var l2 as array = a[n/2+1] ... a[n]
        l1 = mergesort( l1 )
        l2 = mergesort( l2 )
   return merge( l1, l2 )
end procedure


procedure merge( var a as array, var b as array )
        var c as array
        while ( a and b have elements )
           if ( a[0] > b[0] )
                add b[0] to the end of c
              remove b[0] from b
               else
                add a[0] to the end of c
                remove a[0] from a
               end if
        end while

         while ( a has elements )
          add a[0] to the end of c
          remove a[0] from a
          end while

        while ( b has elements )
         add b[0] to the end of c
         remove b[0] from b
        end while
         return c

        end procedure
```

1 IMPLEMENTATION

Quick sort:

```
#include<bits/stdc++.h>
//#include<cstdlib>
```

```
using namespace std;
int cnt=0;

void swap(int *a, int *b)
{
   int temp;
   temp = *a;
   *a = *b;
   *b = temp;
   cnt++;
}

int Partition(int a[], int l, int h)
{
   int pivot, index, i;
   index = l;
   pivot = h;
   for(i = l; i < h; i++)
    {
      if(a[i] < a[pivot])
      {
         swap(&a[i], &a[index]);
         index++;
      }
   }

   swap(&a[pivot], &a[index]);
   return index;
}

int RandomPivotPartition(int a[], int l, int h)
{
   int pvt, n, temp;
   n = rand();
   pvt = l + n%(h-l+1);
   swap(&a[h], &a[pvt]);
   return Partition(a, l, h);
}
```

```
int Quick_Sort(int a[], int l, int h)
{
   int pindex;
   if(l < h)
   {
      pindex = RandomPivotPartition(a, l, h);
      Quick_Sort(a, l, pindex-1);
      Quick_Sort(a, pindex+1, h);
   }
   return 0;
}

int main()
{
   ifstream input;
   ofstream output;
   int n=5000;
   int arr[5000];
   int number,i;


   output.open("Output_for_Quick_sort.txt");

   cout<<"The Elements Are : \n\n";

   for(i=0; i<5000; i++)
   {
      number=rand()%5000;
      cout<<number<<" ";
      output<<number<<" ";
   }
   output.close();
   cout<<endl;

   input.open("Output_for_Quick_sort.txt");
   output.open("Output.txt");

   for(i=0;i<5000;i++)
   {
```

```
        input>>arr[i];
    }
    auto arr_size = sizeof(arr) / sizeof(arr[0]);
    Quick_Sort(arr, 0, n-1);
    cout<<"\n\n\nAfter Quick Sort Operation : \n";
    for (i = 0; i < arr_size; i++)
    {
        cout<<arr[i]<<" ";
        output<<arr[i];
    }
    cout<<endl;
    cout<< "Number of swap : ";
    cout<<cnt;
    return 0;
}
```

Marge sort :

```
#include <bits/stdc++.h>

using namespace std;

void merge(int arr[], int const p, int const  q, int const r)
{

    int n1 = q - p + 1;
    int n2 = r - q;

    int L[n1], M[n2];

    for (int i = 0; i < n1; i++)
        L[i] = arr[p + i];
    for (int j = 0; j < n2; j++)
        M[j] = arr[q + 1 + j];


    int i, j, k;
    i = 0;
```

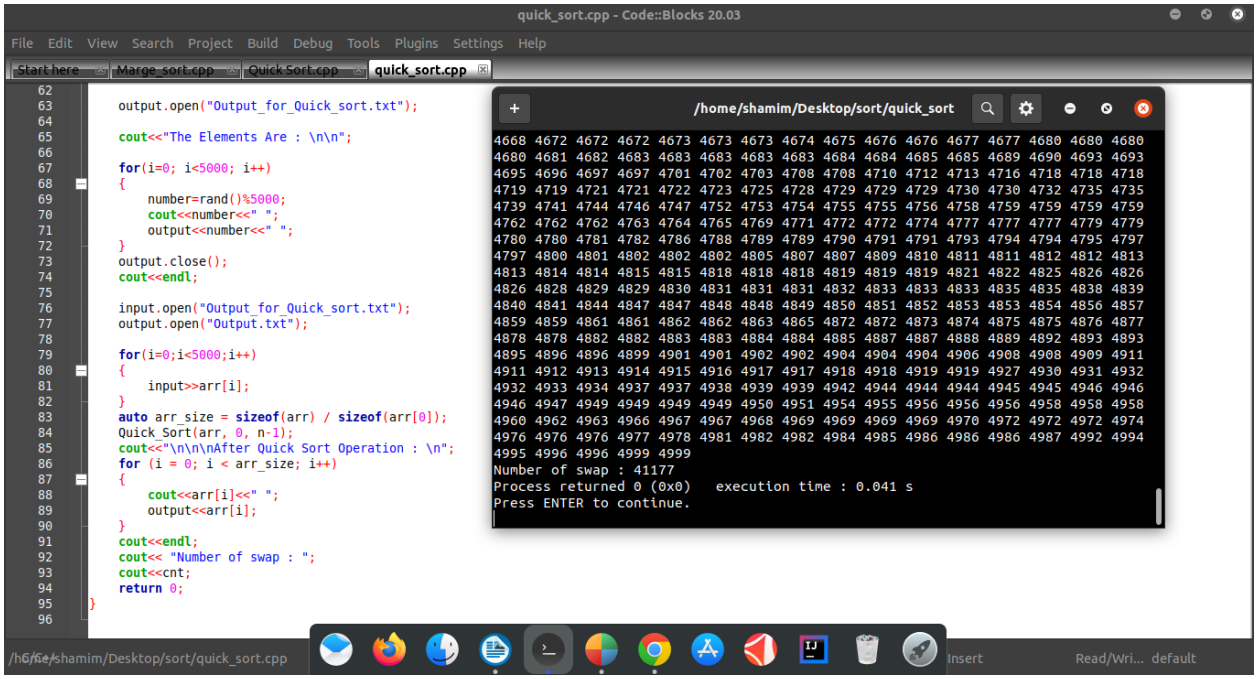
```
    j = 0;
    k = p;


    while (i < n1 && j < n2)
    {
        if (L[i] <= M[j])
    {
        arr[k] = L[i];
        i++;
    }
    else
    {
        arr[k] = M[j];
        j++;
    }
    k++;
    }


  while (i < n1)
  {
    arr[k] = L[i];
    i++;
    k++;
  }

    while (j < n2)
  {
    arr[k] = M[j];
    j++;
    k++;
  }
}


void mergeSort(int arr[], int l, int r)
{
  if (l < r) {
```

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

    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);


    merge(arr, l, m, r);
  }
}


void printArray(int arr[], int size)
{
  for (int i = 0; i < size; i++)
    cout << arr[i] << " ";
  cout << endl;
}
int main()
{
    ifstream input;
    ofstream output;
    int n=5000;
    int arr[5000];

    input.open("Output.txt");
    output.open("Output_for_Marge_Sort.txt");
    int i;

    for(i=0;i<5000;i++)
    {
        input>>arr[i];
    }
    auto arr_size = sizeof(arr) / sizeof(arr[0]);

  mergeSort(arr, 0, arr_size-1);

  cout << "Sorted array: \n";
```

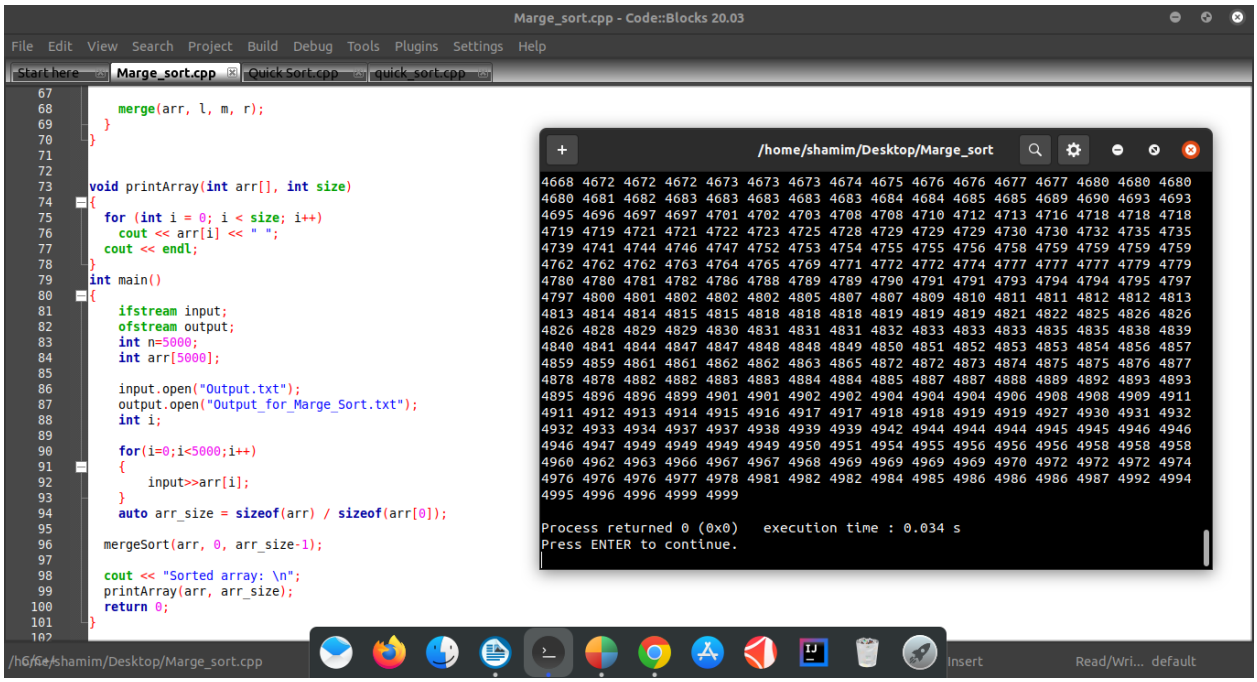
```
  printArray(arr, arr_size);
  return 0;
}
```

5.TEST RESULT / OUTPUT

Quick sort:



Marge sort :



ANALYSIS AND DISCUSSION /COMPARISON

1 In the merge sort, the array is parted into just 2 halves .
whereas
In case of quick sort, the array is parted into any ratio. There is no compulsion of dividing the array of elements into equal parts in quick sort.

2 The worst case complexity of quick sort is O(n2) as there is need of lot of comparisons in the worst condition.
whereas
In merge sort, worst case and average case has same complexities O(n log n).

3 The quick sort is internal sorting method where the data is sorted in main memory.
whereas
The merge sort is external sorting method in which the data that is to be sorted cannot be accommodated in the memory and needed auxiliary memory for sorting.

4 Merge sort is more efficient and works faster than quick sort in case of larger array size or data sets.
whereas
Quick sort is more efficient and works faster than merge sort in case of smaller array size or data sets.

So overall we can say marge sort algorithm is better to work with.