



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]

L ab 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 $\text{list}[i] > \text{pivot}$ then stop.

Step 4 - Decrement j until $\text{list}[j] < \text{pivot}$ then stop.

Step 5 - If $i < j$ then exchange $\text{list}[i]$ and $\text{list}[j]$.

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

Step 7 - Exchange the pivot element with $\text{list}[j]$ 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
    }
}
```

Merge 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.

Merge 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.

Merge 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;
}

```

Merge 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_Merge_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";
```



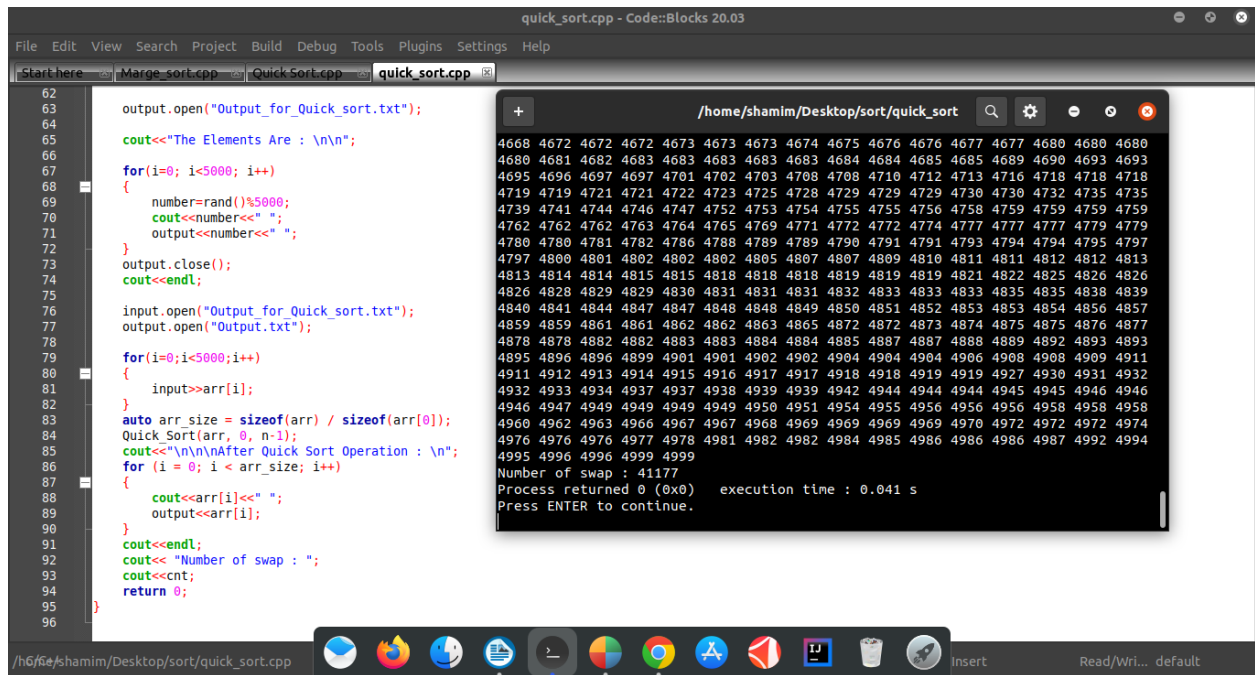
```

printArray(arr, arr_size);
return 0;
}

```

5.TEST RESULT / OUTPUT

Quick sort:



```

quick_sort.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Tools Plugins Settings Help

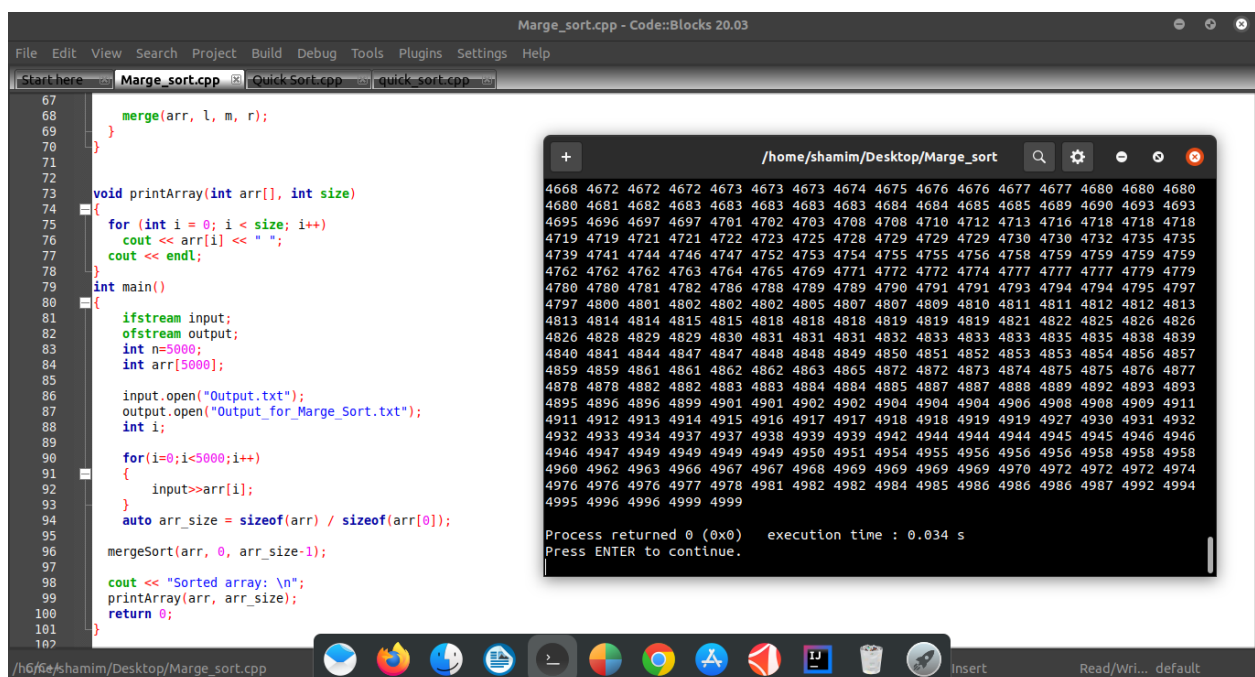
Start here Marge_sort.cpp QuickSort.cpp quick_sort.cpp

62 output.open("Output_for_Quick_sort.txt");
63
64 cout<<"The Elements Are : \n\n";
65
66 for(i=0; i<5000; i++)
67 {
68     number=rand()%5000;
69     cout<<number<<" ";
70     output<<number<<" ";
71 }
72 output.close();
73 cout<<endl;
74
75 input.open("Output_for_Quick_sort.txt");
76 output.open("Output.txt");
77
78 for(i=0; i<5000; i++)
79 {
80     input>>arr[i];
81 }
82 auto arr_size = sizeof(arr) / sizeof(arr[0]);
83 QuickSort(arr, 0, n-1);
84 cout<<"\n\nAfter Quick Sort Operation : \n\n";
85 for (i = 0; i < arr_size; i++)
86 {
87     cout<<arr[i]<<" ";
88     output<<arr[i];
89 }
90 cout<<endl;
91 cout<<"Number of swap : ";
92 cout<<cnt;
93 return 0;
94
95
96

/home/shamim/Desktop/sort/quick_sort
4668 4672 4672 4672 4673 4673 4673 4673 4674 4675 4676 4676 4677 4677 4680 4680 4680
4680 4681 4682 4683 4683 4683 4683 4683 4684 4684 4685 4685 4689 4690 4693 4693
4695 4696 4697 4697 4701 4702 4703 4708 4708 4710 4712 4713 4716 4718 4718 4718
4719 4719 4721 4721 4722 4723 4725 4728 4729 4729 4729 4730 4730 4732 4735 4735
4739 4741 4744 4746 4747 4752 4753 4754 4755 4755 4756 4758 4759 4759 4759
4762 4762 4762 4763 4764 4765 4769 4771 4772 4772 4774 4777 4777 4777 4779 4779
4780 4780 4781 4782 4786 4788 4789 4789 4790 4791 4791 4793 4794 4794 4795 4797
4797 4800 4801 4802 4802 4802 4805 4807 4807 4809 4810 4811 4811 4812 4812 4813
4813 4814 4814 4815 4815 4818 4818 4818 4819 4819 4819 4821 4822 4825 4826 4826
4826 4828 4829 4829 4830 4831 4831 4831 4832 4833 4833 4833 4835 4835 4838 4839
4840 4841 4844 4847 4847 4848 4848 4848 4849 4850 4851 4852 4853 4854 4856 4857
4859 4859 4861 4861 4862 4862 4863 4865 4872 4872 4873 4874 4875 4875 4876 4877
4878 4878 4882 4882 4883 4883 4884 4884 4885 4887 4887 4888 4889 4892 4893 4893
4895 4896 4896 4899 4901 4901 4902 4902 4904 4904 4904 4906 4908 4908 4909 4911
4911 4912 4913 4914 4915 4916 4917 4917 4918 4918 4919 4919 4927 4930 4931 4932
4932 4933 4934 4937 4937 4938 4939 4939 4942 4944 4944 4944 4945 4945 4946 4946
4946 4947 4949 4949 4949 4949 4950 4951 4954 4955 4956 4956 4956 4958 4958 4958
4960 4962 4963 4966 4967 4967 4968 4969 4969 4969 4969 4970 4972 4972 4974 4974
4976 4976 4976 4977 4978 4981 4982 4982 4984 4985 4986 4986 4987 4992 4994
4995 4996 4996 4999 4999
Number of swap : 41177
Process returned 0 (0x0)    execution time : 0.041 s
Press ENTER to continue.

```

Marge sort :



```

Marge_sort.cpp - Code::Blocks 20.03
File Edit View Search Project Build Debug Tools Plugins Settings Help

Start here Marge_sort.cpp QuickSort.cpp quick_sort.cpp

67 merge(arr, l, m, r);
68 }
69
70
71
72
73 void printArray(int arr[], int size)
74 {
75     for (int i = 0; i < size; i++)
76         cout << arr[i] << " ";
77     cout << endl;
78 }
79
80 int main()
81 {
82     ifstream input;
83     ofstream output;
84     int n=5000;
85     int arr[5000];
86
87     input.open("Output.txt");
88     output.open("Output_for_Marge_Sort.txt");
89     int i;
90
91     for(i=0; i<5000; i++)
92     {
93         input>>arr[i];
94     }
95     auto arr_size = sizeof(arr) / sizeof(arr[0]);
96
97     mergeSort(arr, 0, arr_size-1);
98
99     cout << "Sorted array: \n";
100     printArray(arr, arr_size);
101     return 0;
102 }

/home/shamim/Desktop/Marge_sort
4668 4672 4672 4672 4673 4673 4673 4673 4674 4675 4676 4676 4677 4677 4680 4680 4680
4680 4681 4682 4683 4683 4683 4683 4683 4684 4684 4685 4685 4689 4690 4693 4693
4695 4696 4697 4697 4701 4702 4703 4708 4708 4710 4712 4713 4716 4718 4718 4718
4719 4719 4721 4721 4722 4723 4725 4728 4729 4729 4729 4730 4730 4732 4735 4735
4739 4741 4744 4746 4747 4752 4753 4754 4755 4755 4756 4758 4759 4759 4759
4762 4762 4762 4763 4764 4765 4769 4771 4772 4772 4774 4777 4777 4777 4779 4779
4780 4780 4781 4782 4786 4788 4789 4789 4790 4791 4791 4793 4794 4794 4795 4797
4797 4800 4801 4802 4802 4802 4805 4807 4807 4809 4810 4811 4811 4812 4812 4813
4813 4814 4814 4815 4815 4818 4818 4818 4819 4819 4819 4821 4822 4825 4826 4826
4826 4828 4829 4829 4830 4831 4831 4831 4832 4833 4833 4833 4835 4835 4838 4839
4840 4841 4844 4847 4847 4848 4848 4849 4850 4851 4852 4853 4853 4854 4856 4857
4859 4859 4861 4861 4862 4862 4863 4865 4872 4872 4873 4874 4875 4875 4876 4877
4878 4878 4882 4882 4883 4883 4884 4884 4885 4887 4887 4888 4889 4892 4893 4893
4895 4896 4896 4899 4901 4901 4902 4902 4904 4904 4904 4906 4908 4908 4909 4911
4911 4912 4913 4914 4915 4916 4917 4917 4918 4918 4919 4919 4927 4930 4931 4932
4932 4933 4934 4937 4937 4938 4939 4939 4942 4944 4944 4944 4945 4945 4946 4946
4946 4947 4949 4949 4949 4949 4950 4951 4954 4955 4956 4956 4956 4958 4958 4958
4960 4962 4963 4966 4967 4967 4968 4969 4969 4969 4969 4970 4972 4972 4974 4974
4976 4976 4976 4977 4978 4981 4982 4982 4984 4985 4986 4986 4987 4992 4994
4995 4996 4996 4999 4999
Process returned 0 (0x0)    execution time : 0.034 s
Press ENTER to continue.

```

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 $O(n^2)$ 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 merge sort algorithm is better to work with.