PRACTICAL:3

3.1

Aim: Implement and perform analysis of worst case of Merge sort and Quick sort.Compare both algorithm.

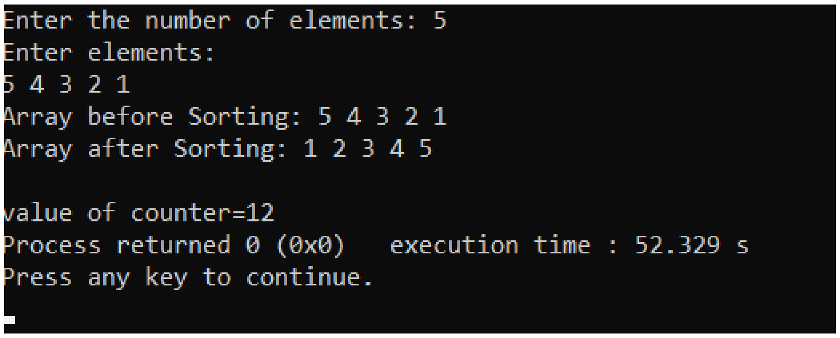
Merge sort

Program:

```
#include<iostream>

using namespace std;

int counter=0;

void swapping(int &a, int &b) {

   int temp;

   temp = a;

   a = b;

   b = temp;

}

void display(int *array, int size) {

   for(int I = 0; i<size; i++)

      cout << array[i] << " ";

   cout << endl;

}

void merge(int *array, int l, int m, int r) {

   int I, j, k, nl, nr;


   nl = m-l+1; nr = r-m;

   int larr[nl], rarr[nr];


   for(I = 0; i<nl; i++)
```

```
      larr[i] = array[l+i];
   for(j = 0; j<nr; j++)
      rarr[j] = array[m+1+j];
   I = 0; j = 0; k = l;

   while(I < nl && j<nr) {
        counter++;
      if(larr[i] <= rarr[j]) {
         array[k] = larr[i];
         i++;
      }else{
         array[k] = rarr[j];
         j++;
      }
      k++;
   }
   while(i<nl) {
       counter++;
      array[k] = larr[i];
      i++; k++;
   }
   while(j<nr) {
       counter++;
      array[k] = rarr[j];
      j++; k++;
   }
}
void mergeSort(int *array, int l, int r) {
  int m;
  if(l < r)
{
    int m = l+(r-l)/2;


    mergeSort(array, l, m);
    mergeSort(array, m+1, r);
    merge(array, l, m, r);
  }
}
int main()
 {
  int n;
  cout << “Enter the number of elements: “;
  cin >> n;
  int arr[n];
  cout << “Enter elements:” << endl;
  for(int I = 0; i<n; i++) {
    cin >> arr[i];
  }
  cout << “Array before Sorting: “;
  display(arr, n);
  mergeSort(arr, 0, n-1);
  cout << “Array after Sorting: “;
  display(arr, n);
  cout<<endl<<”value of counter=”<<counter;
}
```

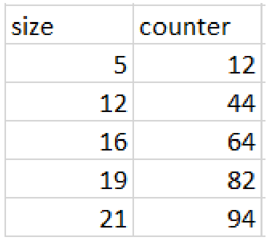
Output:



```
Enter the number of elements: 5
Enter elements:
5 4 3 2 1
Array before Sorting: 5 4 3 2 1
Array after Sorting: 1 2 3 4 5

value of counter=12
Process returned 0 (0x0)   execution time : 52.329 s
Press any key to continue.
```

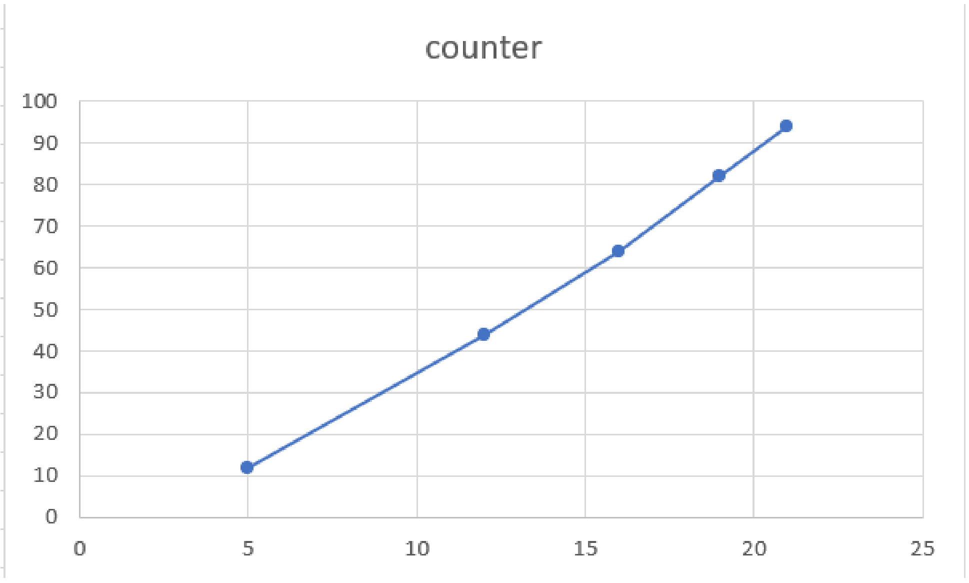
Analysis Table:

Worst Case (array is reversely sorted):



size	counter
5	12
12	44
16	64
19	82
21	94

Graph:



Quick sort

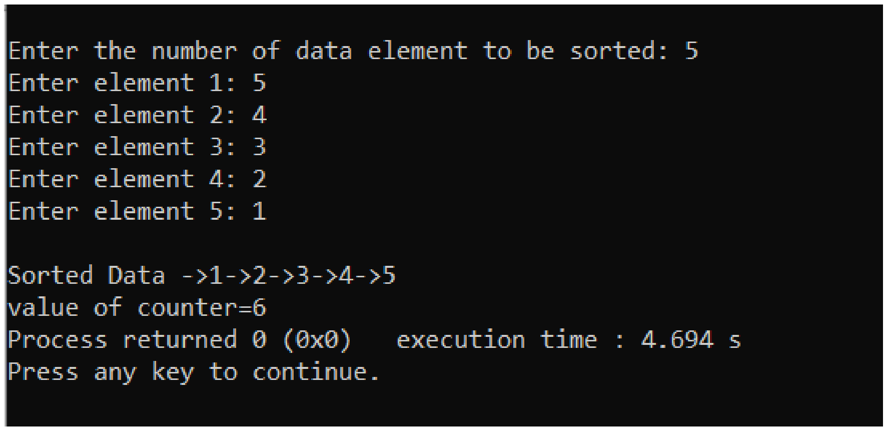
Program:

```
#include<iostream>

#include<cstdlib>

int counter=0;

using namespace std;


void swap(int *a, int *b) {

   int temp;

   temp = *a;

   *a = *b;

   *b = temp;

}


int Partition(int a[], int l, int h) {

    counter++;

   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 QuickSort(int a[], int l, int h) {
  int pindex;
  if(l < h) {
    pindex = RandomPivotPartition(a, l, h);
    QuickSort(a, l, pindex-1);
    QuickSort(a, pindex+1, h);
  }
  return 0;
}
int main() {
  int n, i;
  cout<<"\nEnter the number of data element to be sorted: ";
  cin>>n;
  int arr[n];
  for(i = 0; i < n; i++) {
    cout<<"Enter element "<<i+1<<": ";
    cin>>arr[i];
  }
  QuickSort(arr, 0, n-1);
  cout<<"\nSorted Data ";
  for (i = 0; i < n; i++)
```

```
    cout<<"->"<<arr[i];
  return 0;
  cout<<endl<<"value of counter="<<counter;
}
```

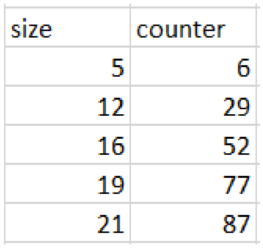
Output:



```
Enter the number of data element to be sorted: 5
Enter element 1: 5
Enter element 2: 4
Enter element 3: 3
Enter element 4: 2
Enter element 5: 1

Sorted Data ->1->2->3->4->5
value of counter=6
Process returned 0 (0x0)   execution time : 4.694 s
Press any key to continue.
```

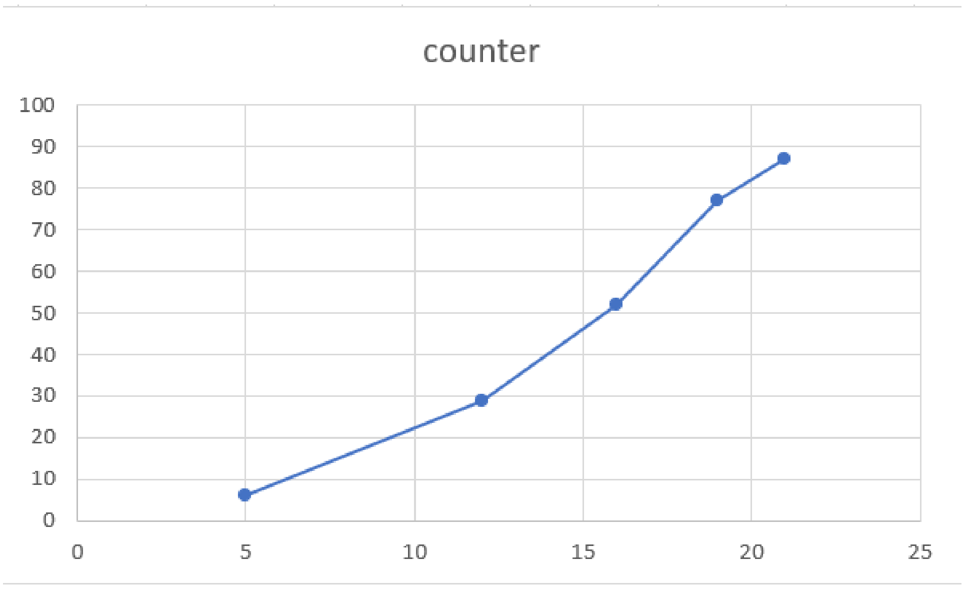
Analysis Table:

Worst Case (array is reversely sorted):



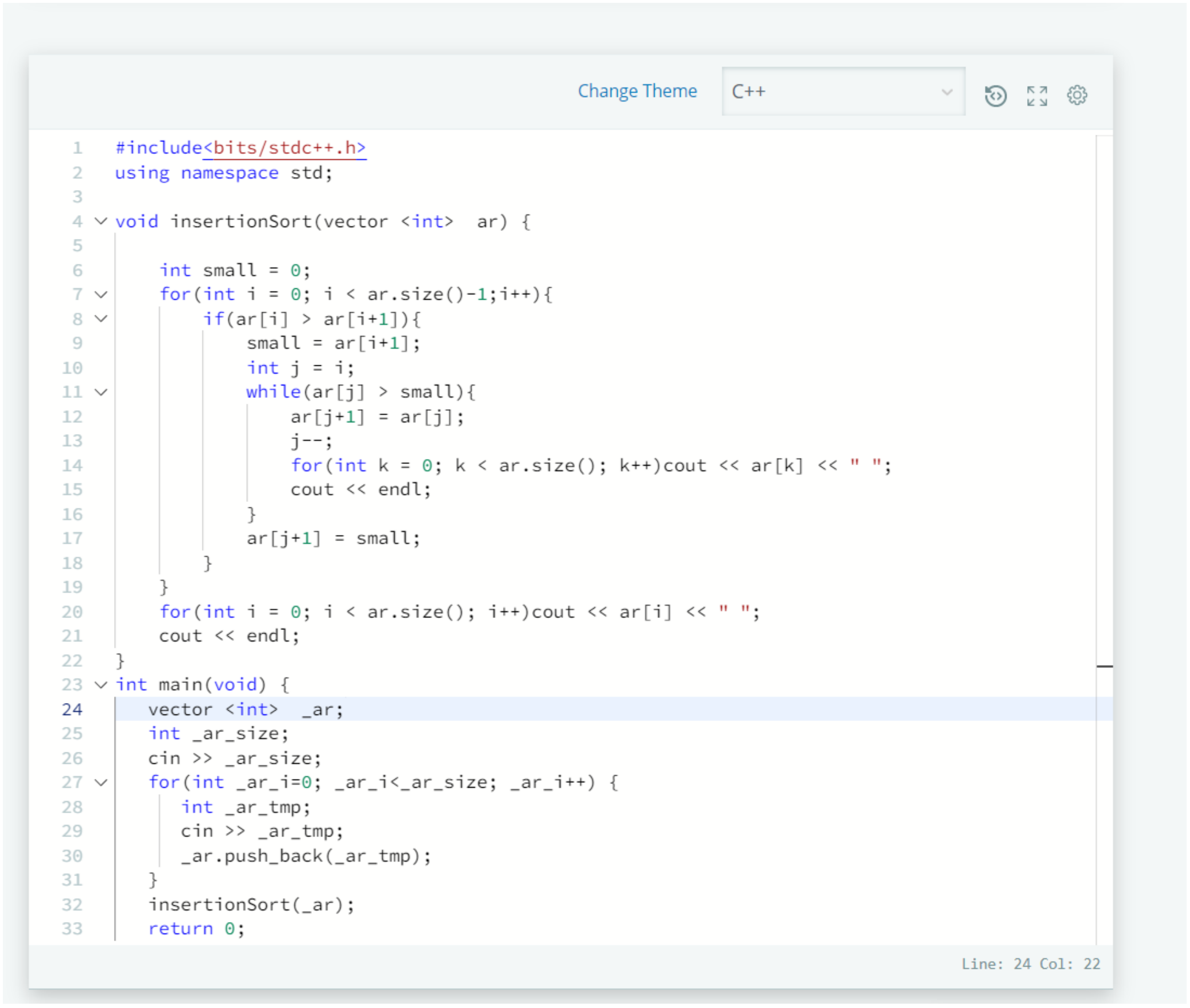
size	counter
5	6
12	29
16	52
19	77
21	87

Graph:



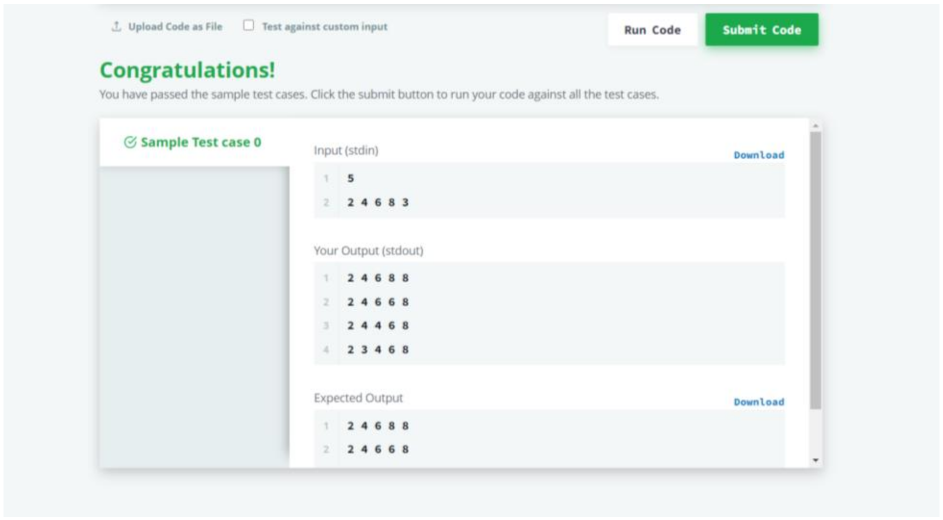
Conclusion: In this practical, we have learned about the analysis of worst case of Merge sort and Quick sort, worst case complexity of merge sort is higher than the quick sort.

Insertion sort part 1



```cpp
#include<bits/stdc++.h>
using namespace std;

void insertionSort(vector <int>  ar) {

    int small = 0;
    for(int i = 0; i < ar.size()-1;i++){
        if(ar[i] > ar[i+1]){
            small = ar[i+1];
            int j = i;
            while(ar[j] > small){
                ar[j+1] = ar[j];
                j--;
                for(int k = 0; k < ar.size(); k++)cout << ar[k] << " ";
                cout << endl;
            }
            ar[j+1] = small;
        }
    }
    for(int i = 0; i < ar.size(); i++)cout << ar[i] << " ";
    cout << endl;
}
int main(void) {
   vector <int>  _ar;
   int _ar_size;
   cin >> _ar_size;
   for(int _ar_i=0; _ar_i<_ar_size; _ar_i++) {
      int _ar_tmp;
      cin >> _ar_tmp;
      _ar.push_back(_ar_tmp);
   }
   insertionSort(_ar);
   return 0;
```

Line: 24 Col: 22



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Sample Test case 0

Input (stdin)

```
5
2 4 6 8 3
```

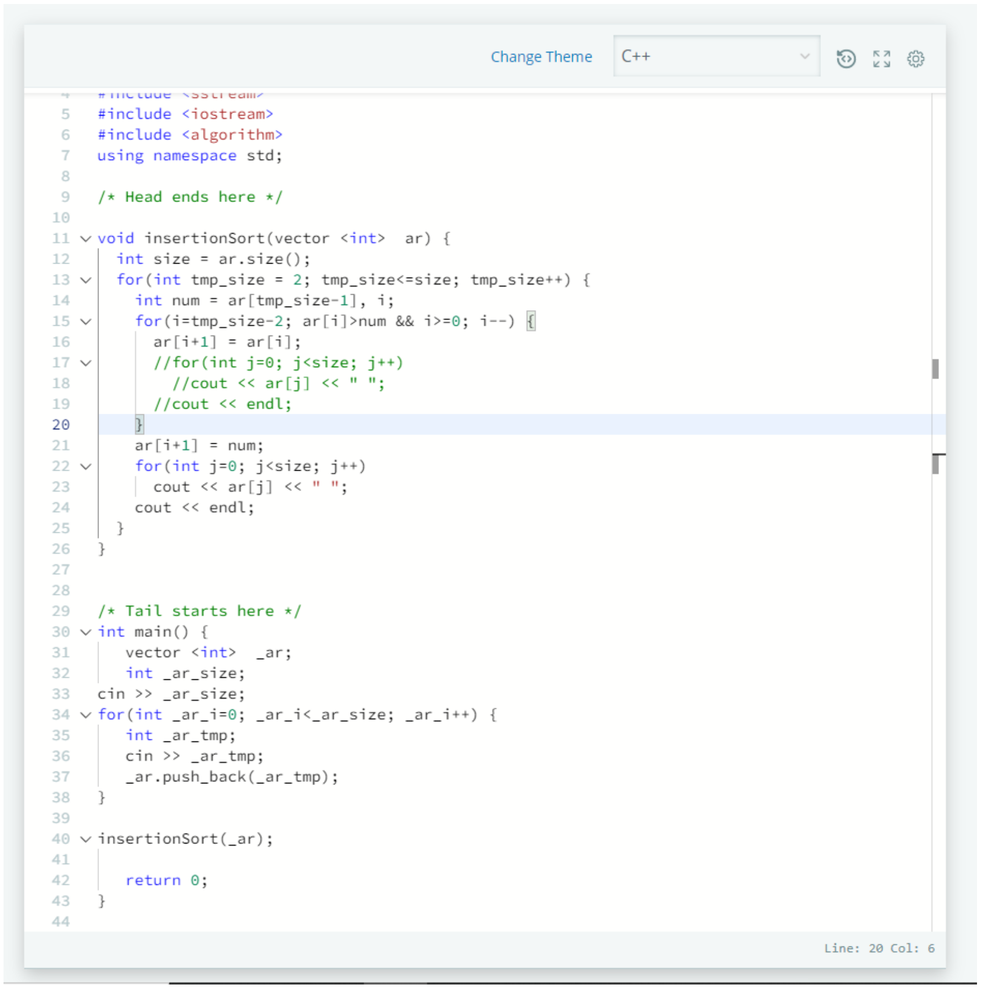
Your Output (stdout)

```
2 4 6 8 8
2 4 6 6 8
2 4 4 6 8
2 3 4 6 8
```

Expected Output

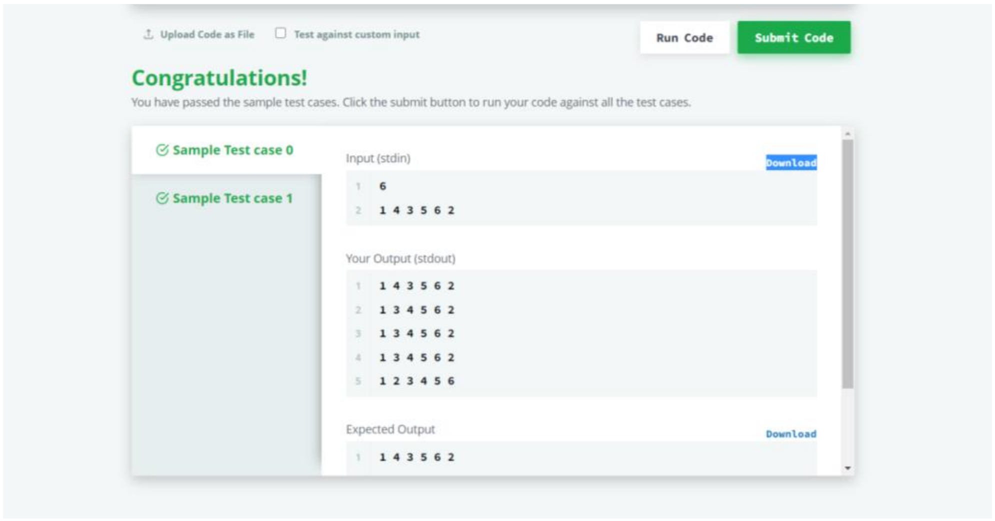
```
2 4 6 8 8
2 4 6 6 8
```

Insertion sort part 2



```cpp
#include <sstream>
#include <iostream>
#include <algorithm>
using namespace std;

/* Head ends here */

void insertionSort(vector <int>  ar) {
  int size = ar.size();
  for(int tmp_size = 2; tmp_size<=size; tmp_size++) {
    int num = ar[tmp_size-1], i;
    for(i=tmp_size-2; ar[i]>num && i>=0; i--) {
      ar[i+1] = ar[i];
      //for(int j=0; j<size; j++)
        //cout << ar[j] << " ";
      //cout << endl;
    }
    ar[i+1] = num;
    for(int j=0; j<size; j++)
      cout << ar[j] << " ";
    cout << endl;
  }
}


/* Tail starts here */
int main() {
   vector <int>  _ar;
   int _ar_size;
cin >> _ar_size;
for(int _ar_i=0; _ar_i<_ar_size; _ar_i++) {
   int _ar_tmp;
   cin >> _ar_tmp;
   _ar.push_back(_ar_tmp);
}

insertionSort(_ar);

   return 0;
}
```

Line: 20 Col: 6



Congratulations!

You have passed the sample test cases. Click the submit button to run your code against all the test cases.

Input (stdin)

```
6
1 4 3 5 6 2
```

Your Output (stdout)

```
1 4 3 5 6 2
1 3 4 5 6 2
1 3 4 5 6 2
1 3 4 5 6 2
1 2 3 4 5 6
```

Expected Output

```
1 4 3 5 6 2
```