**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);

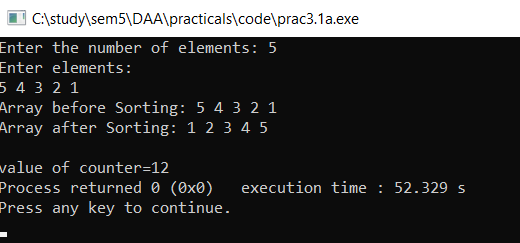
cout << “Array after Sorting: “;

display(arr, n);

cout<<endl<<”value of counter=”<<counter;

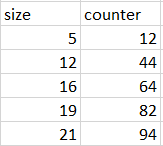
}

**Output:**

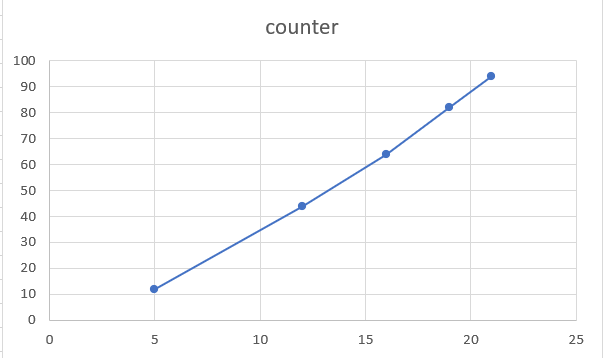
****

**Analysis Table:**

**Worst Case (array is reversely sorted):**

****

**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++)

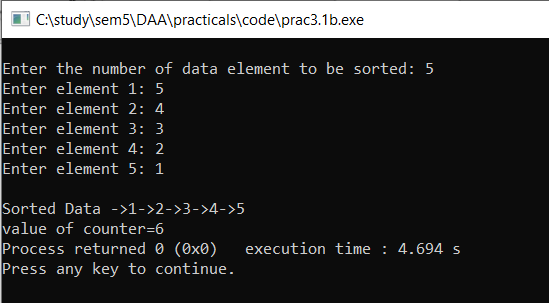
cout<<"->"<<arr[i];

return 0;

cout<<endl<<"value of counter="<<counter;

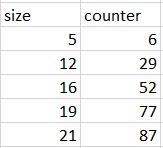
}

**Output:**

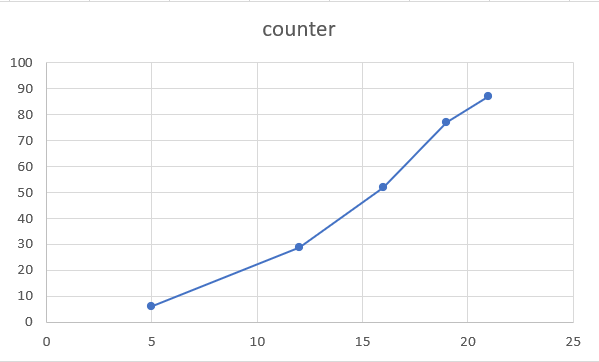
****

**Analysis Table:**

**Worst Case (array is reversely sorted):**

****

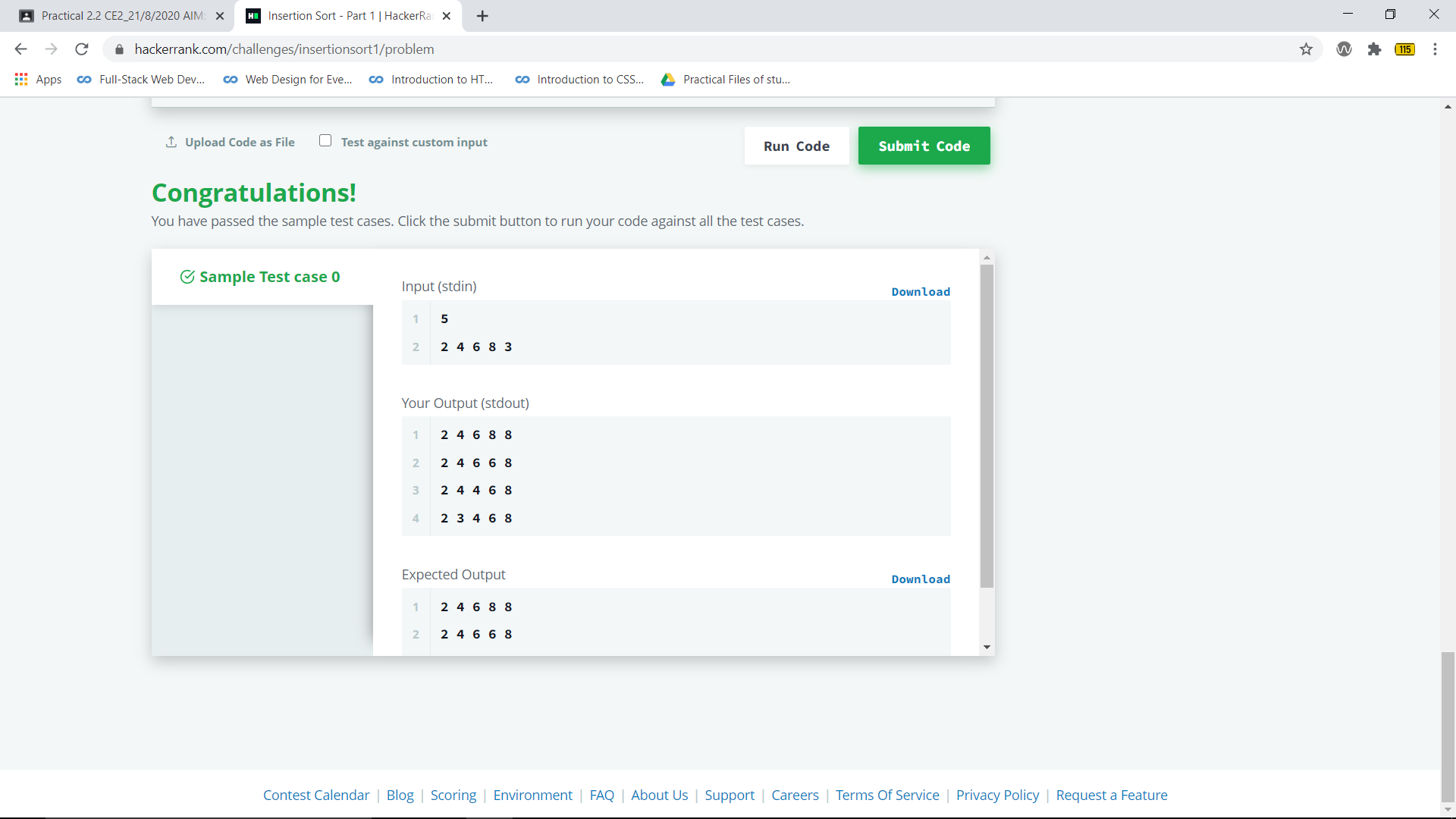
**Graph:**

****

**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**





**Insertion sort part 2**

