I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of **0** for the involved assignment for my first offense and that I will receive a grade of **"F" for the course** for any additional offense.

Name: Dhaval Kapgate   Sign:

I. Instruction Count of Insertion Sort

- The sorting code (excluding the star printing parts):

```
void InsertionSort(int no_of_input,int input[])
{
    int counter1,counter2;
    for (counter1 = 1 ; counter1 < no_of_input; counter1++)
    {
        counter2 = counter1;
        while (input[counter2] < input[counter2-1] && counter2>0)
        {
            swap(&input[counter2],&input[counter2-1]);
            counter2--;
        }
    }
}
```

- The barometer operation - swap(&input[j],&input[j-1]);
- The code contains an outer for loop running from counter1=1 to no_of_input-1 and Inner while loop that runs from counter2=counter1 to no_of_input-1.
- Using Method 2 (let i=counter1,j=counter2,n=no_of_input):

Instruction count $= \sum_{i=1}^{n-1} \sum_{j=i}^{n-1} 1$

$= \sum_{i=1}^{n-1} \sum_{j=i}^{n-1} 1$

$= \sum_{i=1}^{n-1} n - i$

$= \frac{n(n-1)}{2}$

- Now,

  $n(n-1)/2 \in O(n^2)$

  Also, $n(n-1)/2 \in \Omega(n^2)$

  Thus, instruction count for insertion sort $\in \Theta(n^2)$

II. Instruction Count of Count Sort

- The sorting code (excluding the star printing parts):

```
void CountSort(int no_of_input,int input[])
{
        int count[100],counter; //count[] keeps track of no of occurance of element
                                //in list
        for(counter=0;counter<100;counter++)
                count[counter]=0;
        for(counter=0;counter<no_of_input;counter++)
                count[input[counter]]++;
        int current=0;
        printf("\n");
        for(counter=0;counter<=99;counter++)
        {
                while(count[counter]!=0)
                {
                        input[current++]=counter;
                        count[counter]--;
                }
        }
}
```

- The barometer operations - count[input[counter]]++;
- The code contains a for loop running from counter1=0 to no_of_input-1 (and the second for loop that actually runs for no_of_input times for writing back to array).
- Using Method 2 (let i=counter,n=no_of_input):

  Instruction count = $(\sum_{i=0}^{n-1} 1)=n$
- Now,

  $n \in O(n)$

  Also, $n \in \Omega(n)$

  Thus, instruction count for count sort $\in \Theta(n)$

III. Instruction Count of Merge Sort

- The sorting code (excluding the star printing parts):

```
void Merge(int low,int middle,int high,int input[],int input_size)
{
        int temp[1000];
        int lo=low;
        int current=low;
        int mid=middle+1;
        for(;(lo<=middle)&&(mid<=high);current++)
        {
                if(input[lo]<=input[mid])
                        temp[current]=input[lo++];
                else
                        temp[current]=input[mid++];
        }
        int k;
```

```
            if(lo>middle)
            {
                        for(k=mid;k<=high;k++)
                                    temp[current++]=input[k];
            }
            else
            {
                        for(k=lo;k<=middle;k++)
                                    temp[current++]=input[k];
            }
            for(k=low;k<=high;k++)
            input[k]=temp[k];
}
void partition(int low,int high,int input[],int input_size)
{
            int middle;
            if(low<high)
            {
                        middle=(low+high)/2;
                        partition(low,middle,input,input_size);
                        partition(middle+1,high,input,input_size);
                        Merge(low,middle,high,input,input_size);
            }
}
void MergeSort(int no_of_input,int input[])
{partition(0,no_of_input-1,input,no_of_input);}
```

- The merge sort is a divide and conquer method that divides the list into two equal parts and merge them. The actual sorting takings place in the merge stage.
- Using Method 2

  In the partition function, there are two recursive calls to itself with half the list as parameters (till size of list is 1) and a merge function containing a for loop where the actual sorting takes place. So we know,

  $T(1)=1$

$T(n)$	$=2T(n/2)+n$…….(i)	(since the list is divided into 2 equal sublist and a for loop for sorting that runs n times)
	$=2[2T(n/4)+ n/2]+n$	(substituting (i) above)
	$=4T(n/4)+2n$	
	$=4[2T(n/8)+(n/4)]+2n$	(substituting (i) above)
	$=8T(n/8) + 3n$	
	$=2^k(n/2^k)+n+n+....+n$	(substituting (i) above k times where $n=2^k$ i.e. $k=\log_2 n$)
	$=nT(1)+kn$	(since $n=2^k$)
	$=n+kn$	(since $T(1)=1$)
	$=n+n\log_2 n$	(since $k=\log_2 n$)

- Now,

  $n+n\log_2 n \in O(n\log_2 n)$

  Also, $n+n\log_2 n \in \Omega(n\log_2 n)$

  Thus, instruction count for merge sort $\in \Theta(n\log_2 n)$