CS575 - Section 1
Project 1:
Time Complexity Analysis:

"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."

Submitted by: Saurabh Chaudhari

1. Counting Sort: (using method 2)

Sorting Loop:

```
for (iterator = 0; iterator < input_size; iterator++ )
    {
            temp = input[iterator];
            count[temp] = count[temp] + 1;
            if (input_size < 21)
            {
                    animate2(input,count,input_size,iteration);
                    iteration++;
            }
    }
```

This loops runs runs for N times for generating count array

Printing loop:

```
for (iterator = 0; iterator < COUNT_MAX_SIZE; iterator++)
            {
                    for (inner_i=0; Inner_i<count[iterator]; inner_i++)
                    {
                            input[j]=iterator;
                            j++;
                    }
            }
```

This loops also runs for N times and inserts the respective sorted values in original array.
And each iteration takes constant time say 1,
Total Number of instructions are N+N = 2N

$\sum_{i=0}^{n} 1 + \sum_{i=0}^{n} 1$ = ln(n) + ln(n) =2ln(n)

Therefore, Time Complexity of counting sort is of order O(n).

2. **Insertion Sort**: (using Barometer method)

For this sort,

At each iteration the input size is divided in half till the input size reaches to single element which is sorted as is.

```
for (iterator=0; iterator < input_size; iterator++)
{
	inner_i = iterator;
	while (inner_i > 0 && input[inner_i-1] > input[inner_i])
	{
		temp = input[inner_i];
		input[inner_i] = input[inner_i-1];
		input[inner_i-1] = temp;
		inner_i--;
	}
	if (input_size < 21)
	{
		fprintf(stderr, "\n\tIteration: %d\n",iterator+1);
		animate(input,input_size);
	}
}
```

Now,

Take the comparison as barometer operation,

So in worst case:

The Total number of comparisons are

total compares = 1 + 2 + 3 + … + (n-1)

but 1 + 2 + 3 + … + n = $\frac{n(n+1)}{2}$

Using this equation,

Total compares/instructions = $\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

Hence by ignoring constants, the complexity of insertion sort is O(n^2) – O(n)

Taking only the highest order, the complexity of order O(n^2).

3. Merge Sort:

```
void merge (int input[], int low, int mid, int high,int input_size)
{
   int temp[MAX_SIZE];
   int i = low, j = mid + 1, k = 0;

   while (i <= mid && j <= high)
   {
      if (input[i] <= input[j])
         temp[k++] = input[i++];
      else
         temp[k++] = input[j++];
   }
   while (i <= mid)
      temp[k++] = input[i++];

   while (j <= high)
      temp[k++] = input[j++];
   k--;
   while (k >= 0)
   {
      input[low + k] = temp[k];
      k--;
   }
   if(input_size < 21)
         {
                     fprintf(stderr, "Iteration: %d\n",globali );
                     animate(input,high);
                                 globali++;
                     }
}
void merge_sort(int input[], int low, int high,int input_size)
{
            int i,k;
   if (low < high) {
      int mid = (high + low)/2;
      merge_sort(input, low, mid,input_size);   ---------------------- N/2 Operations
      //animate(input,mid);
      merge_sort(input, mid + 1, high,input_size); ---------------------- N/2 Operations
      //animate(input,high);
      merge (input, low, mid, high,input_size); ---------------------- N Operations
   }
```

}

This method divides the input data into half and calls the same function for those recursively.

And merge function merges the data with N iterations for N data items.

Using recurrence method,

T(N) = 1 when n=1; and

T(N) = 2T(N/2) + N;

Solving this,

T(N) = 2(2T(N/4)+N)+N

= 4(2T(N/8)+N+N+N

.

.

T(N) = 2^k T($\frac{N}{2^k}$) + KN

Assume N=2^k

So, T(N) = N + T(1) + KN = N + KN

Now complexity is,

T(N) = lg(N) + N*lg(N).

Hence Merge sort has complexity $\theta(nlgn)$.