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.

Omkar Nibandhe

Time complexity using *method 2*

Insertion Sort

```
for(i = 0; i < length; i++){
        arr[i]= ( rand() % 1000) + 1;                    Constant Time
}
else{
for(i = 0; i < length; i++){
        arr[i]= ( rand() % 15+1);                        Constant Time
}
}
insert_display(arr, length);
printf("\n-------Computation.\n");
for(i = 1; i < length; i++){                             N Times
        j=i;
        while(j>0 && arr[j] < arr[j-1]){                 N Times
                //swap
                temp = arr[j];                           Constant Time
                arr[j] = arr[j-1];                       Constant Time
                arr[j-1] = temp;                         Constant Time
                j--;                                     Constant Time
        }
        if(length<20)                                    Neglecting this
                insert_display(arr, length);
}
```

Worst case Time complexity = O (N * (N))
Worst case Time complexity = O(N^2)

Counting Sort

```
for(i = 0; i < length; ++i){                             //initialize input array.
        arr[i]=( rand() % 100 );                         Constant Time

}
else{
for(i = 0; i < length; ++i){ //initialize input array.
        arr[i]=( rand() % 15+1 );                        Constant Time

}
}
for(i = 0; i < 100; i++){                                Neglecting. Constant Time
```

```
        temp[i]=0;
        finalResults[i]=0;
    }

    for(j = 0; j < length; j++){                    N Times
        temp[arr[j]] = ( temp[arr[j]] + 1 );
    }

    for(i=1;i<100;i++){                             Constant Time
        temp[i]=temp[i]+temp[i-1];

    }
    for(j=length-1; j >= 0; j-- ){                      N Times
        finalResults[ --temp[ arr[j] ] ] = arr[j];      Constant Time
            if(length<=20){                             Neglecting this part
                    for(i = 0; i < length; i++){
                    int k=0;
                    for(k=0;k<finalResults[i];k++)
                        printf("*");
                    printf("\n");
                }
            }

        }
```

Worst case time complexity = O (N + N)
Worst case time complexity = O (N)

Merge Sort

```
void merge(int arr[], int l, int m, int r)
    {
      for(i = 0; i < n1; i++)                   N time
          Left[i] = arr[l + i];
      for(j = 0; j < n2; j++)                   N time
          Right[j] = arr[m + 1+ j];             Constant Time

      while (i < n1 && j < n2)
      {
          if (Left[i] <= Right[j])
          {
            arr[k] = Left[i];                   Constant Time
            i++;                                Constant Time
          }
          else
          {
            arr[k] = Right[j];                  Constant Time
            j++;                                Constant Time
          }
```

```
            k++;
        }

                                                    // remaining at left
        while (i < n1)                              N Times
        {
            arr[k] = Left[i];                       Constant Time
            i++;                                    Constant Time
            k++;                                    Constant Time
        }

                                                    // remaining at right
        while (j < n2)                              N times
        {
            arr[k] = Right[j];                      Constant Time
            j++;                                    Constant Time
            k++;                                    Constant Time
        }
    }
```

Time Complexity for function merge() = O (n)

```
void mergeSort(int arr[], int l, int r, int length)
    {
        if (l < r)
        {
            int m = l+(r-l)/2;
            int k=0;
            mergeSort(arr, l, m, length);       O ( N ) <- doing merge for 1st half i.e n/2
            mergeSort(arr, m+1, r, length);     O ( N ) <- doing merge for 2nd half i.e n/2

            merge(arr, l, m, r);                                O ( N ) <- derived above.
                    }
            }
        }
    }
```

Time to mergeSort N/2 elements:
T(1) = 1
T(N) = DIVIDE SET INTO N/2 + DIVIDE SET INTO N/2 + MERGE STATES.
T(N) = T(N/2) + T(N/2) +N

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

BY RECURRENCE,
T(N)/N = T(N/2) / (N/2) + 1
T(N/2) / (N/2) = T (N/4) / (N/4) + 1
T(N/4) / (N/4) = T (N/8) / (N/8) + 1
.
.
.
T(2) / 2 = T(1) / 1 + 1
ADDING THESE, WE GET,
T(N)/N + T(N/2)/(N/2) + T(N/4)/(N/4) + … + T(2)/2 = T(N/2) / (N/2) + T (N/4) / (N/4) + T (N/8) / (N/8) +ln N

T(N) / N = T(1)/1 + ln N … T(1) IS CONSTANT. I.E T(1) =1
T(N) / N = 1 + ln N
T(N) = N (ln N)
THUS, **TIME COMPLEXITY = O (N ln N)**

Omkar Nibandhe oniband1@binghamton.edu Section 1