You are suggested to use the teaching servers hulk.soic.indiana.edu or tank.soic.indiana.edu for practicing C programs.

Lab 2: Sorting algorithm

1.Bubble Sort

Bubble sort is a popular sorting algorithm, by swapping the two neighboring element in reverse order repeatly.
[0][1]...[i][i+1]..[n-1][n]

Pseudo Code:

```
        input: array A
        BubbleSort(A)
        {
            for (int j = 0 ; j < length(A);j++)
                for (int i = 0 ;i < length(A)-1-j;i++)
                    if A[i] > A[i+1]                    /*for the jth time, let the jth largest element go to the tail*/
                        swap(A[i],A[i+1])
        }
```

Complexity:O(n^2)

2.Insert Sort

The basic idea of insert sort is: for each run, the ith run, i.e. find out the ith smallest (or largest) element and insert it into the ith position.

Pseudo Code:

```
        input: array A
        InsertSort(A)
        {
               int hold; /* get the element to be sorted( seeking a right place to place it)*/
               for ( int i =1; i < length(A); i++)
                   hold = array[i];                                /*get a copy of the element to be sorted*/

                    /*****find a position to insert that element*****/
                    /* if the current element, that is array[j-1] is larger than "hold", then move array[j-1] backward*/
                            /*leaving the position for "hold" or the element even smaller than "hold"*/

                    /* if the current element array[j-1]is smaller than "hold", since the elements before array[j-1] are already sorted,*/
                            /*just stop, since "hold" is already in the right place*/
                   /*repeat this process*/

                   for (int j =i ; j > 0 and array[j-1] > hold;j--)
                   {
                        array[j] = array[j-1];
                   }
                  array[j] = hold;
        }
```

3.Merge Sort

Merge sort is a classical example for divide-and-conquer algorithm. It is composed of two parts: divide the task into units and then solve & merge each of them, in recursive manner.
First take a look at the recursion part:

```
            MergeSort( array, begin, start)
            {
                mid = 0;
                if (begin < end)
                {
                    mid = (begin+end)/2;
                    MergeSort(array,begin,mid);
                    MergeSort(array,mid+1,end);
                    Merge(array,begin,mid,end);
                }
            }
```

Now take a close look at how Merge( ) works:

```
            Merge (array, begin, mid, end)
            {
            /*first part of the array*/
            begin1 = begin;
            end1   = mid;
            /*second part of the array*/
            begin2 = mid +1;
            end2   = end;

            int temp[] /* store the sorted two parts*/
            foreach p <- begin1 to end1 and q <- begin2 to end2
                if array[p] < array[q]
                    copy array[p] to temp
                else
                    copy array[q] to temp

            /*deal with the last elements still left in the first or second part of array*/
            if the first part is not empty
                copy all the left elements to temp
            else if the second part is not empty
                copy all the left elemetns to temp
```

```
copy each element in temp to array, starting from index = low ;
}
```