PROBLEM 1

To begin with we have our array values and sizes pre-determined and initialized. The first part will find find the largest number from the given array. This is so that we can next determine how many times the sorting by digits place needs to be done in order to get all the places. Finding the maximum has an order of growth of linear N as stated in the lecture chart which corresponds to loops which can be seen below.

```
for(i=0;i<9;i++)
{
   if(low<x[i])
      low=x[i];
}
```

The number of time the comparison will occur depends on the largest number and how many digits it aforementioned largest number has. This set loop will continue until the largest number becomes zero. Since it is a loop it is thus N in order of growth.

```
      while(y[j][m]!=0)
      {
         x[a]=y[j][m]; // This does the storing elements back to our given array after each sorting or comparison from the double dimensional array
         a++; // Now we continue incrementing index value of the given array
         m++;
      }
      m=0;
   }
```

With this portion we can see that the elements we sorted are stored back to our given array and than the value of the index is increased. This will allow for the next digit place to be sorted. We can see that it has to go through this entire process each time and then repeat for another digit place meaning it could have a long run time. In total there are about 6 loops with the linear order of growth of N. This increases the runtime a decent amount.

```
a=0;
```

This has the job of resetting index value of the given array to begin the process anew meaning the runtime will increase/

D)The code is not stable as we can see from this portion:

```
// while(y[j][m]!=0)
//          {
//             x[a]=y[j][m]; // This does the storing elements back to our given array after each sorting or comparison from the double dimensional array
//             a++; // Now we continue incrementing index value of the given array
//             m++;
//          }
//          m=0;
```

Since the index is moving each time the previous sort is not taken into account meaning the sorting each time has no sorting relations to the previous sort.

E)Constant extra space implies that regardless of n the code will always have the same runtime. Since my code checks each value in the array it does not currently use O(1) because adding more values would increase the runtime. In order to implement it we must do for example:

```
//int i = 0;
//    while (i<n)
//    {
//        // If this element is already processed,
//        // then nothing to do
//        if (arr[i] <= 0)
//        {
//            i++;
//            continue;
```

Initializing i as zero and then finding elements of the array corresponding to a given index would mean all elements corresponding to that index would be sorted accordingly rather than comparisons in between them being made. Thus the runtime would be the same no matter the size of the array.