CS 100: Section Assignment 9

(For the week of April 5)

Section assignments are discussed in section and are not submitted for grading. They relate to recent lecture topics and usually to the current Programming Assignment. Prelim questions are based on Section Assignments, Programming Assignments, and Lecture examples.

1. Suppose `a` is an initialized array of integers in the main method. Write a fragment that prints the median value.

2. Suppose `a`, `b`, and `c` are three initialized arrays in the main method. Assume that they are all sorted. Write a fragment that produces a single sorted array made up of their values. (I.e., merge the three arrays.)

3. Implement the following method in the class `SearchNSort` :

```
// Yields a copy of the array a.
public static int[] copy(int[]a)
```

4. Rewrite `LinSearch` so that it returns the index of the last occurrence of `x` in the array `a.`

```
public class SearchNSort
{
   // Yields the index of the first occurrence of x in the array a.
   // Yields -1 if no array value equals x.
    public static int LinSearch(int[]a,int x)
    {
       int n = a.length;
       int k=0;
       while (k<a.length && x!=a[k])
          k++;
       if (k==a.length)
          return -1;
       else
          return k;
    }

    // Yields -1 if no array value equals x.
    public static int BinSearch(int[] a,int x)
    {
       int n = a.length;
       int L = 0;
       int R = n-1;
       int m;
       while (L<=R)
       {
          m = (L+R)/2;   // The midpoint index.
          if (x==a[m])
             return m;
          if (x>a[m])
             // x is not in a[0..m]
             L = m+1;
          else
             // x is not in a[m..n-1]
             R = m-1;
       }
       return -1;
    }
```

```
// Permutes the values in a so that a[0]<=a[1]<=...<=a[n-1]
// where n = a.length
 public static void sort(int[] a)
 {
    int n = a.length;
    int temp; // For swapping array elements
    int j;
    for (int k=1;k<n;k++)
    {
       // a[0..k-1] is sorted. Now sort a[0..k] by
       // "pushing" it left as long as
       // it is smaller than its "left neighbor".
       j=k;
       while(j>=1 && a[j]<a[j-1])
       {
          temp = a[j-1]; a[j-1] = a[j]; a[j] = temp;
          j=j-1;
       }
    }
 }

// Permutes the values in a so that a[0]<=a[1]<=...<=a[n-1]
// where n = a.length
 public static void sort(String[] a)
 {
    int n = a.length;
    String temp; // For swapping array elements.
    int j;
    for (int k=1;k<n;k++)
    {
       // a[0..k-1] is sorted. Now sort a[0..k] by
       // "pushing" it left as long as
       // it is smaller than its "left neighbor".
       j=k;
       while(j>=1 && a[j].compareTo(a[j-1])<0)

       {
          temp = a[j-1]; a[j-1] = a[j]; a[j] = temp;
          j=j-1;
       }
    }
 }
```

```
   // Permutes the values in a so that a[0]<=a[1]<=...<=a[n-1]
   // where n = a.length
   public static void MergeSort(int[] a)
   {
      int n = a.length;
      if (n==1)
         // Nothing to do
         return;
      else
      {
         // Split the array in half
         int m = n/2;
         int[] aLeft  = new int[m];
         int[] aRight = new int[m];
         for(int i=0;i<m;i++)
         {
            aLeft[i] =  a[i];
            aRight[i] = a[i+m];
         }

         // Sort the left and right halves
         MergeSort(aLeft);
         MergeSort(aRight);

         // Merge the results
         int iLeft=0;
         int iRight=0;
         for (int k=0;k<n;k++)
            if (iLeft==m)                             { a[k] = aRight[iRight]; iRight++;}
            else if (iRight==m)                       { a[k] = aLeft[iLeft];   iLeft++; }
            else if (aLeft[iLeft] <= aRight[iRight])  { a[k] = aLeft[iLeft];   iLeft++;}
            else                                      { a[k] = aRight[iRight]; iRight++;}
       }

    }

}
```