Unit Six Lecture Two:

Topic 1: The Merge Sort

The idea behind the merge sort is to split the array in half, sort each half, and merge the results together. As it turns out, if the computer keeps dividing the array into smaller and smaller subarrays (using recursion), sorting each half and merging them back together, it visits dramatically fewer array elements than the selection sort requires.

As a matter of fact, the Big O notation for the merge sort is O(n*log(n)). This means that when attempting to sort an array with n elements you will visit approximately n*log(n) array elements to get the job done.

Example:

Number of Elements	Processing Time
10,000	3.5 sec
1,000,000	9 min

For more information on how the Big O notation was determined for the Merge Sort, see pages 715 - 717 in your text.

Because the coding of a merge sort is more complicated than a selection or insertion sort, it is placed in its own class rather than simply in a method. Below you will find the code for a MergeSorter class (ascending order).

```
public class MergeSorter
{
     private int[] a;                   // instance variable

     public MergeSorter(int[] anArray)  // parametric constructor
     {
          a = anArray;
     }

     public void sort()
     {
          if (a.length <= 1) return;   // stops infinite recursion

          int[] first = new int[a.length/2];
          int[] second = new int[a.length - first.length];

          System.arraycopy(a,0,first,0,first.length);
          System.arraycopy(a,first.length,second,0,second.length);
```

```
                MergeSorter firstSorter = new MergeSorter(first);
                MergeSorter secondSorter = new MergeSorter(second);

                firstSorter.sort();
                secondSorter.sort();
                merge(first,second);
          }

          private void merge(int[] first, int[] second)
          {
                int i = 0;
                int j = 0;
                int k = 0;

                while (i < first.length && j < second.length)
                {
                      if (first[i] < second[j])
                      {
                            a[k] = first[i];
                            i++;
                      }
                      else
                      {
                            a[k] = second[j];
                            j++;
                      }
                      k++;
                }

                System.arraycopy(first,i,a,k,first.length-i);
                System.arraycopy(second,j,a,k,second.length-j);
          }
    }
```

Topic 2: The QuickSort

The idea behind the QuickSort is to repeatedly (using recursion) select a pivot in the array and rearrange the array elements so that all numbers less than the pivot are placed to the left of the pivot and all numbers greater than the pivot are placed to its right.

The QuickSort also has a Big O notation of O(n*log(n)), but because it doesn't use temporary arrays it runs faster than the merge sort in most cases.

Because the coding of a QuickSort is more complicated than a selection or insertion sort, it is placed in its own class rather than simply in a method. Below you will find the code for a QuickSorter class (ascending order).

```
public class QuickSorter
{
      private int[] a;                    // instance variable

      public QuickSorter(int[] anArray)  // parametric constructor
      {
            a = anArray;
      }

      public void sort(int from, int to)
      {
            if (from >= to)  return;     // stops infinite recursion

            int p = partition(from, to);
            sort(from, p);
            sort(p+1, to);
      }

      private int partition(int from, int to)
      {
            int temp;
            int pivot = a[from];
            int i = from - 1;
            int j = to + 1;

            while (i < j)
            {
                  i++;
                  while (a[i] < pivot)
                        i++;

                  j--;
                  while (a[j] > pivot)
                        j--;

                  if (i < j)
                  {
                        temp = a[i];
                        a[i] = a[j];
                        a[j] = temp;
                  }
            }
            return j;
      }
}
```

Unit 6 Assignment 2: Comparing the Merge Sort & QuickSort