CS 202 Lab/Programming Assignment 7

Due: Before the Final Exam on May 5, 2015

In this assignment, you will download the Lab7.java program from the Kodiak, and enhance it to calculate the number of comparisons, number of moves for each sorting algorithm we have learned in class and find the time it takes to run each algorithm. Specifically,

1. Download Lab7.java from Kodiak: http://Kodiak.wne.edu.

```
public class Lab7
{
     public static void main(String[] args)
     {
          final int MAX_SIZE = 50000;
          Integer[][]  x = new Integer[3][MAX_SIZE];
                           // row 0 is a sorted array
                           // row 1 is a reversely sorted array
                           // row 2 is a random array
          Integer[]    a = new Integer[MAX_SIZE];
          ...


          generateArrayElements(x);

          for each ith type of the three types of array
          {
               print appropriate heading;

               copy ith row of two-dimensional array x to array a
               selectionSort(a);
               print statistics;

               copy ith row of two-dimensional array x to array a
               bubbleSort(a);
               print statistics;

               ...

          }

     }
```

```
    private static void generateArrayElements(Integer[][] x)
    {
        int n = x[0].length;
        ...

        // first generate a sorted array that contains Integer
        // objects of values 1, 2, ..., n
        ...

        // generate a reversely sorted array that contains
        // Integer objects of values n, n-1, ..., 2, 1
        ...

        // finally generate an array that contains Integer objects
        // of values 1 through n which are randomly distributed in
        // the array
        // But, how????  (See generateArrayElements method of
        //                 Generic QuickSort example.)
        ...

    }



    private static void selectionSort(int[] a)
    {
        int    minIndex, min;
        int    n = a.length;
        ...

        for (int i = 0; i < n-1; ++i)
        {
            minIndex = i;

            ...
        }
    }



    private static void bubbleSort(int[] a)
    {
        int    n = a.length;
        ...

        for (int i = 0; i < n-1; ++i)
        {
            ...
        }
    }


    // ... other sorting algorithms
}
```

2. Note that in Lab7.java, Quick Sort, Heap Sort, Merge Sort, and Shell Sort are generic methods that can sort on arrays of objects that implement the **Comparable** interface and define the **compareTo** method. You need to convert all other sorting algorithms to generic methods using the same technique. Your program will sort arrays of **Integer** objects.

3. To call each of the generic sorting algorithms, you should pass an array of **Integer** objects. For testing the generic sorting algorithms on various types of arrays, you will need to create a **two-dimensional** array of Integer objects. The first row contains a **sorted array**, and the second row contains a **reversely sorted array**. The third row of the two-dimensional array contains Integer(1), Integer(2), ..., Integer(n) that are **randomly distributed** in the array. You will need to use a random number generator. Java provides a random number generator: **Math.random()**. Note that Math.random() returns a random number (a real number) between 0 and 1,

   $$0 \leq \text{Math.random()} < 1$$

   So, if p is an integer, then

   $$0 \leq p * \text{Math.random()} < p$$

   and

   $$(\text{int}) (p * \text{Math.random()})$$

   would give an integer between 0 and p – 1. That is,

   $$0 \leq (\text{int}) (p * \text{Math.random()}) \leq p - 1$$

   See generateArrayElements method of Generic QuickSort example for details of constructing such a randomly distributed array.

4. How do you pass statistics (including the number of comparisons, number of moves and the running time) between the main program and a sorting algorithm? Since Java only supports pass-by-value, you need to use an object to pass such information. So, declare a class with 3 data members of the **long** data type. Create an object of this class in the main program and pass it to each sorting algorithm.

   In each sorting algorithm, you need to **add code** to increment the number of comparisons and the number of moves at appropriate places. (Note that in some algorithms, this may not be very simple.)

5. How do you keep track of the time used by an algorithm? You may use the **System.currentTimeMillis()** function, which returns the current time as a **long** integer of milliseconds since January 1, 1970, 00:00:00 GMT. So this is what you will need to do in each sorting algorithm:

```
public static <T extends Comparable<? super T>>
    void quickSort(T[] a,
                   a second parameter to pass sorting statistics)
     {
          int    n = a.length;
          ...

          initialize sorting statistics;
          what is the current time now?

          qsort(a, 0, n-1, ...);

          what is the current time now?
          Calculate the elapse time;
     }
```

7. **Format** and print statistics, such as

Sorted array:

Algorithms	Number of Comparisons	Number of Moves	Running Time (Milliseconds)
Selection Sort	499500	2997	0
Bubble Sort	499500	0	0
...			

Reversely Sorted array:

Algorithms	Number of Comparisons	Number of Moves	Running Time (Milliseconds)
Selection Sort	499500	2997	0
Bubble Sort	499500	1498500	50
...			

Random array:

Algorithms	Number of Comparisons	Number of Moves	Running Time (Milliseconds)
Selection Sort	499500	2997	50
Bubble Sort	499500	742521	0
...			

Note that to format the output, you may use the **printf** method.