Lab 03

Simple Sorting Methods (Buble Sort, Selection Sort, Insertion Sort)

Revised by Tran Thanh Tung

1. **Objectives**
   a. Know how, in reality, three simple sorting methods work.
   b. Know how to use analysis tool to compare performance of sorting algorithms

2. **Problem statement**
   a. Write a Java program to measure time (in seconds) needed for each simple sorting algorithms applying on ***the same random array*** of integer values. Sizes of arrays are accordingly 10000, 15000, 20000, 25000, 30000, 35000, 40000, 45000 and 50000. Each time, you write down the measured time in following table.

Table 1 - Experiment 1: Simple sorting on random data

	Bubble Sort	Selection Sort	Insertion Sort
10000			
15000			
20000			
25000			
30000			
35000			
40000			
45000			
50000			

   b. Write some code to measure time (in seconds) needed for each simple sorting algorithms applying on ***Inversely sorted*** and ***Already-sorted order*** integer arrays of **10000** elements.

Table 2 - Experiment 2: Simple sorting in special cases

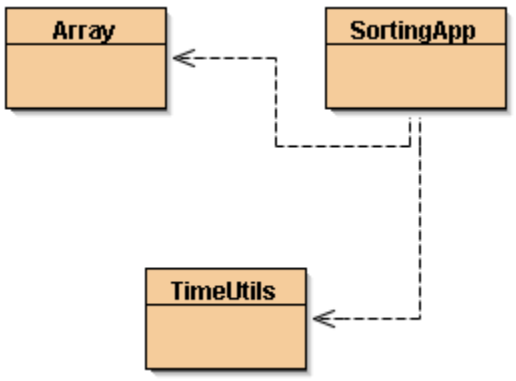
	Bubble Sort	Selection Sort	Insertion Sort
Inverse order			
Already-sorted			

   c. Based on above table, give your comments on real complexity of the three simple sorting algorithms. (Remember, all of them are O(n^2) in theory).

3. **Instruction:** (Follow instructions step-by-step)
   a. Take a look at sample source files and read it carefully. There are three files:
      i. *Array.java*: contains class *Array*,
      ii. *TimeUtils.java*: contains class *TimeUtils* and
      iii. *SortingApp.java*: contains *main()* method.

	Class Name	Typical Method	Notes
Array.java	*Array*	public void	Create an array of random long-integer

		randomInit(int numElements)	values with size specified by numElements.
		public void *bubbleSort*()	Bubble sorting method.
		public void *selectionSort*()	Selection sorting method. You will add its code by yourself (textbook).
		public void *insertionSort*()	Insertion sorting method. You will add its code by yourself (textbook).
TimeUtils .java	*TimeUtils*	public static long *now*()	Get current time in milli-seconds since epoch.
SortingApp .java	*SortingApp*	public static void *main*(String[] args)	Main method used to measure time.



Class Diagram

b. The Array class is highly similar with the one which you have learnt in text book (Chapter 2). It now has some more methods. The *randomInit()* method is used to generate an array of random long integer values.

```
public void randomInit(int numElements)
{    // create a random machine
    Random aRandom = new Random();
    nElems = numElements;
    for (int i = 0 ; i < nElems ; i++)
    {
     // assign a random long integer value
     // to current element of the array
        a[i] = aRandom.nextLong() % 100000000;
    }
}
```

The method uses *Random* class in package *java.util*, so you need to import it first before using. After creating a random machine, you can generate random long integer values by calling its *nextLong()* method. You can refer to http://download.oracle.com/javase/6/docs/api/ for more information about class *Random*.

c. To generate an array of random long integer values, you first create an *Array* object and then call its *randomInit()* method as following (in *SortingApp.java*):

```
int maxSize = 10000;        // array size
Array arr;                  // reference to array
arr = new Array(maxSize);   // create the array
arr.randomInit(maxSize);    // generate random array's elements
```

d. The above code segment creates an array of 10000 random elements. Then, you need to apply some sorting methods on it. To call it, simply use the *Array* object (named *arr* as above) and call its *bubbleSort()* method (newly added code segment is in **bold**):

```
int maxSize = 10000;         // array size
Array arr;                   // reference to array
arr = new Array(maxSize);    // create the array
arr.randomInit(maxSize);     // generate random array's elements
arr.bubbleSort();            // bubble sort them
```

e. Now you can sort an array of 10000 random values. You will need to measure time (in milli-seconds) which bubble sorting method takes to run. The static method *now* in class *TimeUtils* is used for this purpose. It returns number of milli-seconds since epoch. You can refer to http://download.oracle.com/javase/6/docs/api/ for more information about how to use *Calendar* class.

```
public static long now()
  {
    Calendar cal = Calendar.getInstance();
    Date currentDate = cal.getTime();
    return currentDate.getTime();
  }
```

f. The code in *SortingApp* is now modified to measure time needed by *Bubble* sorting method.

```
int maxSize = 10000;        // array size
Array arr;                  // reference to array
arr = new Array(maxSize);   // create the array
arr.randomInit(maxSize);    // generate random array's elements
long startTime, endTime;

// get time just before running sorting
startTime = TimeUtils.now();

arr.bubbleSort();           // bubble sort them

// get time just after running sorting
endTime = TimeUtils.now();

// time needed in milli-seconds
duration = endTime - startTime;
System.out.print("Time " + duration + "ms");
```

g. Until now, hopefully you can understand the way we measure running time of *Bubble* sort. Your task now is to write *selectionSort()* and *insertionSort()* method in class *Array*. If you forget them, you can refer to Chapter 3 in textbook.

   Note: because after sorting by bubble sort, the array is changed. Thus you need to create a copy of the original array to apply for each sorting method. Copy constructor Array (Array oriArray) can be used to create a copy of the array.

   To check algorithms code correct or not, you can use *display()* method. It prints out content of an array.

h. Play around your program with different *maxSize* values and record their running time into a table as mentioned above. You can use MS Excel to store running times.

i. Ask your lab advisor how to produce a chart from your table.

j. Give some comments about running times of three simple sorting methods.

k. *Advance*: You notice that the running times depend on the random generated array. To eliminate it, you can run each sorting method several times (5, for example) and then take average. It's your task!

l. Finish.

4. Submission

You have to show your code and submit to lab advisor table 1, table 2 in handwriting.