# Assignment 3
**(Full Score: 40 points)**

(**Due by 2/11/2011/Friday Midnight at Moodle**)

| | |
|---|---|
| Your name: | Score: |

Create a framework for the experimental testing of the Java code performance.

Requirements:

1. Create a Java class that has two methods:
   a. An *int findMax(int[] arr)* method that searches the array *arr* of type *int* for the largest value and returns the index of the first largest element in this array.
   b. A *main()* method to test the performance of the *findMax(int[] arr)* method.

2. Test method *findMax*(*int[] arr*) with different input array sizes, which will be provided by the user prompted for the input (suggested sizes would be 10, 100, 500, 1000, 5000, 10000, 50000, 100000, … and 88888888. You may modify these values or add new ones based on the actual running times of your program).

3. Use the Java API class *java.util.Random*. This class has a method *nextInt(n)* that returns "a pseudorandom, uniformly distributed `int` value between *0* (inclusive) and" *n* (exclusive), e.g. each call to *nextInt(5)* returns values one of the following randomly selected values: *0, 1, 2, 3, 4*. Use the calls to the *nextInt(n)* method in a loop to initialize all your test arrays with pseudorandom integers. To make sure that these integers have both positive and negative values and are uniformly distributed, before assigning a value to the array element subtract *n/2* from the *nextInt(n)* output. The value of *n* can be any positive integer – the suggested value is 10000000.

4. Use the Java API static method *System.nanoTime()* to record the execution time of the method *findMax(int[] arr)* invocation (this time duration does ***not*** cover the input and output executions)

5. Your program must output: (1). The size of the array (i.e., the number of randomly generated numbers), (2). The running time in finding the maximum number, and (3). The result (in double) of time divided by the array size.

6. Perform the Big-O analysis: Use the experimental results from the item 5 above (you must make at least 15 runs for different array sizes and record the results,

preferably in a table), to formulate in the assignment report in a Word/PDF file your conclusions on ***whether the performance of the code tested is consistent with the theoretical Big-O analysis and explain why***.

Hints:

1. To measure how long some code takes to execute, use the algorithm below:
   *long startTime = System.nanoTime();*
   *// the code being tested*
   *long endTime = System.nanoTime() – startTime;*

2. The method *nextInt(int n)* returns a pseudorandom, uniformly distributed *int* value between *0* (inclusive) and the specified value of *n* (exclusive), drawn from this random number generator's sequence. Every time when you create a new *int* array for testing, use a loop to initialize the array elements with the pseudorandom values returned by consecutive calls to the *nextInt()* method.

3. Use Java coding guidelines when naming your identifiers and creating class fields. The fields should be *private* or *protected* and appropriate accessor and mutator methods should be provided.

4. Use NetBeans to create a new project, a new class that implements the lab assignment and to compile, run, test, and debug (if necessary) your code.

5. Save your work regularly, especially at the end of each class.

6. Keep a detailed record of all steps performed.

**Submission:**
Submit your project ***source code and the Word/PDF file*** (results and analysis of the algorithm in big-O) in a compressed file to **Moodle by 2/11/Friday-Midnight**!