ASSIGNMENT 3

COMP-202A, Fall 2013, All Sections

Due: November 8th, 2013 (23:59)

Please read the entire pdf before starting.

You must do this assignment individually and, unless otherwise specified, you must follow all the general instructions and regulations for assignments. Graders have the discretion to deduct up to 10% of the value of this assignment for deviations from the general instructions and regulations. These regulations are posted on the course website. Be sure to read them before starting.

Part 1:	0 points
Part 2, Question 1:	40 points
Part 2, Question 2:	30 points
Part 2, Question 3:	30 points
	100 points total

It is very important that you follow the directions as closely as possible. The directions, while perhaps tedious, are designed to make it as easy as possible for the TAs to mark the assignments by letting them run your assignment through automated tests. While these tests will not determine your entire grade, it will speed up the process significantly, which will allow the TAs to provide better feedback and not waste time on administrative details. Plus, if the TA is in a good mood while he or she is grading, then that increases the chance of them giving out partial marks :)

Part 1 (0 points): Warm-up

Do **NOT** *submit this part, as it will not be graded. However, doing these exercises might help you to do the second part of the assignment, which will be graded. If you have difficulties with the questions of Part 1, then we suggest that you consult the TAs during their office hours; they can help you and work with you through the warm-up questions.*

Warm-up Question 1 (0 points)

Write a method `printArray` which takes as input an `int[]` and prints the result to the screen. To display it nicely, you should print a { at the beginning, a } at the end, and commas and spaces between all the values. For example, if the array has the 3 elements 1,2, and 3, your code should print
`{ 1, 2, 3 }` .

Warm-up Question 2 (0 points)

Write a method `generateMultiples` which takes as input an int `n` and an int `count` and produces an array of int filled with the first `count` multiples of `n`. For example, if `n` is 3 and `count` is 5, your method should return the array {`3,6,9,12,15`}

Warm-up Question 3 (0 points)

Using the `Random` class in the package `java.util`, write the program rock, paper, scissor. To do this, you should declare a variable of type `Random` by writing:

```
Random r = new Random();
```

Then you can generate "random" integers between 0 and n (including 0 but not including n) by writing: `r.nextInt(n)` where n is a positive integer. For example, `r.nextInt(3)` would generate a random number of 0,1, or 2.

Your game should work by asking the user to specify rock paper or scissor, based on what number they type. The computer should then randomly pick one of these 3 numbers and you can determine who wins. Remember that rock beats scissor, scissor beats paper, and paper beats rock. For full details on the game, see `http://en.wikipedia.org/wiki/Rock-paper-scissors` and `http://www.worldrps.com/`.

There is a description of more detail about using the Random class below. If you are trying to debug your program, and you want to get the same "random" results every time, you can write: `Random r = new Random(1)` instead when you declare the variable. By writing the 1, you will make the numbers the same every time but they won't follow a noticeable pattern.

Warm-up Question 4 (0 points)
Write a method `scalarMultiply` which takes as input a `double[] array`, and a `double scale`, and returns `void`. The method should modify the input array by multiplying each value in the array by `scale`. Question to consider: When we modify the input array, do we actually modify the value of the variable `array`?

Warm-up Question 5 (0 points)
Write a method `deleteElement` which takes as input an `int[]` and an `int target` and deletes all occurrences of `target` from the array. The method should return the new `int[]`. Question to consider: Why is it that we have to return an array and can't simply change the input parameter array?

Warm-up Question 6 (0 points)
Write the same method, except this time it should take as input a `String[]` and a `String`. What is different about this than the previous method? (Hint: Remember that `String` is a reference type.)

Warm-up Question 7 (0 points)
Write a class `Vector`. A `Vector` should consist of three `private` properties of type double: x,y, and z. You should add to your class a constructor which takes as input 3 doubles. These doubles should be assigned to x,y, and z. You should then write methods `getX()`, `getY()`, `getZ()`, `setX()`, `setY()`, and `setZ()` which allow you to get and set the values of the vector.

Warm-up Question 8 (0 points)
Add to your `Vector` class a method calculateMagnitude() which returns a double representing the magnitude of the vector. The magnitude can be computed by taking

$$\sqrt{x^2 + y^2 + z^2}$$

Part 2

The questions in this part of the assignment will be graded.

In these questions you will do the following. First, you will write several methods that use arrays. Then you will write a class that represents a StopWatch which can be used for timing your programs. Finally, you will perform experiments using your StopWatch class in which you compare the speed of your methods. In doing so, we will test Barack Obama's knowledge of computer science theory.

Question 1: Array methods (40 points)
The following code should be put in a class `ArrayUtilities`

Write the following five methods: You may NOT use any library methods for these methods.

- `public static boolean linearSearch(int[] array, int target)` : This method should take as input an array of `int` as well as an `int`. It should return `true` if the element `target` is present in `array`. It should do so by examining each element of `array` one at a time sequentially starting

from the beginning of the array until the end. The method should return `false` if the element is not present.

- `public static boolean binarySearch(int[] array, int target)` : This method should take as input a *sorted* (in increasing order) array of `int` as well as an `int`. Like the above method, it should return `true` or `false` based on whether the element `target` is present in `array` or not. However, because the array is sorted, you can search the array in a more clever way, which is similar to how one would search through a dictionary for a word.

  1. First, create two variables, `left` and `right`. Set `left` equal to 0, and `right` equal to `array.length - 1`. These two variables will represent the lower and upper bounds of the part of the array that the value `target` COULD be in.
  2. Next, create a third variable `middle` which is the arithmetic average of `left` and `right`.
  3. Check whether the array at the position `middle` is greater than, less than, or equal to the `target` element.
  4. If `array[middle]` equals the target, then success! You found it, and can return `true`.
  5. If `array[middle]` is less than the target, then you KNOW that since the array is sorted, the target is NOT in the first half of the array. So you know that if `target` appears in the array, it appears at an index of at least `middle + 1`. (Much like if you are looking for the word *programmer* in a dictionary, and you flip to the page for L, you immediately realize that the word is LATER in the dictionary if it appears at all.) Update the variable `left` accordingly.
  6. If `array[middle]` is greater than the target, you similarly know that target is not in the second half of the array and that it occurs at an index of *at most* `middle - 1`. Update the variable `right` accordingly.
  7. Repeat the above until you either find the value `target` or `left` becomes greater than `right`, at which point you know the element is not present and can return `false`.

  Your method may assume that the array given is sorted. That is to say, it is not necessary to check this condition ahead of time and it does not matter whether your program produces the "right" result if the array is not sorted.

  You may read more about binary search at `http://en.wikipedia.org/wiki/Binary_search_algorithm`.

  Hint: If your code runs forever, you probably have an infinite loop. In this case, a good way to debug the problem is to, inside your loop, print the value of the variables left, right, middle and array[middle]. This will let you figure out which variable is not getting updated properly.

- `public static int[] copy(int[] array)` : This method should take as input an `array` and it should duplicate it before returning the duplicate. Remember that since arrays are *reference* types, you will need to make a brand new array and, one at a time, copy elements from one into the other. Otherwise, you will not be making a second array. This means that if you are not using the new operator and a loop in your method, you probably did not copy anything.

- `public static void sort(int[] array)` : This method should take as input an *unsorted* array and it should sort it in increasing order so that the smaller elements are at the front of the array and the larger elements are later. You should do so by using a method called *Bubble Sort*. Barack Obama has commented that this is not the most efficient way to sort things, `http://www.youtube.com/watch?v=k4RRi_ntQc8`. We would like to test his claim and so will implement this sorting algorithm.

  You can read more about Bubble sort at `http://en.wikipedia.org/wiki/Bubble_sort`.

  The idea is the following:

1. Set up 2 nested loops.
2. In the outer loop, range `i` from 0 until `array.length - 1`
3. In the inner loop, range `j` from `i+1` until `array.length - 1`
4. At each step of the inner loop, check if `array[i] > array[j]`. Since `j` by definition is always greater than i, this would indicate an out of order sequence, and so you should swap the element at `array[i]` with that at `array[j]`

Hint: While testing, you will probably want to use the `printArray` method you wrote in the warm up questions.

Hint 2: We will talk about this in class when we deal with reference types, but remember if you change the value of a specific spot of an array inside of a method, it will change in the method that called it as well. However, if you change the whole array (by writing `array = new int[100]` for example, then you will NOT see a change in the method that called it.

- `public static int[] generateRandom(int n)` : This method should take as input an `int n` and generate an array of size `n`. It should then fill it with random integers by using the `Random` class and choosing a random int up to the value `Integer.MAX_VALUE` which is the largest possible value an `int` can hold in Java.

  First, create a `Random` object by writing `Random r = new Random(1)` at the top of your method. **Very important: ONLY DO THIS ONE TIME PER METHOD** or else you may get different results in later questions. Then you can generate a random integer by calling a method `nextInt` on the object `r`. For example: `r.nextInt(10)` will generate a random integer between 0 and 9 (including 9 and 0).

  Note that your numbers will be "predictably" random in that from one run until the next they will be exactly the same. We'll talk in class briefly about what one could do to solve this issue in practice, but for the assignment it is much better this way as it is easier to debug any issues you run into. For more information on random numbers, please see `http://docs.oracle.com/javase/1.4.2/docs/api/java/util/Random.html`. Note that you need to import `java.util.Random` to do this.

Question 2: StopWatch class (30 points)

The following code should be placed in a class `StopWatch` *and thus a file* `StopWatch.java`

In this question, you will define a new type `StopWatch` which is designed to represent a real world stop watch object. A real stop watch has a few things you can do to it:

- start the watch
- stop the watch
- view the time

The last option only works in practice if the watch is stopped since otherwise it is moving too fast to get a good count.

Define a new class `StopWatch`. In this class, you should define two *private* properties `long startTime` and `long stopTime`.

In addition to these properties, you should define six methods:

1. A method `start` which sets the property `startTime`. It can do this by calling the method `System.nanoTime()` which gets the number of nano seconds since a fixed event occurs. Store this result into the property `startTime`.
2. A method `stop` which sets the property `stopTime`. It does this as well by calling the method `System.nanoTime()` Store this result into the property `stopTime`.

3. A method `getTimeNano` which returns a long by calculating the difference between `stopTime` and `startTime`. The idea is that it doesn't matter what time was used as the base to calculate the number of nano seconds in the `nanoTime` method as it will be consistent when you get it later on. So you have measured the number of nano seconds between the time the methods `stop` and `start` were called.

4. A method `getTimeMicro` which returns a long representing the number of microseconds between the time that `stop` and `start` were called. Note that there are one thousand nanoseconds in a microsecond. So your answer will be the same as in the `getNanoseconds` method but divided by one thousand.

5. A method `getTimeMilli` which returns a long representing the number of milliseconds between the time that `stop` and `start` were called. Note that there are one million nanoseconds in a millisecond.

6. A method `getTimeSeconds` which returns a long representing the number of seconds between the time that `stop` and `start` were called. Note that there are one billion nanoseconds in a second.

All of the above methods can be done in just one or two lines. The point is that by writing these simple methods you make the code in the next question simpler and more intuitive to work with.

Note that you may assume the stop watch methods are called in the proper order. That is, you may assume that the programmer calls start, then stop, and then at least one of the get time methods.

You can optionally test your class by writing a main method. In this, you can create a `StopWatch`, call the method `start` and `stop` and surround the code you want to time. Then you can print the result afterwards. For example:

```
public static void main(String[] args)
{
    StopWatch watch = new StopWatch();
    watch.start();

    for (int i = 0; i < 100; i++)
    {
        System.out.println("I'm going to test how fast it is to print this 100 times");
    }

    watch.stop();

    System.out.println("It took " + watch.getTimeNano() + " nanoseconds to complete that task");
}
```

The reason the above is more intuitive is that the commands now mirror more closely how we think of executing this sort of task in real life. Rather than keeping track of an implementation detail of the fact that we called the method `System.nanoTime()` we can think of starting and stopping a stopwatch. This makes the code easier to read/understand and helps minimize bugs.

Question 3: Testing the president's claim (30 points)

In this question, you will compare the speed of the methods you wrote in the first question. A key concept in programming and computer science (and life!) is to make claims under the *worst case scenario*. It is great to know that it's possible you can go to Las Vegas and walk out a billionaire, but really it's most important to consider that in the worst case, you will end up broke. (Another, important aspect of computer science is to analyze the *average case scenario* but in many situations this is more difficult as it depends on knowledge of your input data. In the Las Vegas analogy, this involves a complex analysis of the rules of the games you are playing. In the array examples below it relies on knowledge of what sorts of arrays your method is being used on.)

Here is what you should do:

First, set up a loop in which you change `n` from 10 until 500 million. At each step of your loop you should double the value of n (as opposed to the "standard" way of increasing n by one).

In the body of your loop, you should do the following:

1. Create a `StopWatch` object. You will call `start`, `stop`, and `getTimeMicro` and appropriately place them around your method calls.
2. First generate an array filled with random values of size n. Store how long this took in microseconds.
3. Next search for the number -1 in your array using the `linearSearch` method. Note that the number -1 will NEVER occur in your array because of the way we generate values. This means that we are testing the *worst case scenario* for the time linearSearch would take. This is since a number NOT appearing requires searching through the whole array to conclude it's not present. Whereas if the number appears, we will see it and can stop searching earlier. Store how long this search took in microseconds.
4. Next call the `duplicate` method from the first question and store the result into a new array variable. Store how long this method took in microseconds.
5. Call the `bubbleSort` method that you wrote. Remember that this will CHANGE the array you initially had. Store how long this method took in microseconds. See important note below if your program is taking too long here.
6. Call the method `Arrays.sort` and pass it the duplicated array. `Arrays.sort` is a library method that will sort an array it is given. It does NOT use bubble sort, and the method it uses is faster. You use it by passing the array as input and voila! It is sorted. Store how long this method took in microseconds.
7. Finally, call the method `binarySearch` and look for the value -1 in the sorted array (again a worst case scenario with respect to run time). Store how long this time took in microseconds.
8. Now, print all the times of the various methods, by printing, on one line, in order:

   `n,timeGenerateArray,timeLinearSearch,timeDuplicate,timeBubbleSort,timeLibrary,timeBinarySearch`

Do this for each value of n and you will have a table as below of run times.

```
> run SearchComparison
10,754,2,2.0,5,37,2
20,17,1,1.0,10,10,1
40,34,1,2.0,37,23,1
80,60,2,3.0,150,46,1
160,83,3,4.0,466,101,2
320,224,9,9.0,1234,248,1
640,326,12,12.0,859,560,1
1280,875,24,26.0,1998,1117,2
2560,1837,70,64.0,8746,80,2
5120,3591,144,151.0,34510,136,2
10240,786,323,323.0,147253,255,2
20480,306,11,21.0,614641,731,2
40960,588,18,31.0,2642584,1811,3
81920,1255,40,71.0,10330390,2767,4
163840,2402,104,175.0,1,27508,5
327680,5399,167,980.0,1,41400,5
655360,10910,281,4183.0,1,89825,5
1310720,18016,553,1364.0,1,187398,6
2621440,38682,1169,9534.0,1,385152,6
5242880,83077,2043,12574.0,1,847580,6
10485760,186875,9198,59617.0,1,1714917,6
```

```
20971520,318668,8461,100669.0,1,3669946,8
41943040,620468,16761,158592.0,1,7304924,7
83886080,1188193,36870,298203.0,2,15541616,6
167772160,2304454,69352,563400.0,1,32231608,6
335544320,4604220,134922,1127605.0,1,67180903,7
```

You should see that binary search is MUCH faster than linear search. Additionally, the library method for sorting will be MUCH faster than bubble sort, showing that Obama was correct that bubble sort is NOT the way to go.

IMPORTANT NOTE ABOUT BUBBLE SORT: You will see that your sorting algorithm for bubble sort is too slow to run once `n` becomes larger than 100,000 or 1 million depending on your computer's speed. Because of this, you will most likely need to add an if statement to skip the bubble sort step when `n` is large. You can then print 0 in the table at that point. There are some entries like this in the table above starting with the entry for n equals 163,840.

Not for credit question: Approximately how long would it take to run if you did not add this if statement? Hint: It would not finish before the end of the semester!

Follow up not for credit question: Since the output is comma separated, you should be able to copy the results into a text file and then import them into a spreadsheet such as open office or excel. You can then graph the number of microseconds as a function of `n`, the size of the array. What functions do these numbers look like to you?

On one of your runs, copy and paste the output into a text file called `results.txt` so that you can send the results of the experiment to the TA. Note that if you are running from the command prompt, you can take a short cut to do this by writing:

```
java SearchComparison > results.txt
```

This will cause the output of the program to be "printed" to the file results.txt instead of to the screen. You should see the file created in the same folder as your command prompt window is set at (i.e. where SearchComparison.class is located).

What To Submit

You should submit your assignment on MyCourses. In order to do this, you will need to make a `zip` of the file. You can do this on windows by following the instructions at this link: `http://condor.depaul.edu/slytinen/instructions/zip.html`. On a mac or linux, you can find instructions at `http://osxdaily.com/2012/01/10/how-to-zip-files-in-mac-os-x/`

You should submit a zip file called `Assignment3.zip` with the following files inside of it.

`ArrayUtilities.java`
`StopWatch.java`
`SearchComparison.java`
`results.txt`
`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into doing this assignment. If you point out an error that you know occurs in your problem, it may lead the TA to give you more partial credit. On the other hand, it also may lead the TA to notice something that otherwise he or she would not.