ASSIGNMENT 3

COMP-202, Fall 2014, All Sections

Due: October 24^{th}, 2014 (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.

Question 1:	10 points
Question 2:	30 points
Question 3:	20 points
Question 4:	15 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. Marks can be removed if comments are missing, if the code is not well structured, and if your solutions do not respect the assignment requirements.

Assignment

In this assignment, you will use all of the programming building blocks that you have learned so far in class by applying them to a particular problem: **sorting a list**. This problem is quite easy to describe. We take as input a list of things that are out of order, and produce a list with the same things but in ascending or descending order.

While sorting is a simple problem to describe, it is not so simple to solve efficiently. Many of the concepts and techniques that you've learned in class are directly applicable to sorting problems. Conditional statements, loops, methods, passing arguments, computational complexity and recursion, all appear in the context of sorting algorithms. So this is a great way for you to practice **computational thinking**; i.e. how to find a solution to a problem and transform that solution into instructions that can be executed by a computer.

Part 1 (0 points): Warm-up

For this part, you will need the following file: **ArrayUtilities.java**

Exercise 1: Write a method that takes in an array of integers and verifies if they are sorted. It should return true if the array sorted and false otherwise. This method will be useful for testing your code in later problems.

Exercise 2: Write a method that takes an array of Strings and verifies if these Strings are sorted lexicographically. It should return true if the array is sorted and false otherwise. You will want to use the compareTo() method in order to do this. For example, if the array is:

```
{"Vladimir", "Ismail", "Priya", "Jiequn", "Melanie", "Camilo", "Jonathan"}
```

your method should return false. If the array is:

```
{"Camilo", "Ismail", "Jiequn", "Jonathan", "Melanie", "Priya", "Vladimir"}
```

your method would return true. You can test your code using the methods getSortedNamesArray(int n) and getRandomNamesArray(int n) in the ArrayUtilities.java class[1].

Exercise 3: Write a method called *reverse* which takes as input an array of integers and reverses the order of the elements **in place**. This method should not create a new array, and should return nothing. For example, if the input is:

```
{3, 2, 98, 32, -12, 1}
```

after the method is executed the array shall be

```
{1, -12, 32, 98, 2, 3}
```

Part 2:

For this part, you need the following files: ***Sorting.java***, ***ArrayUtilities.java***, ***PlotWindow.java*** *and* ***first-names.txt***.

Question 1: Comparing two String (10 points)

In the file `Sorting.java`, write a method called `Compare`[2] that takes `String s1` and `String s2` as arguments and returns an integer. It should compare `s1` and `s2` alphabetic ally such as follows:

$$\texttt{Compare}(s_1, s_2) = \begin{cases} 1 & \text{if } s_1 > s_2 \\ -1 & \text{if } s_1 < s_2 \\ 0 & \text{otherwise} \end{cases}$$

You cannot use any built in functions[3] for comparing strings provided by `Java` for this question[4]. You should use a *for loop* that compares each character one by one[5]. Recall that we can extract characters from Strings using the `charAt` method. You are to treat upper and lower case as if they were identical[6].

Question 2: Search in an Array (30 points)

a. **Linear Search** - Inside the `Sorting.java` file, write a method called `linearSearch` that receives as input a variable, `a`, of type String array (`String[]`), as well as a target variable denoted `t`[7].

[1] The process of using these functions is described in question 1.
[2] Marks will be deduced if names do not match
[3] That includes `compareTo` method
[4] That includes `toUpperCase` and `toLowerCase` as well
[5] Remember, the Strings might have different lengths
[6] Pay attention to the ascii values of upper *vs.* lower cases
[7] Make sure to respect the arguments' order, *e.g.* linearSearch has a `String` array (`String []`) as 1st argument and a `String` as target for 2nd argument

Your method should iterate through all the elements of `a`[8]. While iterating through the different elements, if the target name matches an entry in `a`, you should return the position of the array at which the target was found. If the target is never found, the method should return `-1`.

For example, if the following array is passed to your method as argument,

```
{"Vladimir", "Ismail", "Priya", "Jiequn", "Melanie", "Camilo", "Jonathan"}
```

with the string `"Melanie"` as target. The method returns 4. On the other hand, using the same array as input but with the string `"BlipBlop123"` as target, it returns -1.

b. **Binary Search** - Assume that array `a` is sorted in alphabetical order. Inside the `Sorting.java` file, write a method named `binarySearch` that takes as arguments a `String` array, `a`[9], and a target String denoted `t`. Instead of iterating through the array you have to implement the binary search algorithm[10] given by the following speudo code[11],

```
procedure binarySearch(A, target)
  min = 0
  max = length(A) - 1

  while (max is greater than equals than min)
    mid = midpoint(min, max)

    if(A[mid] is target)
      return mid;
    else if (A[mid] is smaller than target)
      min = mid + 1
    else
      max = mid - 1
    end if
  end while

  return -1
```

You have to use the `Compare` method defined above to compare the two Strings and to know if you want to go to the left or right of the array[12]. If the method finds the target, it returns its array index, `-1` otherwise.

Question 3: Sorting Arrays (45 points)

a. **Bubble sort** - Write a method called `bubbleSort` that takes as input, `String[] a`, and sorts this array using bubbleSort. We define the number of comparisons as the number of times any entry is compared to an other one. In other words, this will be the number of time the method `Compare` is called in your code. Your bubble sort algorithm should follow this pseudo-code[13] and should return the number of comparisons made.

[8] This is achieved using a loop
[9] `a` is of type `String[]`
[10] `http://en.wikipedia.org/wiki/Binary_search_algorithm`
[11] Feel free to implement the recursive version of binary Search
[12] You are forbidden to use compareTo()
[13] This is not java code

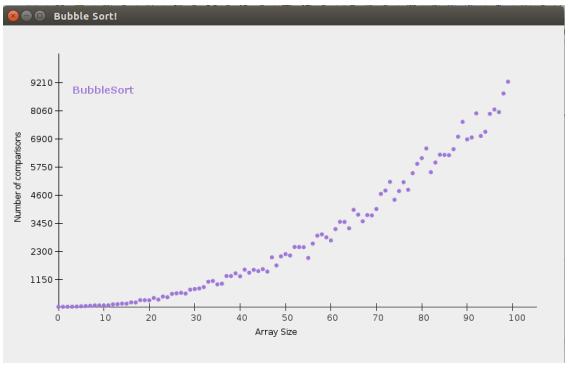

Figure 1: An example of a plot generated with the PlotWindow class.

```
procedure bubbleSort(A)
    n = length(A)
    while swapped
       swapped = false
       for i from 1 to n-1 do
          if A[i-1] greater A[i]
             swap(A[i-1], A[i])
             swapped = true
          end if
       end for
    end while
end procedure
```

Using the provided method called `plotBubbleSortTest` in `Sorting.java` you should be able to plot the number of comparisons needed as the input size grows. The plot should look like the example in Figure 1. **N.B.** The provided pseudo code does not include any counter for comparisons.

b. **Comb sort** - This question's instructions are identical to the previous question except you are to implement `Comb Sort`. Write a method denoted `combSort` that takes as input, `String[] a`, and returns the number of comparisons made. The pseudo-code for this algorithm is expressed as follows,

```
procedure combSort(A)
    gap = length(A), where gap is an integer
    shrink = To determine       <---------- N.B. decimal

    while swapped or gap is not 1
        gap = gap/shrink
        if gap is smaller than 1
          gap = 1
        end if

        i = 0
        swapped = false

        while i+gap is smaller than length(A)
          if A[i] is greater than A[i+gap]
            swap(A[i], A[i+gap])
            swapped = true
          end if
          i++
        end while
    end while
```

```
end procedure
```

Using the provided method called `plotCombSortTest` in `Sorting.java` you should be able to plot the number of comparisons needed as the input size grows. Note that the shrink value, which is a decimal number, is not provided. Try different values of `shrink` and use the plot method to determine the best candidate value[14]. **N.B.** The provided pseudo code does not include a counter for comparisons.

c. **Compare the results** - Plot the number of comparisons of both algorithms with input size ranging from 1 to 1000 in a method called `plotCombBubble`. By looking at the provided methods for plotting your Comb and Bubble sort methods, you can infer how to create this new plot. The method `pw.addPlot(int[])` is to be used for plotting multiple entries. For this question you must submit the plot.

Which algorithm is faster? We say that a sorting algorithm is faster than another if it performs fewer comparisons. So, which algorithm makes the smaller number of comparisons? Please provide your answer as a comment in `Sorting.java`'s `plotCombBubble` method.

Question 4: Comprehension (15 points)

Please write your answer in the `Question4.txt` file. You do not need to provide the code, only the answers to the questions. Please organise your answers so as to make it as easy as possible for the marker to read and understand. Each question is worth 5 points.

a. If we call the `binarySearch` method with the following input array,

```
{"aaa", "aab", "aba", "abb", "baa", "bab", "bbb", "bbb"}
```

and with `abc` as the target, how many comparisons[15] will be made including only the comparisons inside the method `Compare`?

b. The `bubbleSort` method sorts the elements of an array in increasing order. If we wanted to change this method to sort the array in decreasing order, how many lines of code do you need to change?

c. The following iterative sequence is defined for the set of positive integers:

$n \to n/2$(n is even)

$n \to 3n + 1$(n is odd)

Using the rule above and starting with 13, we generate the following sequence: $13 \to 40 \to 20 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$

It can be seen that this sequence (starting at 13 and finishing at 1) contains 10 terms. Although it has not yet been proven (Collatz Problem), it is thought that all such sequences will finish with 1. Which starting number, under 3 million, produces the longest chain?

N.B.: Once the chain starts the terms are allowed to go above three million.

What To Submit

You have to submit one zip file that contains all your files to myCourses - Assignment 3. If you do not know how to zip files, please ask your friends or your favourite search engine. For these kinds of small problems, Google can be your best friend.

[14] Hint: if you make the `shrink` value too big, `combSort` will be the same as `bubbleSort`. But making it too small might make things worse!

[15] only character comparisons, do not count the length comparisons.

`Sorting.java` - Java code
`Plot.png` - Plot image
`Question4.txt` - please provide meaningful information

`Confession.txt` (optional) In this file, you can tell the TA about any issues you ran into while 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.

Marking Scheme

Up to 30% can be removed for bad indentation of your code as well as omitted comments, poor coding structure, or missing files. Marks will be removed as well if the class names are not respected.

Question 1

The compare method returns 0 if the strings are the same	3	points
The compare method works correctly for strings of the same size	3	points
The compare method works correctly for strings of different size	4	points
	10	**points**

Question 2

Correctly implemented `linearSearch`	10	points
Correctly implemented `binarySearch`	20	points
	30	**points**

Question 3

The `bubbleSort` produces sorted arrays	10	points
The `combSort` produces sorted arrays	15	points
The number of comparisons are counted correctly in each sorting method	10	points
Produced a plot of that compares the two methods	10	points
	45	**points**

Question 4

a.	5	points
b.	5	points
c.	5	points
	15	**points**