CS 249 Project #1
Fall 2013
Due on 10/4/13 by Midnight

Project Description: The purpose of this project is to ensure you understand how to work with one of Java's most basic data structures, arrays, and that you can implement basic algorithms to manipulate the data stored with them as well as use them to implement other ADTs (i.e., queues). Additionally, you will practice analyzing efficiency (i.e., order notation).

Project Submission: You should upload a zip file of your project on BBLearn. The project should contain all of your code (i.e., all of .java files, do NOT include any IDE-specific files) along with a readable screen capture (or UNIX script) of your program compiling and running (i.e., you should the output of your program). Additionally, you MUST include a short write up that:

- Details exactly how to execute your code to test each part
- Discusses the efficiency (i.e., Big Oh) of each method (except drivers)
- Addresses any questions associated with task, and
- Identifies anyone you collaborated* with.

*Note: Collaboration with respect to how to solve the problem is acceptable. However, you must write and comment your own code individually.

Grading: Your grade will be based on completeness and correctness (60%), as well as design and consistent use of coding standards (e.g., formatting, comments, etc.) (30%), and the answers to each question and discussion of efficiency (10%).

PART I: Sorting Implementations

(A) Modify the insertionSort() method given in your text so that it counts the number of copies (i.e., assignment from one variable to another) and the number of comparisons (i.e., a comparison between two integers in the array) it makes during a sort and displays the totals. To count comparisons, you'll need to break up the double condition in the inner while loop of the implementation used in the book. Use this program to measure the number of copies and comparisons for different amounts of inversely sorted data. Do the results verify $O(N^2)$ efficiency? Do the same for almost-sorted data (only a few items out of place). What can you deduce about the efficiency of this algorithm for almost-sorted data?

(B) Another simple sort is the even-odd sort. The idea is to repeatedly make two passes through the array. On the one pass you look at all the pairs of items, a[j] and a[j+1], where j is even (j = 2, 4, 6, ...). If their key values are out of order, you swap them. On the next pass you do the same for all the odd values (j = 1, 3, 5, ...). You do these two passes repeatedly until the array is sorted. Your main method must illustrate it works for varying amounts of data. You'll need to figure out how many times to do the two passes. Also, give an example of where or when a sorting algorithm like this would be particularly useful and explain why.

PART II: Queues

(A) Write a method for the Queue class (given in the text) that displays the contents of the queue. Note that this does not mean simply displaying the contents of the underlying array. You should show the queue contents from the first item inserted to the last, without indicating to the viewer whether the sequence is broken by wrapping around the end of the array. Hint: Be careful that one item and no items display properly, no matter where front and rear are.

(B) Create a Dequeue class based on the discussion of dequeues (i.e., double-ended queues) in Chapter 4 of your text. It should use an array and include insertLeft(), insertRight(), removeLeft(), removeRight(), isEmpty(), and isFull() methods. It will need to support wrap-around at the end of the array, as queues do. Write a driver class to execute these methods and show that the wrap around works on either side.