CPS112: Computational Thinking with Data Structures & Algorithms
Lab 10: Searching for for an answer

1. (**Conceptual/Programming**)
   a. You have been given the file *Binomial.class* (yes .class). This file contains a (broken) memoized implementation of the Binomial Coefficient calculation from last lab. If you run the file you will get an index out of bounds error. Using `jdb`, find the following information
      i. What values of *n* and *k* cause the program to crash?_____
      ii. Give the values of *n* and *k* for the two recursive calls immediately preceding the call that crashed the program._____
   b. You have been given the file *Mystery.class*
      i. Set a breakpoint for the constructor and begin debugging. What command did you enter?
      _____

      ii. What instance variables (i.e. fields) and their types does this class contain?
      _____
      _____
      _____
      _____

      iii. What methods (excluding built-in Java methods) does this class contain?
      _____
      _____
      _____
      _____
      _____
      _____

      iv. In the `main` method, three variables are created. What are they are what arguments are given to the constructor for each?
      Variable 1: _____ Arguments: _____
      Variable 2: _____ Arguments: _____
      Variable 3: _____ Arguments: _____

2. (**Programming**) Bitonic arrays
   Given an array, *A*, of numbers, we say that *A* is *bitonic* if the elements of *A* are first strictly increasing and after reaching a maximum value, they are strictly decreasing. That is, for some index *j*, $0 \le j \le n$-1, the array *A* is strictly increasing A[0] < A[1] < ...< A[j]; and is strictly decreasing after that, A[j] > A[j+1] > ... >

A[n-1]. Note that j could be 0 or n-1. For this problem you will write two different methods for finding the maximum element in a bitonic array as follows: Write and test your code in the provided file, *Bitonic.java*.

Method 1: `linearFindMax(int [] bitonicArray)`
Find the position of the maximum element in a bitonic array by simple linear search.

Method 2: `binaryFindMax(int [] bitonicArray)`
Find the position maximum element in a bitonic list using "binary search" algorithm. The algorithm is as follows:

a. Examine the "middle" element of `bitonicArray` and the elements immediately to its left and immediately to its right (you need to consider the boundary positions which only contain 1 neighbor, i.e., position at 0 or `bitonicArray.length - 1`).
   i. If the middle element is bigger than its left and right neighbors, then it is the maximum element.
   ii. Else, depending on the comparisons, search for the maximum element in the left half of the list or search for the maximum element in the right half of the list.

   Both search functions should return the position of the maximum item. I have provided you with test code to ensure your methods work for three general cases:
   1. when the max is at the beginning of the array
   2. when it is at the end of the array
   3. when it is somewhere in between.

b. What is the worst-case time complexity (in Big-O notation) of `linearFindMax` and `binaryFindMax` for an array of *n* items? Why?

– MIDWAY CHECK –

3. (**Programming**) This problem will be good practice for your homework on sorting using iterators. In the file *Duplicates.java* you have been given the shell of a method to check if a list contains duplicate values using iterators. You have also been given some test code. You can see from the test code that the method should work for both array-backed lists and linked lists. Fill in the method based on the following general algorithm:
   a. Begin by cloning the iterator given as a parameter.
   b. Using the cloned iterator, go through the list one value at a time starting with the first item. How do you get the value at a certain position in the list using an iterator method?

c. Once you have a value, clone again (think carefully which iterator gets cloned) to go through the remainder of the list to check it against every other remaining value in the list. If at any point two values match, return true.
d. If you get through the entire list, return false, there are no duplicates.

– FINAL CHECK –