Name: _____

USC username (e.g., `ttrojan`): _____

CS 455 Final Exam
Spring 2014 [Bono]

Monday, May 9, 2014

There are 6 problems on the exam, with 66 points total available. There are 7 pages to the exam, including this one; make sure you have all of them. There is also a separate double-sided one-page code handout. If you need additional space to write any answers, you may use the backs of exam pages (just direct us to look there).

Note: if you give multiple answers for a problem, we will only grade the first one. Avoid this issue by labeling and **circling** your final answers and crossing out any other answers you changed your mind about (though it's fine if you show your work).

Put your name and USC username at the top of the exam. Please read over the whole test before beginning. Good luck!

	value	score
Problem 1 & 2	12 pts.	
Problem 3	8 pts.	
Problem 4	6 pts.	
Problem 5	20 pts.	
Problem 6	20 pts.	
TOTAL	66 pts.	

Problem 1 [3 points]

Give the big-O worst case time to solve each of the following problems (put your answers in the space provided to the left):

_____ 1. compute the maximum value in an array of size *n*

_____ 2. use sequential search on a sorted array (where we end the search early when we get a to a value bigger than the one we are looking for) of size *n*

_____ 3. use mergesort to sort n values in an array

Problem 2 [9 points total]

Consider the following ***C++*** code sequence:

```
Node * p = new Node(3);
p->next = new Node(1);
Node * r = p->next;
r = new Node(7);
Node * s = new Node(4);
s = p->next;
cout << p->data << " " << r->data << " " << s->data << endl;
```

Part A. Show a box-and-pointer diagram for this code sequence (cross out old values of pointers and/or data so we can see how things changed):

Part B. Show the output of the code sequence:

Problem 3 [8 points]

This is a ***Java*** problem. Consider the following interface:

```
public interface PointVisitor {
  public void visit(Point p);
}
```

and a function that uses this interface to apply a `PointVisitor` callback function to all the elements in an array of `Point` objects:

```
public static void visitAll(Point[] points, PointVisitor visitor) {
  for (int i = 0; i < points.length; i++) {
    visitor.visit(points[i]);
  }
}
```

Hint: `PointVisitor` allows us to customize the behavior of `visitAll`, just like `java.util.Comparator` allows us to customize the behavior of `Arrays.sort` and `Collections.sort`. See also reminder of interface/implements syntax on the code handout. The `Point` class is also on the code handout.

Write the code necessary to use `visitAll` to complete the implementation of method `expand`, so that it moves each point to be twice as far from the origin as it was before. You may have other code in addition to the method itself. You do not have to write code to deal with round-off errors. Solutions that don't use `visitAll` will receive no credit. You may assume `visitAll` and `expand` are in the same class.

> **Example**: if the array, `points`, contained [(13, 12), (9, -9), (-10, -5), (5, 6)] before the call,
>
> after a call to `expand(points)` it would contain: [(26, 24), (18, -18), (-20, -10), (10, 12)]

```
// moves each point in the array to a location twice as far from the origin
public static expand(Point[] points) {
```

Problem 4 [6 points]

Consider the following ***Java*** program that uses exceptions. Note: `MyException` and `FileNotFoundException` are both subclasses of `IOException`:

```
public class MyProg {
   public static void main(String[] args) {
      try {
         ArrayList<String> lines = readFile();
         int maxSoFar = 0;
         for (int i = 0; i < lines.size(); i++) {
            if (lines.get(i).length() > maxSoFar) {
               maxSoFar = lines.get(i).length();
            }
         }
         System.out.println("Max line is " + maxSoFar);
      }
      catch (MyException exc) {
         System.out.println("Error #1");
      }
      catch (IOException exc) {
         System.out.println("Error #2");
      }
   }

   public static ArrayList<String> readFile() throws IOException {
      ArrayList<String> lines = new ArrayList<String>();
      try {
         Scanner inFile = new Scanner(new File("inputData"));
         if (!inFile.hasNext()) {
            throw new MyException();
         }
         while (!inFile.hasNextLine()) {
            lines.add(inFile.nextLine());
         }
      }
      catch (FileNotFoundException exc) {
         System.out.println("Error #3");
      }
      return lines;
   }
}
```

What is the output of the program for each of the following scenarios:

1. The file *`inputData`* does not exist.

2. The file *`inputData`* is empty.

3. The file *`inputData`* has the contents:

   ```
   a big bad
   elephant
   went forward
   today
   ```

Problem 5 [20 pts]

Implement the *Java* class `AssocList`, that has the methods illustrated below and specified in method comments. For full credit your implementation must be able to do each of its operations in O(1) time.

Description of `AssocList`: An `AssocList` associates sequential `ints` with unique `strings`. If we add a new string to the `AssocList` it associates it with the next unused number; it assigns 0 for the first one. The main other operations are to find the associated number, given a string, and find the associated string, given a number.

For example, for a `Date` class that needs to accept and/or print months in name or number form, it might be useful to associate month numbers with month names, such as in the following code:

```
AssocList monthList = new AssocList();

monthList.addAssoc("dummy");  // is associated with 0:
                  // (a trick so real month number sequence starts at 1)
monthList.addAssoc ("January");  // is associated with 1
monthList.addAssoc ("February"); // is associated with 2
. . .
monthList.addAssoc ("December"); // is associated with 12
. . .
System.out.println("November is month number "
                    + monthList.getNumber("November"));

System.out.println("Month 3 is called " + monthList.getName(3));

System.out.println("Number of months is" + monthList.numAssocs() - 1);
                                                  // minus 1 because of dummy
```

Hints: You are encouraged to use the Java library however you wish to help you – several Java containers and algorithms are described on the code handout. Also, to meet the time requirement above, you are allowed to use more space (i.e., memory) than you might otherwise need.

Detailed descriptions of the methods appear with the class interface as well as space for your answer below and continued on the next page:

```
public class AssocList {
    // space for instance variable(s) here









  // creates an empty AssocList
  public AssocList() {









  }
```

Problem 5 (cont.)

```
  // returns number of associations in this AssocList
  public int numAssocs() {




  }
  // make a new association,
  // and return the number to be associated with the given name.
  // first name added will get the number 0, second name will get 1, etc.
  // if this name is already in AssocList, returns its old association
  // (i.e., does not change AssocList in that case)
  public int addAssoc(String name) {









  }
  // find number that goes with name in this AssocList
  // or -1 if name is not in AssocList
  public int getNumber(String name) {









  }
  // find name that goes with number in this AssocList
  // PRE: 0 < = number < numAssocs()
  public String getName(int number) {









  }
}
```

Problem 6 [20 pts]

Implement the *C*++ function `fibSeq` which creates and returns a linked list of the first *n* numbers in the Fibonacci sequence.

The Fibonacci sequence starts as follows: 1, 1, 2, 3, 5, 8, 13, . . .
The rules for generating the sequence are, where f_n, below, means the *n*'th Fibonacci number:

$f_1 = 1$

$f_2 = 1$

for any $n > 2$, $f_n = f_{n-1} + f_{n-2}$

Thus, the nth Fibonacci number is the sum of the previous two numbers in the sequence. So, in the sequence shown above, the third value is 1 + 1 = 2; the fourth value is 1 + 2 = 3, and the fifth value is 2 + 3 = 5, etc.

Examples below (linked list of ints shown as a space separated sequence of ints surrounded by parentheses):

n	return value of **`fibSeq(n)`**
8	*(1 1 2 3 5 8 13 21)*
2	*(1 1)*
1	*(1)*
3	*(1 1 2)*

Note: The `Node` and `ListType` definitions are on the code handout.

```
// PRE: n > 0
ListType fibSeq(int n) {
```