Computer Science 141 – Final Exam

December 15, 2003

General Directions. This is an open-book, open-notes, open-computer test. However, you may not communicate with any person, except me, during the test. You have three hours in which to do the test. Put your answer to each question in the space provided (use the backs of pages if you need more space). Be sure to **show your work**! I give partial credit for incorrect answers if you show correct steps leading up to them; conversely, I do not give full credit even for correct answers if it is not clear that you understand where those answers come from. Good luck.

This test contains 8 questions on 8 pages.

Question 1 (15 Points). Suppose the rules for Towers of Hanoi were changed, so that puzzles that involve *n* disks also have *n* poles. All other features of the game (the rules, the goal, etc.) remain the same. Does solving this new version of the puzzle still require an exponential number of moves? Why or why not?

Question 2 (15 Points). Sleazy Sammy's Computer Emporium and Used Fish Warehouse is deciding whether to invest in developing the world's greatest chess-playing computer program. However, the executives at Sleazy Sammy's have heard that playing a winning game of chess is an intractable problem, and they are arguing over what this means for their project. Various executives express the following opinions:

A. “Intractable” means that no program can ever win a game of chess.

B. “Intractable” means that any program that always wins at chess will need to run on a fast supercomputer in order to play in a reasonable amount of time.

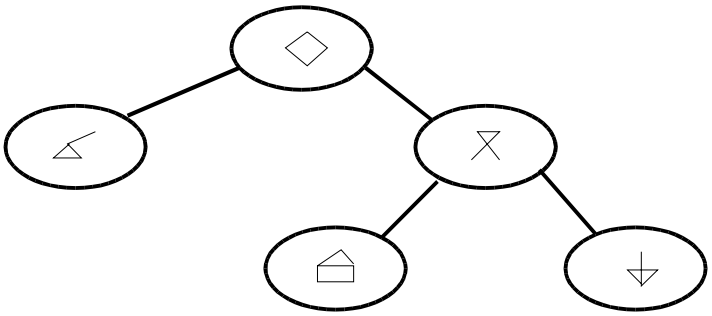
C. “Intractable” means that a program that always wins at chess could take billions of years to play a game.

D. “Intractable” is an irrelevant theoretical concept that has nothing to do with real software development.

E. “Intractable” means that only really smart computer scientists will be able to figure out how to write the program.

Sleazy Sammy finally hires you as a consultant to settle the argument. Which of the above views do you support, and why?

Question 3 (10 Points). Here is a binary search tree whose nodes contain shapes:



While one doesn't normally think of shapes such as these having an order (i.e., one shape being “less than” or “greater than” another), I in fact made up an ordering for shapes so that I could create this tree. Assuming that the tree is a correct binary search tree, list the shapes in order from least to greatest according to my ordering.

Question 4 (15 Points). Suppose you believe that a certain algorithm has an execution time of $O(2^n)$. To test this belief, you measure the algorithm's actual running time on inputs of various sizes, and collect the following data:

Input Size (n)	***Time (mS)***
1	6
2	20
5	96
10	512
20	2000

Are these measurements consistent with the $O(2^n)$ hypothesis? Why or why not?

Question 5 (20 Points). Define a list, *E*, to be "embedded" in another list, *L*, if some subset of the items in *L*, not necessarily adjacent, but taken in the order they appear in *L*, form *E*. For example, the list [a [b []]] is embedded in the list [p [q [a [r [b [s []]]]]]]. Similarly, [a [b []]] is embedded in [p [a [b []]]], and even in [a [b []]] itself. On the other hand, [a [b []]] is not embedded in [p [b [a []]]]. Note that the empty list is embedded in every list, but no non-empty list is embedded in the empty list (thus, for example, [] is embedded in [p [q []]], but [p [q []]] is not embedded in []).

Write (pseudocode is fine) a recursive algorithm that determines whether one list is embedded in another. Your algorithm should be invoked by sending a message to one list, with a second list as the message's parameter; your algorithm should return True if the parameter is embedded in the list to which the message was sent, False otherwise. For example, "*L*.embeds(*E*)" would return True if *E* is embedded in *L*.

Question 6 (15 Points). Here is an algorithm that supposedly returns True if a binary tree contains an even number of items, and False if the tree contains an odd number of items:

```
// in class ExtTree, a subclass of BinaryTree
public boolean even() {
   if ( this.isEmpty() ) {
      return true;
   }
   else if (      ((ExtTree)this.left()).even()
            && ! ((ExtTree)this.right()).even() ) {
      return true;
   }
   else if (    ! ((ExtTree)this.left()).even()
            &&   ((ExtTree)this.right()).even() ) {
      return true;
   }
   else {
      return false;
   }
}
```

Prove that this algorithm is correct, i.e., that it really does return True if and only if the tree contains an even number of items.

Question 7 (15 Points). Suppose you are interested in how long it takes one of our software robots to paint a tile. So you write the following code:

```
...
Robot testBot = new Robot();
long start = System.currentTimeMillis();
testBot.paint( java.awt.Color.red );
long end = System.currentTimeMillis();
System.out.println( end – start );
...
```

You run this code 10 times, and it outputs the following numbers:

0, 0, 0, 0, 1, 0, 0, 0, 0, 0

Unfortunately, these numbers don't give you a very accurate sense for how long a “paint” operation takes. Explain how you would gather more accurate data. (Hint: Much of your explanation can consist of showing how you would modify the above code.)

Question 8 (15 Points). The following algorithm prints certain strings of nested parentheses. For example, if prints “(()())” when its parameter is 2, “()” when its parameter is 1, etc.

```
public static void printParens( int n ) {
  if ( n > 0 ) {
    System.out.print( “(“ );
    printParens( n-1 );
    printParens( n-1 );
    system.out.print( “)” );
  }
}
```

Derive (including any necessary proofs of closed forms) an expression for the number of characters this algorithm prints, as a function of *n*.

Have a Happy Holiday!