CS 211 Computers and Programming
Fall 2002

Prelim II Solution

November 19th, 2002

1. (Induction, 15 points)

   The Fibonacci sequence is the sequence

```
fib(1) = 1
fib(2) = 1
fib(n) = fib(n-1) + fib(n-2) (for n > 2)
```

   Let n_0 be the smallest positive integer such that $fib(n_0) > n_0^2$.

   (a) (5 points) Find n_0.

   (b) (10 points) Use induction to show that for all $n \geq n_0$, $fib(n) > n^2$.
   State clearly what the base and inductive cases are.

   Solution:

   (a) $n_0 = 13$

   (b)
```
Induction on n.

Base cases: n = 13, 14.
fib(13) = 233 > 13^2 = 169,
fib (14) = 377 > 196.

Induction step:
  Induction hypothesis: for n = 13, 14, ..., k, fib(n) > n^2
  We want to show that for n = k+1, fib(n) > n^2
  fib(k+1) = fib(k) + fib(k-1)
  From induction hypothesis, we know fib(k) > k^2 and fib(k-1) > (k-1)^2,
  so fib(k+1) > k^2 + (k-1)^2
  k^2 + (k-1)^2 = k^2 + k^2 - 2k + 1
                = k^2 + 2k + 1 + k^2 - 4k
                = (k+1)^2 + k(k-4)
  since k >= 13, k-4 > 0
  therefore, (k+1)^2 + k(k-4) > (k+1)^2
  so fib(k+1) > (k+1)^2 + k(k-4) > (k+1)^2

2 points for stating the base case.
2 points for stating the inductive case clearly.
6 points for correct proof.
```

2. (Asymptotic complexity, 10 points)

   (a) (2 points) What is the most accurate big-O complexity of f(n) = 2nlog(n) + 4n + 17log(n)? You do not need to justify your answer.

   (b) (2 points) Kareem Abdul-Java claims that the big-O complexity of the function f(n) in the previous part is $O(n^3)$. Is he right? Give an informal explanation why or why not.

   (c) (4 points) Give a formal proof that log(n+1) is O(log(n)) by finding a witness pair to establish this fact.

   (d) (2 points) Scarlett O'Java had an argument with Rhett Butler about sorting methods. She claimed that merge-sort will run faster than quick-sort on any input array (any size, any values) because merge-sort is an O(nlog(n)) algorithm while quick-sort is $O(n^2)$ algorithm. Is Scarlett right? If not, explain briefly why not.

Solution:

   (a) O(nlog(n))

   (b) Yes, because n^3 grows faster than $nlog(n)$.

   (c)
```
We must find a witness pair (k,n0) such that

log(n+1) <= k. log(n)  for n > n0.

Choose k = 2 and n0 = 2.

Since (n+1) < n*n for all n >= 2, the result is established.

2 points for correct k
2 points for correct n0
```

   (d) No. Quicksort is only $O(n^2)$ in worst case. It has $O(nlog(n))$ expected running time. In practice it is usually run faster than mergesort since the constant factor for quicksort is smaller.

3. (Graph search, 20 points)

Consider the following graph.

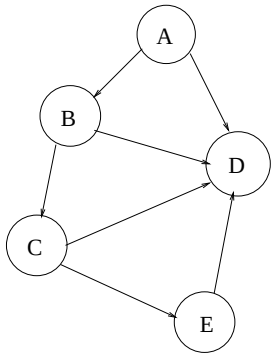

Figure 1: Graph search example

(a) (4 points) Write down two different breadth-first orderings of the nodes in this graph, starting from node A.

(b) (6 points) Write down two different depth-first orderings of the nodes in this graph, starting from node A.

(c) (10 points) The famous German aviator Graph Zeppelin wants to build the graph in Figure 1 using the TreeCell class discussed in lecture. That is, each node of the graph must be represented by a TreeCell object, and each edge of the graph is a left or right reference in a TreeCell object. Write down a sequence of Java statements which construct this graph. For your convenience, the TreeCell class is reproduced at the end of this exam.

Solution:

(a) (2 points per solution) A B D C E and A D B C E

(b) (3 points per solution) A B C D E and A B C E D

(c)
```
TreeCell a = new TreeCell(A);
TreeCell b = new TreeCell(B);
TreeCell c = new TreeCell(C);
TreeCell d = new TreeCell(D);
TreeCell e = new TreeCell(E);
a.setLeft(b);
```

```
a.setRight(d);
b.setLeft(c);
b.setRight(d);
c.setLeft(e);
c.setRight(d);
e.setLeft(d);

3 points for right number of calls to constructor.
1 point for each of the 7 edges inserted correctly.
```

4. (Lists and Trees, 25 points)

Write a class method that takes a binary tree as input, and returns a list containing all the elements in the tree in post-order. The tree is implemented using the TreeCell class (given at the end of this exam for your convenience), while the list must be implemented using the ListCell class (also given at the end of this exam). Your method will be passed the root of the binary tree as a parameter. You may not use any of the setter methods in the TreeCell class.

Hint: you may find it convenient to define a recursive helper function that takes both a tree and a list as input parameters, and returns a list as its result. Think carefully about how to traverse the tree to build the list painlessly. Our solution has about 10 lines of code. The ListCell and TreeCell classes are reproduced at the end of this exam.

Solution: The only tricky part of this problem is to realize that a list must be built from the bottom-up (unless you use setter methods), so the post-order linearization of the tree must be accomplished by making a pre-order traversal of that tree. Use a small example to understand this; it will also be discussed in section.

```
// Since new cell is added at the beginning of the list,
// in order to have post-order ordering of the tree nodes
public static ListCell toList(TreeCell root)
{
  if (root == null) return null;
  ListCell list = new ListCell(root.getDatum(), null);
  list = preorderTraversal(root.getRight(), list);
  list = preorderTraversal(root.getLeft(), list);
  return list;
}
// we need to visit the tree nodes in pre-order.
private static ListCell preorderTraversal(TreeCell root, ListCell list)
{ if (root == null) return list;
  list = new ListCell(root.getDatum(), list);
  list = preorderTraversal(root.getRight(), list);
  list = preorderTraversal(root.getLeft(), list);
  return list;
}
```

1 point for static

```
2 points for correct type declaration of toList header.
2 points for checking if root is null
2 points for appending value at root
6 points for correct calls to helper function
2 points for correct header for preorderTraversal
2 points for checking if root is null
2 points for appending value at root
6 points for making correct recursive calls
```

5. (Heaps, 15 points)

   (a) (10 points) Uriah Heap has a binary tree constructed from the TreeCell class described in class. He wants to know if the tree is a heap. Write a class method that takes the root of the binary tree as input, and returns true if the tree is a heap and false otherwise.

   (b) (5 points) What is the asymptotic complexity of your method? Justify your answer briefly.

   Solution:

   (a)
```
public static boolean isHeap(TreeCell root)
{
  if (root == null) return true;

  //check that left and right subtrees are heaps
  boolean leftIsHeap =  isHeap(root.getLeft());
  boolean rightIsHeap = isHeap(root.getRight());

  //compare value at root with values in children if any
  boolean leftIsLess = true;
  boolean rightIsLess = true;
  if (root.getLeft() != null))
    leftIsLess = ((Comparable)(root.getDatum())).compareTo(root.getLeft().ge
  if (root.getRight() != null))
    rightIsLess = ((Comparable)(root.getDatum())).compareTo(root.getRight().

  //put it all together
  return leftIsHeap & rightIsHeap & leftIsLess & rightIsLess;
}

1 point for static
1 point for correct type declaration
1 point for checking is root is null
1 point for chcecking left is heap
1 point for checking right is hap
1 point for checking left root value is less
1 point for checking if left is null
1 point for checking right root value is less
1 point for checking if right is null
```

```
1 point for correct comparisons
-1 point for no cast to Comparable
-1 point for not using getter methods
```

(b) O(n). Each tree node is visited once, and in each visit a constant number of operations are performed.

6. (Touring award, 15 points))

Willy Loman wants to make a travelling salesman's tour of the cities in the following graph. That is, he wants the tour to start at node A, and end at node A, visiting all cities and never visiting any city (other than A) more than once.

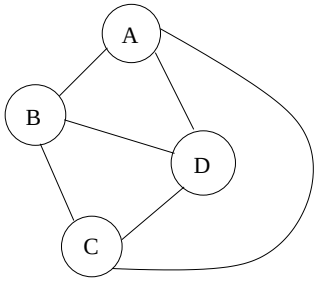

Figure 2: Willy Loman's world

(a) (6 points) Write down all the tours that Willy Loman can make.

(b) (3 point) How many tours are there in this graph?

(c) (6 points) In a graph with n nodes, what is the maximum number of possible tours? Hint: consider the complete graph with n nodes in which each node is connected to every other node.

Solution:

(a) (1 point per solution)

```
A B C D A
A B D C A
A C B D A
A C D B A
A D B C A
A D C B A
```

(b) 6

(c) n-1! (4 points for n!, 0 for any other solution)

```
class ListCell {

 protected Object datum;
 protected ListCell next;

  public ListCell(Object o, ListCell n){
    datum = o;
    next = n;
  }

  //this is sometimes called the "car" method
  public Object getDatum() {
    return datum;
  }

  //this is sometimes called the "cdr" method
  public ListCell getNext(){
    return next;
  }

  //this is sometimes called the "rplaca" method
  public void setDatum(Object o) {
    datum = o;
  }

  //this is sometimes called the "rplacd" method
  public void setNext(ListCell l){
    next = l;
  }

 public String toString(){
   String rString = datum.toString();
   if (next == null) return rString;
   else return rString + " " + next.toString();
 }
}


class TreeCell {
  protected Object datum;
  protected TreeCell left;
  protected TreeCell right;

  public TreeCell(Object i) {
    datum = i; //left and right are null by default
```

```
  }

  public TreeCell (Object i, TreeCell l, TreeCell r) {
    datum = i;
    left = l;
    right = r;
  }

  public void setDatum(Object o) {
    this.datum = o;
  }

  public Object getDatum() {
    return datum;
  }

  public void setLeft(TreeCell t) {
    this.left = t;
  }

  public TreeCell getLeft() {
    return left;
  }

  public void setRight(TreeCell t) {
    this.right = t;
  }

  public TreeCell getRight() {
    return right;
  }

  public String toString() {
    String lString,rString;
    if (left == null)
      lString = "()";
    else
      lString = left.toString();
    if (right == null)
      rString = "()";
    else
      rString = right.toString();
    return "(" + lString + " " + datum + " " + rString + ")";
  }
}
```