CS 211	Computers and Programming Spring 2002	
	Prelim II	**April 16th, 2002**

NAME:______________________________

CU ID:_________________________

Recitation instructor/time_________________________

You have one and a half hours to do this exam.

All programs in this exam must be written in Java. Excessively convoluted code will not be graded.

****

Problem	Score
1	
2	
3	
4	
5	
6	
Total	

1. (15 points)

   (a) (10 points) Write a Java class method named *append* that takes two *non-empty* linked lists `L1` and `L2` as parameters, and updates the last cell of `L1` so that it points to the first cell of `L2`, as shown in Figure 1 below. The method does not return anything. The ListCell class from lecture is reproduced at the end of the exam. You may NOT use the Java LinkedList class.

   (b) (5 points) What is the asymptotic complexity of your algorithm, expressed as a function of n_1 and n_2 where n_1 is the number of elements in `L1`, and n_2 is the number of elements of `L2` respectively? Justify your answer briefly.

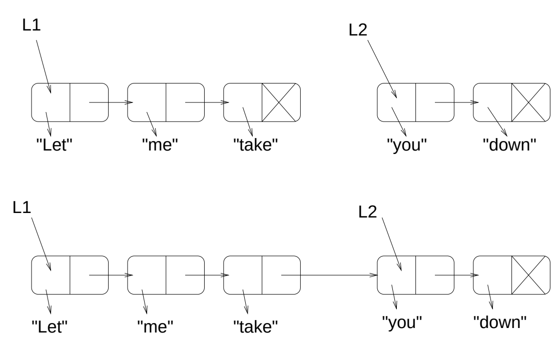


Figure 1: Appending two lists

This page intentionally left blank.

2. (15 points)

   Suppose $f_1(n) = O(g(n))$ and $f_2(n) = O(g(n))$. Answer the following questions.

   (a) Consider the function $h(n) = f_1(n) + f_2(n)$. Is $h(n) = O(g(n))$? Justify your answer formally using witness pairs (k, N) as described in class.

   (b) Consider the function $k(n) = f_1(n) * f_2(n)$. Is $k(n) = O(g(n))$? Justify your answer formally.

This page intensionally left blank.

3. (30 points)

   (a) (10 points) Write a recursive class method that computes the number of nodes in a binary tree. Assume that the root of the tree is a parameter to the method. The TreeCell class from lecture is reproduced at the end of the exam.

   (b) (2 points) Does your method do an in-order, post-order, or pre-order walk of the tree?

   (c) (3 points) What is the asymptotic complexity of your method? Explain your answer briefly.

   (d) (10 points) Write a recursive class method to print the values stored in a binary search tree. You may assume that the root of the tree is a parameter to the method. The values must be printed in *descending* order — that is, the largest value must be printed first. You may assume that each data item has a *toString* method that returns a string representation of that item.

   (e) (2 points) Does your method do an in-order, post-order, or pre-order walk of the tree?

   (f) (3 points) What is the asymptotic complexity of your method? Explain your answer briefly.

This page intensionally left blank.

4. (Short answers) (20 points)

   Time in the following questions refers to *worst-case asymptotic complexity.*

   (a) Linear search of an array requires that the array be sorted. True or false.

   (b) Linear search in a sorted array of n elements takes time ________.

   (c) Binary search in an array requires that the array be sorted. True or false.

   (d) Binary search in a sorted array of n elements takes time ________.

   (e) Quick-sort of an array of n elements takes time ________.

   (f) Merge-sort of an array of n elements takes time ________.

   (g) Insertion into a sorted list of n elements takes time ________.

   (h) Deletion from a sorted list of n elements takes time ________.

   (i) Search in a (not necessarily balanced) binary search tree of n elements takes time ________.

   (j) Deletion in a (not necessarily balanced) binary search tree of n elements take time ________.

5. (12 points) A binary tree is known to have $2^n - 1$ nodes where n is some positive integer greater than 1, but nothing else is known about its structure.

   (a) What is the smallest number of leaf nodes it can have?

   (b) What is the largest number of leaf nodes it can have?

   (c) What is the largest number of edges that can be there in a simple path from the root of the tree to a leaf?

   (d) What is the smallest number of edges that can be there in a simple path from the root of the tree to a leaf?

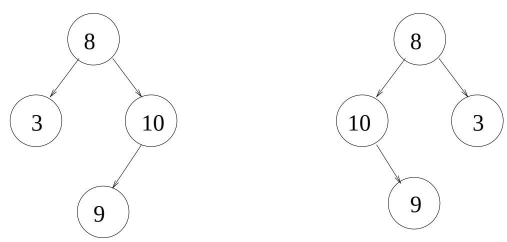
   Justify your answers.

6. (8 points) We know that a sorted array can contain values in either ascending or descending order. As defined in class, binary search trees contain values in "ascending" order — that is, the left sub-tree of a node contains values less than the value at the node, while the right sub-tree contains values greater than the value at that node.

   Define a *reverse* binary search tree to be a binary search tree that contains values in "descending" order — that is, the left sub-tree of a node contains values greater than the value at that node, while the right sub-tree contains values less than the value at that node.

   (a) Write a recursive class method to modify a binary search tree into the corresponding reverse binary search tree. Assume that the root of the tree is passed as a parameter to the method, and that the tree is represented using the TreeCell class given at the end of this exam.

   (b) Does your method perform an in-order, post-order or pre-order walk of the tree?



Binary Search Tree

Reverse Binary Search Tree

This page intentionally left blank.

```
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 + ")";
  }
}
```