COMPUTER SCIENCE 112 - FALL 2009
MIDTERM EXAM

Name: ____________________________________

TA’s Name: ____________________________________

- Be sure your test has 7 questions.
- Be sure to fill your name and TA’s name above, and your name in all subsequent pages where indicated.
- Do not begin until instructed to do so
- This is a CLOSED TEXT and CLOSED NOTES exam. You MAY NOT use calculators, cellphones, or any other electronic device during the exam.

Do not write below this line

```
Problem                              Max       Score
-------                              -----     -----

1 Lists                              10        _____

2 Binary Search Comparison Tree      15        _____

3 Binary Tree Level Order            15        _____

4 Binary Tree Height                 20        _____

5 Queue Even Split                   15        _____

6 Polynomials Multiplication         15        _____

7 Block Search                       10        _____

  TOTAL                             100        _____
```

1. **Lists (10 pts, 3+3+4)**

Compute the big O complexities of the following, *using the fastest possible algorithm for each.* Explain your algorithm as well as the derivation of the result for each. Credit will *not* be given if either of these is not present, even if you have the correct big O number. As part of your answer you *must* clearly identify the operation(s) you count towards the time. In each of these questions, the list could either be a linked list or an array.

a) Worst case time to find the common elements in two *unsorted* lists, one of length n and the other of length m.

b) Worst case time to find the common elements in two lists, one *unsorted* of length n and the other *sorted* of length m. Assume that $n < m$.

c) Worst case time to find the common elements in two lists, one of length n and the other of length m, and both are *sorted*. Assume that $n < m$.

2. **Binary Search Comparison Tree (15 pts, 5+5+5)**

a) Draw the comparison tree for binary search on an array of length 11. Be sure to include failure nodes, and to mark comparisons on the nodes and branches.

b) What is the average number (not big O) of comparisons for success, assuming equal probabilities? Show your work. (You don't have to get the answer down to a single term - no calculator required.)

c) If the entries in the array were the following:

```
10, 20, 30, 35, 40, 60, 62, 65, 70, 90, 100
```

what would be the average number (not big O) of comparisons for failure, while searching for numbers only in the range 1-100, and assuming that all failed searches are equally likely. Show your work. (Again, you don't have to simply your answer to a single term, so no calculator required).

3. **Binary Tree Level Order Traversal (15 pts,10+5)**

Given the following binary tree node and queue class definitions:

```
public class BTNode<T> {            public class Queue<T> {
   T data;                             ...
   BTNode left, right;                 public Queue() { ... }
   BTNode(T data) {                    public void enqueue(T obj) { ... }
      this.data = data;                public T dequeue() { ... }
      left=null; right=null;           public boolean isEmpty() { ... }
   }                                   public int size() { ... }
}                                   }
```

a) Complete the following method that performs a level order traversal of a binary tree.

```
// given a binary tree root, peforms a level order traversal of the tree,
// printing the data at a node when it is visited
public static<T> void levelOrderTraversal(BTNode<T> root) {
```

(b) Derive the *worst case* running time of your implementation assuming there are n nodes in the tree. Identify the basic operations, then derive the worst case total number of times these basic operations are executed as a function of n, and then convert this to big O.

4. **Binary Trees Height (20 pts, 10+10)**

Each node of a binary tree can be filled with a height value, which is the height of the subtree rooted at that node. The height of a node is the maximum of the height of its children, plus one. The height of an empty tree is -1. Here's an example, with the value in parentheses indicating the height of the corresponding node:

```
     P(3)
   /   \
  X(1)  M(2)
 /     /  \
A(0)  J(1) L(0)
      \
       K(0)
```

a) Complete the following recursive method to fill each node of a binary tree with its height value. Assume the `BTNode` definition of Question 3, with the addition of a `height` field to the class. (You can use the `Math.max(int i, int j)` method to get the maximum of two integers.)

```
// Recursively fills height values at all nodes of a binary tree
public static<T> void fillHeights(BTNode<T> root) {
```

b) A binary tree is said to be balanced if *for every node*, the difference in the heights of its children is *at most* one. (The height of an empty tree is -1.) Complete the following implementation to check if a binary tree is balanced, assuming the `height` of each node has been already filled. (You can use the `Math.abs(int i)` method to get the absolute value of an integer.)

```
// Recursive implementation to check is a binary tree is balanced
public static<T> boolean isBalanced(BTNode<T> root) {
```

5. **Queue Even Split (10 pts)**

Suppose there is a long line of people at a check-out counter in a store. Suddenly another counter opens, and people in the even positions (second, fourth, sixth, etc.) in the original line are directed to the new line. If a check-out counter line is modeled by a queue, we can implement this "even split" operation in a `Queue` class which is implemented using a circular linked list (CLL), with a `rear` field that points to the last node:

```
public Queue() { rear = null; }
public void enqueue(T obj) { ... }
public T dequeue() throws NoSuchElementException { ... }
public boolean isEmpty() { ... }
public int size() { ... }
```

Implement an additional method in this class that would perform the even split:

```
// extract the even position items (2nd, 4th, 6th, etc..) from this queue into
// the result queue, and delete them from this queue
public Queue<T> evenSplit() {
```

6. **Polynomials Multiplication (15 pts)**

This question is based on representing polynomials using linked lists. For instance, the polynomial $3x^4 + 2x^2 - 5$ would be stored in this linked list:

```
  (3,4) --> (2,2) --> (-5,0)
```

Each node of the linked list has one non-zero coefficient term stored as a (coefficient,degree) pair, and the terms are stored in **decreasing order** of degrees. For any degree that occurs in the polynomial, there will be exactly one term in that degree. Here's the `Node` class:

```
public class Node {
    public int degree;
    public float coeff;
    public Node next;
    public Node(int degree, float coeff, Node next) {
      this.degree = degree; this.coeff = coeff; this.next = next;
    }
}
```

The following method adds two polynomials and returns a new result polynomial (the input polynomials are NOT modified), and has been implemented for you:

```
// poly1 and poly2 respectively point to the first nodes of two polynomials
// that are added, and the result is a polynomial, for which a
// reference to its first node is returned (or null if the result is zero)
static Node add(Node poly1, Node poly2) { ... }
```

Implement a method to multiply two polynomials, using this `add` method. Your method should NOT change the input polynomials, and should return a reference to the first node of the result polynomial. Complete the following implementation. You may implement additional helper methods if you need to.

```
static Node multiply(Node poly1, Node poly2)
```

7. **Block Search (10 pts)**

Block Search (10 pts, 4+6)

An alternative algorithm for searching on a **sorted** integer array A of size n works as follows. It divides the array into c contiguous blocks each of size s. For simplicity you may assume that s divides n without remainder, i.e. $n = c * s$. Here is the code to search for a key k in array A:

```
boolean blockSearch(int[] A, int c,
                    int s, int k) {
    for (int i=1; i <= c; i++) {
        if (k == A[i*s-1])
            return true;
        if (k < A[i*s-1]) {
            return
            seqSearch(A, k, (i-1)*s, i*s-2);
        }
    }
    return false;
}
```

```
boolean seqSearch(int[] A, int k,
                    int lo, int hi) {
    for (int i=lo; i <= hi; i++) {
        if (k == A[i]) return true;
    }
    return false;
}
```

a) What is the worst case number (not big O) of searches for success? Your answer can be in terms of c, s, and n. Show your work.

b) What is the average number (not big O) of searches for success, assuming equal probabilities of search for all items? Your answer can be in terms of c, s, and n. Show your work. (You don't have to simplify down to a single term.)

SCRATCH PAGE

SCRATCH PAGE