COMPUTER SCIENCE 112 - SPRING 2008
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 (Linked Lists)	15	_____
2 (Queue)	15	_____
3 (Modified Sequential Search)	15	_____
4 (Postfix Expression Evaluation)	15	_____
5 (Binary Search Comparison Tree)	15	_____
6 (Delete-Minimum Data Structure)	15	_____
7 (Stack-based Count Nodes)	10	_____
TOTAL	100	_____

1. **Linked Lists (15 pts)**

Given two **sorted** linked lists L1 and L2, find the common elements. You may assume that neither of L1 and L2 has any duplicate items. Build a new linked list of the common elements **- do not** change L1 or L2. Implement helper methods as necessary - you may not call any method that you have not fully implemented. Assume the following **Node** class:

```
public class Node<T extends Comparable<T>> {
   public T data;
   public Node<T> next;
   public Node(T data, Node<T> next) {
       this.data = data; this.next = next;
   }
}
```

```
// Returns a reference to a new linked list that holds all entries
// common to L1 and L2, null if result is empty. Neither L1 nor L2 is changed.
public static <T extends Comparable<T>> Node<T> commonElements(Node<T> L1, Node<T> L2) {
  /* COMPLETE THIS METHOD */
```

2. **Queue (15 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 the Queue ADT, we can implement this “even split” operation in the Queue. Assume that a Queue class is implemented using a circular linked list (CLL), with a rear field that refers to the last node in the queue CLL, and that the Queue class already contains the following constructors and methods:

```
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:

```
// Extracts the even position items from this queue into
// the result queue, and deletes them from this queue.
// Returns null if the result is empty.
public Queue<T> evenSplit() {
  /** COMPLETE THIS METHOD **/
```

3. **Modified Sequential Search (10 pts, 4+3+3)**

Given the following (modified) sequential search code for searching in a list that is sorted in increasing order of values :

```
public boolean search(int[] A, int target) {
   for (int i=0; i < A.length; i++) {
      if (target == A[i]) return true;
      if (target < A[i]) return false;
   }
   return false;
}
```

a) Derive an exact formula in n (**not** big O) for the average number of target-to-element comparisons for successful searches on an array of size n, assuming that all possibilities of successful match are equally like. Show your work:

b) Derive an exact formula in n (**not** big O) for the average number of comparisons for failed search on an array of size n, assuming equal likelihood of failure at all possible failure locations?

c) Under what conditions would the modified sequential search be better than the regular sequential search? Explain.

4. **Postfix Expression Evaluation (15 pts)**

Implement a method to evaluate a postfix expression. The expression is a string which contains either single-digit numbers (0–9), or the operators +, -, *, and /, and nothing else. There is exactly one space between every two characters. The string has no leading spaces and no trailing spaces. For example:

```
3 5 + 2 * 6 / 8 -
```

Write helper methods (with full implementation) as necessary. You may not call any method that you have not implemented yourself.

```
public static float postfixEvaluate(String expr) {
  /* COMPLETE THIS METHOD */
```

5. **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.)

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.

6. **Delete-minimum Data Structure (15 pts, 5+5+5)**

An application needs to store a set of `Comparable` objects in a data structure. The structure must be dynamic, allowing for inserts and deletes. However, every delete is a special kind of delete in that the object with the **minimum value** is removed from the structure.

a) Devise the most efficient structure (and supporting algorithms) you can that will satisfy these requirements. Clearly describe your structure/algorithms. Contrast your structure/algorithms with other possible structures, to show that yours is more efficient.

b) Analyze the **worst case** big O running time for an insert, and for a delete - clearly specify the unit cost operations counted toward the running time.

c) If you knew that deletes were far fewer than inserts, would you change how your structure works to make it more efficient? If so, describe the changes, and explain how they would make your structure more efficient. If not, explain how your original structure+algorithms would still be most efficient.

7. **Stack-based Count Nodes (10 pts)**

Implement a stack-based non-recursive method to count the number of nodes in a binary tree. Assume the following stack and binary tree node classes:

```
public Stack() {...}
public void push(T obj) {...}
public T pop() throws
  NoSuchElementException {...}
public boolean isEmpty() {...}
public int size() {...}
```

```
public class BTNode<T> {
   public T data;
   public BTNode<T> left, right;
   public BTNode(T data) {
      this.data = data;
      this.left = null;
      this.right = null;
   }
}
```

```
   // Returns the number of nodes in a binary tree, given its root.
   // Implemented using stack-based, non-recursive code.
   public static <T> int countNodes(BTNode<T> root) {
      /* COMPLETE THIS METHOD */
```