# COMPUTER SCIENCE 112 - SPRING 2013
# FIRST MIDTERM EXAM

Name: ______________________________

CIRCLE your recitation: T 5:15   T 6:55   W 3:35   W 5:15   W 6:55   Th 6:55

- Be sure your test has 4 questions.
- **DO NOT TEAR OFF THE SCRATCH PAGES OR REMOVE THE STAPLE.**
- Be sure to fill your name and circle your recitation time above, and your name in all subsequent pages where indicated.
- 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 Circular Linked Lists | 15 | ______ |
| 3 Binary Search Comparison Tree | 15 | ______ |
| 4 Sorted Array Insertion | 15 | ______ |
| TOTAL | 60 | ______ |

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

Implement a (NON-RECURSIVE) method to find the common elements in two sorted linked lists, and return the common elements in sorted order in a NEW linked list. The original linked lists should not be modified. The new linked list should have a complete new set of Node objects (not shared with the original lists). Example:

```
L1: 3->9->12->15->21
L2: 2->3->6->12->19

Result: 3->12
```

You may assume that neither of the original lists has any duplicate items.

```
public class Node {
   public int data;
   public Node next;
   public Node(int data, Node next) {
       this.data = data; this.next = next;
   }
}

// Creates a new linked list consisting of the items common to the input lists
// returns the front of this new linked list, null if there are no common items
public static Node commonElements(Node frontL1, Node frontL2) {
        // COMPLETE THIS METHOD
```

## 2. Circular Linked Lists (15 pts)

Implement a (NON-RECURSIVE) method to delete ALL occurrences of an item from a circular linked list. Your method should return a reference to the last node of the resulting list.

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

// Deletes all occurrences of item from a circular linked list (last node
// pointed to by rear), without using recursion
// Returns the rear of the resulting linked list
// If the item is not in the list, throws a NoSuchElementException
public static <T> Node<T> deleeteAll(T item, Node<T> rear)
throws NoSuchElementException {
    // COMPLETE THIS METHOD
```

### 3. Binary Search Comparison Tree (15 pts, 7+4+4)

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.

## 4. Sorted Array Insertion (15 pts, 11+4)

a) Implement the fastest possible algorithm to insert a new entry into a sorted (in ascending order) array of strings. Duplicates are NOT allowed - throw an IllegalArgumentException if a duplicate is attempted to be inserted. After insertion, the array should still be in sorted order. You will get at most half the credit if your algorithm is not the fastest possible. (Fastest here refers to the real clock time, not big $O$, for large values of $n$).

```
// Inserts a string into a sorted array A, containing n entries, where n is
// strictly less than the length of the array. (There are more spaces in the
// array than entries.) Throws an IllegalArgumentException if string already exists
// (case insensitive match).
// After the insertion, the array is still sorted.
public static void sortedInsert(String[] A, int n, String item) {
```

b) What is the *worst case* big $O$ running time for your implementation? Identify the basic operations and show how they add up to the running time. (For any of the search algorithms done in class, you may assume its known running time without derivaton.) You will not get any credit without an adequate derivation, even if your answer is correct.

# SCRATCH PAGE

**SCRATCH PAGE**

**SCRATCH PAGE**