CS211 Prelim 2. 20 April 2004 NAME_____ NETID_____

This is a 90-minute prelim. There are 6 questions (counting question 0) and 6 pages in total. Be sure to answer all questions. Write clearly and show all your work. It is difficult to give partial credit if all we see is a wrong answer. Use the backs of pages if necessary. You can tear pages apart; we have a stapler at the front of the room.

Question 0. (2 points). Please write your name and netid at the top of each page.

Question 1. Quicksort (18 points).
In class, we developed a quicksort algorithm, and it was given on the handout for recitation 8.

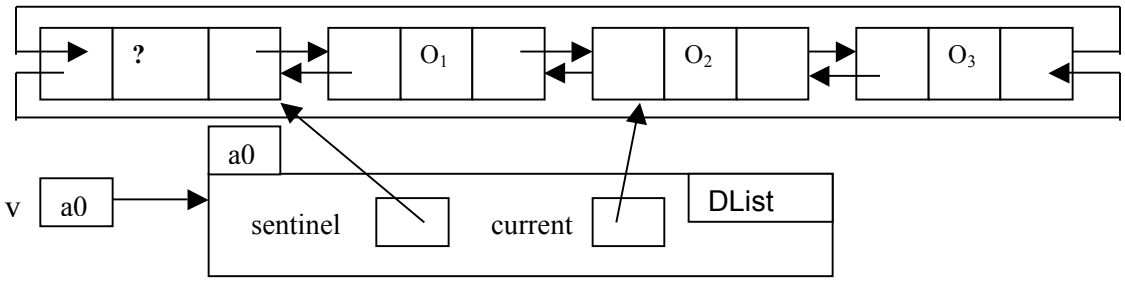
(a) What are the best-case and worst-case time complexities of this quicksort algorithm?

(b) Write the body of quicksort, below. Your algorithm need not be the one that uses only space O(log(k+1-h)), but it must be similar to one we did in class. Quicksort calls other methods. Declare the methods that quicksort calls, with good specifications for them, but **do not write the method bodies**.

```
/** sort b[h..k] in ascending order using quicksort: . */
public static void quicksort(int[] b, int h, int k) {

}
```

Question 2. Doubly linked lists (30 points).

Implement a variant of a doubly linked list with header. Here is an example of such a list, with three items O_1, O_2, O_3.



A variable v that represents a list is of class DList. An instance of DList has a field sentinel that is the name of (or a pointer to) the header node, which is an instance of DNode. The header always exists. *The value of the header node is irrelevant to this question and should not be tested*. If the list of items is not empty, field current contains the name of a node for some item.

Class DNode is used for the sentinel node and for the items of the list. Together, *these nodes form a circular doubly linked list*. Each node has the three fields shown below. These fields may be public, since DNode will be a private inner class of DList.

- prev: the previous node in the circular list
- value: the item that this node contains (or **null** for the sentinel)
- next: the next node in the circular list

If the list of items is empty, then sentinel.prev = sentinel and sentinel. next = sentinel, and sentinel = current.

Below, we give a skeleton of class DList, showing its fields, specifying some of its methods but leaving the method bodies empty, and showing where inner class DNode would go. Your task is to:

(a) Fill in inner class DNode (on the next page). Put whatever fields and methods you think are necessary. There should be a constructor. Remember, the fields can be public.

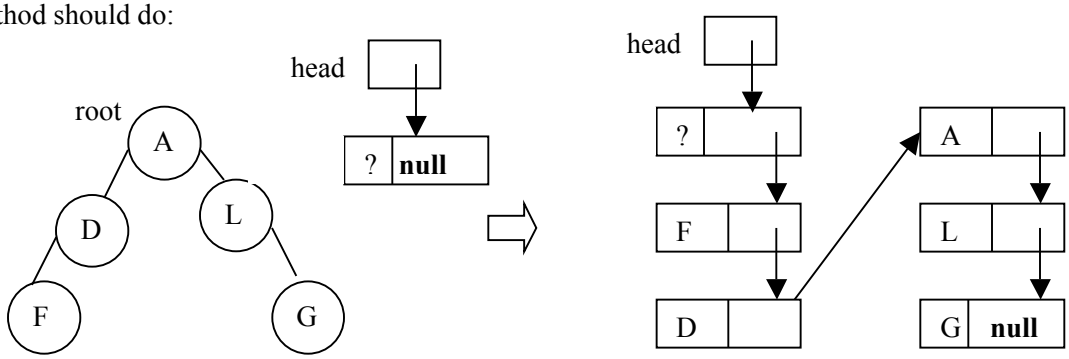
(b) Write the bodies of the methods in class DList.

```
/** an instance is a doubly linked list of item (with header) in which one
    item (if there are any) is designated as the current one. */
public class DList {
    /** Class invariant: sentinel is the head node. Its next and prev fields are the first and
          last nodes of the list of items –or sentinel itself if the list of items empty.
          If the list of items is empty, then current = sentinel; otherwise
          current is a node that contains some item of the list. */
    private DNode sentinel;
    private DNode current;

    /** Constructor: an empty list */
    public DList(){// Fill in this method body




    }   // Note: class DList is continued on the next page.
```

```
// Note: This is class DList, continued from the previous page.
    /** Insert item i into list after the current item and make the inserted item the current
        one (if the list of items is initially empty, then make a one-element list) */
    public void insert(Object i){









    }

    /**  If the list of items is empty, throw a NoSuchElementException.
         Otherwise, remove the current item from the list. If there was an item before the
        removed one, make it the new current one; otherwise, if there was an item after
        the removed one, make it the current item. */
    public void remove(){









    }

    /** An instance is a node of a doubly linked list */
    private class DNode {

















    }

}
```

Question 3. Trees, Lists, and Recursion (20 points).

Write a recursive method (the complete spec is given below) that creates a list of items in a binary tree by visiting the nodes of the tree using an in-order traversal. Here is an example of what a call of the method should do:



Use the following classes for the nodes of the tree and the list. We have made all fields public so that you can access them directly.

```
/**tree node*/
public class TNode {
    public Object data;     // data in this node
    public TNode left;      // left subtree
    public TNode right;     // right subtree
 }
```

```
/** list node*/
public class LNode {
   public Object item; // data in this node
   public LNode next; // next node.
}
```

Write the body of this method. Do not use loops. Be sure to read the specification carefully.

```
/**  Precondition: head is not null, but head.next is null. Tree root is not empty (so root != null)
     Append the in-order listing of the nodes of tree root to head and
     return the (name of or a pointer to the) last node appended */
public static LNode inorder (TNode root, LNode head){




}
```

Question 4 Miscellaneous (15 points).

(a) What does mean for function f(n) to be O(n)? State the formal definition.

(b) What is the worst-case order of execution time for the method you wrote in question 3? Please state informally why this is so. You need not prove the result formally.

(c) Heapsort uses a data structure called a *heap*. State the definition of a heap.

Question 5 Hashing (15 points). When discussing hashing, we gave a scheme in which each element of array b was either **null** or contained an object with the two following fields:

```
Integer element;    // a value
boolean isInSet;    // = "this element is in the set"
```

(a) What does it mean to use linear probing?

Question 0	/02
Question 1	/18
Question 2	/30
Question 3	/20
Question 4	/15
Question 5	/15
Total	/100

(b) Suppose the hash table currently looks like as shown below, where we use an array b[0..6] and we write an object v in the form (v.element, v.isInSet), e.g. b[1].element is 1 and b[1].isInSet is **true**.

0	1	2	3	4	5	6
null	(1, true)	(8, true)	**null**	**null**	**null**	(13, true)

Using the hash function h(x) = x mod 7 (or x % 7 since we use only positive integers in this example), draw the hash table after **each** of the following operations are performed one after the other. We want to see three diagrams. So, copy the above diagram and change it while executing (a). Then copy the result of (a) and change it while executing (b). Then, copy the result of (b) and change it while executing (c).

(a) add the integer 11
(b) remove the integer 8
(c) add the integer 15