UNIVERSITY OF TORONTO
Faculty of Arts and Science

December Examinations 1999

CSC148F: Introduction to Computer Science

Duration: 3 hours

Aids allowed: None.

- **Make sure your examination booklet has 13 pages (not including this one).**
- Write your answers in the spaces provided. Do not feel that you must use all of the space provided. If you run out of space on any question, use the back of a page, and draw an arrow to point this out.
- You will be rewarded for answers that are concise and well thought out, rather than long and rambling.
- Write legibly. Unreadable answers will be given 0.
- When asked to write code, comments are not necessary.

Family Name: ____________________ Given Names: ____________________

Student #: ____________________

Lecturer (circle one):	Chechik (morning)	Horton (afternoon)	Reiter (evening)

1. ________	/ 12
2. ________	/ 6
3. ________	/ 16
4. ________	/ 14
5. ________	/ 14
6. ________	/ 7
7. ________	/ 8
8. ________	/ 14
9. ________	/ 9
Total ________	/ 100

Good Luck!

Question 1

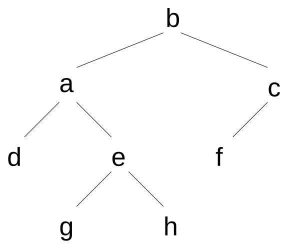
[12 marks in total]

(a) [2 marks]
Write your name and student number **legibly** at the top of *every* page of this exam.

(b) [2 marks]
Suppose you have an array `A` which you wish to treat as "circular", and an index `i` into that array. Complete the following line of code so that it advances `i` to the "next" position in the circular array.

```
    i =

                (i % A.length) + 1;
```

(c) [2 marks]
Here is a binary tree:



Circle one answer to complete the following inorder traversal of this tree: a b c ...

1. ... d a e g e h f c
2. ... g e h d a e f c
3. ... d e f g h
4. ... g h d e f
5. [✓ 5.] The sequence cannot be completed to form an inorder traversal of this tree.

(d) [2 marks]
Give a preorder traversal for the same tree: `b a d e g h c f`

(e) [4 marks]
Assume that you have a tree, represented using nodes and references. For each of the following operations, circle one answer to indicate whether it can be done elegantly without recursion (*i.e.*, without having to build your own stack), or whether it needs recursion to be done elegantly.

print the contents of a binary tree:	Elegant without recursion	[✓ Needs recursion]
print the contents of a BST:	Elegant without recursion	[✓ Needs recursion]
find the maximum value in a binary tree:	Elegant without recursion	[✓ Needs recursion]
find the maximum value in a BST:	[✓ Elegant without recursion]	Needs recursion

Continued

Question 2

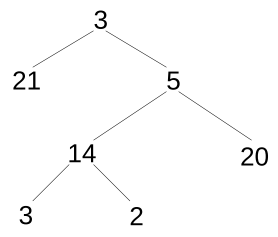
[6 marks in total]

Consider the following method, and the `BTNode` class it uses. Note that `Math.max(a, b)` returns the maximum of `a` and `b`.

```
// Pre-condition: All nodes in the tree
// rooted at r contain Integers.
public static int mystery( BTNode r ) {
     if (r == null)
          return 0;
     else {
          int left = mystery( r.left );
          int right = mystery( r.right );
          return Math.max(left, right)
              + ((Integer)r.data).intValue();
     }
}
```

```
class BTNode {
    public Object data;
    public BTNode left, right;
}
```

(a) [2 marks]
What value is returned if we call `mystery` with a reference to the root of this binary tree:



ANSWER: 28

(b) [2 marks]
How many calls to `mystery` occur, in total, as a result of that initial call to `mystery`?

ANSWER: 15

(c) [2 marks]
Write an appropriate comment that describes what `mystery` returns.

```
mystery(r) returns the largest sum attainable by adding the values
in the nodes on a path from the root to a leaf in the tree rooted at r.
```

Continued

Question 3

[16 marks in total]

For each of the following methods, write an appropriate pre-condition and post-condition, or write "none" if there is none. Don't bother stating pre-conditions that are enforced by the compiler. For example, for an `int` parameter, you don't need to say "i must be an integer".

ANNOUNCEMENT: Write $\boxed{\text{all}}$ appropriate pre- and post-conditions.

(a) [4 marks]

```
// Pre-condition: A is not null and
                  0 <= n <= A.length


// Post-condition: Sets values of A[0..n-1] to v


public static void sneezy (int[] A, int n, int v) {
     for (int i=0; i<n; i++)
         A[i] = v;
}
```

```
An alternative acceptable solution is to omit the "0 <= n" part of the
precondition.  If n is 0 or less, the post-condition is trivially true.
```

(b) [4 marks]

```
// Pre-condition: p refers to a node in a non-empty circularly-linked list.


// Post-condition: returns the number of nodes in the list.


public static int happy (Node p) {
    Node temp = p.next;
    int n = 1;
    while (temp != p) {
        n++;
        temp = temp.next;
    }
    return n;
}
```

Continued

(c) [4 marks]

```
// Pre-condition: A is not null.


// Post-condition: returns the position of the 1st occurrence of n in A,
                   or -1 if it doesn't occur in A.


public static int sleepy (int[] A, int n) {
    for (int i=0; i<A.length; i++) {
          if (A[i] == n)
               return i;
    }
    return -1;
}
```

(d) [4 marks]

```
// Pre-condition: n occurs in A.


// Post-condition: returns the position of the last occurrence of n in A.


public static int bashful (int[] A, int n) {
    int i = A.length-1;
    while (A[i] != n)
          i--;
    return i;
}
```

Continued

Question 4

[14 marks in total]

Consider the following ADT: A **sequence** is a container that holds items. It has no fixed size, and any position in which no value has been stored is considered to contain `null`. The operations that can be performed on a sequence are:

- set(i, item): sets the ith position of the sequence to item.
- get(i): returns the value in the ith position of the sequence, or null if nothing has been stored at that position.

The positions in a sequence are counted from zero. Thus, the first position is position zero.

Here is a Java interface for sequences:

```
interface Sequence {
    // Sets the i-th position of the sequence to item.
    public void set (int i, Object item);

    // Returns the value in the i-th position of the sequence, or
    // null if nothing has been stored at that position.
    public Object get (int i);
}
```

CLARIFICATION: get() has the precondition that $i \geq 0$

(a) [5 marks]

Suppose we have a class called `LinkedSequence` that implements `Sequence`. Complete the following method, which is to go *outside* the `LinkedSequence` class. It should return the item just before the first `null` value in `seq`, or `null` if the first position of the sequence holds the value `null`.

```
public static Object beforeNull (LinkedSequence seq) {

   Object previous = seq.get(0);
   if (previous == null)
        // The very first element is null, so return null.
        return null;
   else {
        // Otherwise, go looking for the first null, starting from the
        // second position (position 1).  Remember what's before, so we
        // can return it.
        Object next = seq.get(1);
        int n = 3;
        while (next != null) {
                previous = next;
                next = seq.get(n);
                n++;
        }
        return previous;
   }
}
```

Continued

Suppose class `LinkedSequence` implements `Sequence` using a singly-linked list. It begins with an empty linked list, and adds new nodes on the end whenever necessitated by the `set` method. Below is an outline for the class, as well as the `Node` class that it uses:

```
class LinkedSequence implements Sequence {

    // The first node in the linked list.              class Node {
    // Contains the first element of the sequence.         public Object data;
    private Node first;                                    public Node next;
                                                       }
    // Methods would go here.
}
```

(b) [5 marks]
Write the `get` method for class `LinkedSequence`. Do not add any instance variables to class `LinkedSequence`.

```
    // Returns the value in the i-th position of the sequence, or
    // null if no object has been stored at that position.
    public Object get (int i) {

        int n = 1;
        Node temp = first;
        while (n < position) {

                if (temp == null) {
                        break;
                }
                else {
                        temp = temp.next;
                        n++;
                }
        }
        if (temp == null)
                return temp;
        else
                return temp.data;
    }
```

Continued

(c) [2 marks]
Given what you now know about class `LinkedSequence`, what is the worst-case big-oh time complexity of your `beforeNull` method? Make sure you define any variables that you use.
You can answer this question even if you haven't solved part (a) or (b).

ANSWER: $O(n^2)$, where n is the number of elements before the first null.
Actually, the answer should be $O(n^2 + 1)$, so that if n is 0, it's $O(1)$ not $O(0)$.

(d) [2 marks]
Suppose the `beforeNull` method were to go *inside* class `LinkedSequence` instead of outside. This would allow us to write a faster version of the method. What would the worst-case big-oh time complexity of the method be? Make sure you define any variables that you use. Do not assume the existence of any instance variables other than `first`.

ANSWER: $O(n)$

Continued

Question 5
[14 marks in total]

(a) [2 marks]
Write a fragment of code that takes $O(n^2)$ time but not $O(n)$ time.

```
for (int i=0; i<n; i++)
   for (int j=0; j<n; j++)
       k++;
```

(b) [2 marks]
Write a loop that takes $O(1)$ time.

```
for (int i=0; i<5000; i++)
    k++;
```

(c) [2 marks each]
What is the worst-case big-oh time complexity of each of the following problems? Assume that the problems are solved efficiently and that there are no restrictions on the algorithms that may be used.

Search an unsorted array of n elements for a given value.
ANSWER: $O(n)$

Search a sorted linked list of n elements for a given value.
ANSWER: $O(n)$

Search a binary search tree with n nodes for a given value.
ANSWER: $O(n)$

Determine whether or not a sorted array of n elements has any duplicate values.
ANSWER: $O(n)$

Determine whether or not an unsorted array of n elements has any duplicate values.
ANSWER: $O(n \log n)$

Continued

Question 6
[7 marks in total]

Consider the following method:

```
   public static int huh( int[] A, int start, int finish ) {

       System.out.println("Range: " + start + " " + finish);
       int ans;
       if (start == finish)
          ans = A[start];
       else {
          int middle = (start + finish) / 2;
          int one = huh(A, start, middle);
          int two = huh(A, middle+1, finish);
          System.out.println("Results: " + one + " " + two);
          ans = Math.max( one, two );
       }
       System.out.println("Ans: " + ans);
       return ans;
   }
```

Assume that we have an array `A` of size 4, as shown below. What would be the output if we called `huh(A, 0, 3)`?

A	98	4	222	−1
	0	1	2	3

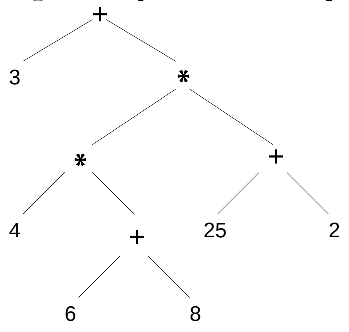
```
Range: 0 3
Range: 0 1
Range: 0 0
Ans: 98
Range: 1 1
Ans: 4
Results: 98 4
Ans: 98
Range: 2 3
Range: 2 2
Ans: 222
Range: 3 3
Ans: -1
Results: 222 -1
Ans: 222
Results: 98 222
Ans: 222
```

Continued

Question 7

[8 marks in total]

Suppose we have a binary tree in which each leaf stores a `Double` and each internal node stores an operator, either `+` or `*` (represented as a `String`). Such a tree can represent simple arithmetic expressions. For example, the following tree represents the expression $(3+((4*(6+8))*(25+2)))$:



Complete the following method for computing the value of an expression represented by such a tree. It uses the same `BTNode` class as on page 2.

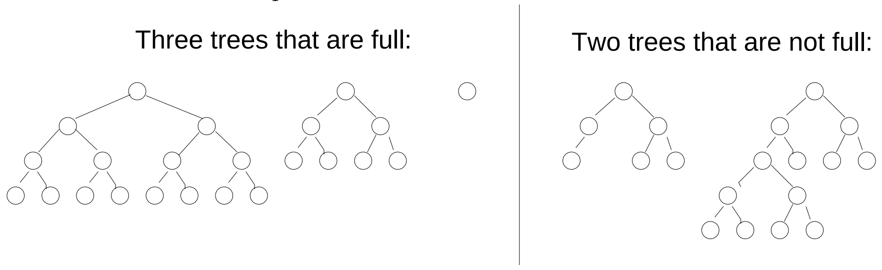
```
     // Pre-condition: root is not null, every leaf in the tree is a Double,
     //      and every internal node contains either "+" or "*".
     public static Double value( BTNode root ) {

        double answer = 0.0;

        if (root.data instanceof Double)
                answer = ((Double)root.data).doubleValue();

        else if (((String)root.data) == "+")
                answer = value(root.left).doubleValue() +
                         value(root.right).doubleValue();

        else if (((String)root.data) == "*")
                answer = value(root.left).doubleValue() *
                         value(root.right).doubleValue();

        return new Double(answer);
     }
}
```

Continued

Question 8

[14 marks in total]

A **full** binary tree is one in which every node except the leaves has exactly two children, and all leaves are on the same level. Examples:



Recall that the height of a tree is the number of nodes on the longest path from the root to a leaf.

(a) [2 marks]
Suppose we were going to prove that for all $h \geq 1$, a full binary tree of height h has $2^h - 1$ nodes. Would it be best to use weak or strong induction? Circle one.

[✓ weak] strong

Explain why. Be specific.

```
Because in a complete tree (unlike a regular tree), we are sure that
the height of each subtree is exactly one less than the height of the
tree itself.  So if make a strong induction assumption, the induction
step will not make use of more than the last of the assumptions.
```

(b) [8 marks]
Now do the proof. Write your proof on the next page.

(c) [2 marks]
Suppose we have a full binary tree with n nodes. What can you assert about the value of n?

```
n must be one less than some power of 2
(and that power of 2 must be greater than or equal to 1).
```

(d) [2 marks]
How many leaves does a full binary tree with n nodes have?

```
(n+1)/2
```

Continued

Proof for part (b):

Prove: For all $h \geq 1$, a full binary tree of height h has $2^h - 1$ nodes.

Base Case: Prove that a full binary tree of height 1 has $2^1 - 1$ nodes.

- Consider a full binary tree of height 1.
- Such a tree has exactly 1 node (by definition of height).
- And $2^1 - 1 = 2 - 1 = 1$.
- So a full binary tree of height 1 has $2^1 - 1$ nodes.

Let k be an arbitrary integer ≥ 1.

Induction Hypothesis: Assume that a full binary tree of height k has $2^k - 1$ nodes.

Induction Step: Prove that a full binary tree of height $(k + 1)$ has $2^{k+1} - 1$ nodes.

- Consider a full binary tree of height $(k + 1)$.
- Since $k \geq 1$, $(k + 1) \geq 2$, so the tree has height at least 2.
  This means it has at least one child.
- But since it is a *complete* tree, if it has one child, it must have two.
- And since it is a complete tree, both its subtrees have height exactly k.
- By the induction hypothesis, these subtrees therefore each have $2^k - 1$ nodes.
- So in total, the tree has 1 node for the root, plus $2 \times (2^k - 1)$ nodes for its subtrees.
- $1 + 2 \times (2^k - 1) = 1 + 2^{k+1} - 2 = 2^{k+1} - 1$.
- So a full binary tree of height $(k + 1)$ has $2^{k+1} - 1$ nodes.

Continued

Question 9

[9 marks in total]

Here are interfaces for `Stack` and `Queue`:

```
public interface Queue {
    // Append o to me.
    public void enqueue(Object o)
            throws QueueFullException;
    // Remove and return my first Object.
    public Object dequeue();
            throws QueueEmptyException;
    // Return true if I am empty.
    public boolean isEmpty();
}
```

```
public interface Stack {
    // Push o onto me.
    public void push(Object o)
            throws StackFullException;
    // Pop off my top object and return it.
    public Object pop()
            throws StackEmptyException;
    // Return true if I am empty.
    public boolean isEmpty();
}
```

Write a non-recursive method to print a linked list in reverse order. Its only parameter should be the first `Node` in the linked list. Use the `Node` class on page 6.

Assume that you have a class `MyStack` that implements `Stack`, and a class `MyQueue` that implements `Queue`. You may find these classes to be useful in your solution. You need not catch any exceptions that are thrown.

```
    public static void backwards( Node front )
                            throws StackEmptyException, StackFull Exception {

        ArrayStack s = new ArrayStack();

        Node temp = front;
        while(temp != null) {
                s.push(temp);
                temp = temp.next;
        }

        while (!s.isEmpty()) {
                temp = (Node)s.pop();
                System.out.println(temp.data);
        }
}
```