UNIVERSITY OF TORONTO
Faculty of Arts and Science

December Examinations 1997

csc148f

Duration — 3 hours

Aids allowed: None.

- **Make sure your examination booklet has 17 pages (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 concise, well-thought-out answers, rather than long rambling ones.
- Write legibly. Unreadable answers will be given 0.

Family Name: ______________________ Given Names: ______________________

Student #: ______________________ Lecturer: ______________________

1.	________	/ 5
2.	________	/ 4
3.	________	/ 10
4.	________	/ 13
5.	________	/ 12
6.	________	/ 7
7.	________	/ 11
8.	________	/ 7
9.	________	/ 19
10.	________	/ 12
Total	________	/ 100

Question 1

[5 marks]

Assume that the following Node class has already been defined for making nodes in a doubly linked list:

```
class DNode {
    public String s;
    public Node next, previous;
}
```

Complete the `remove` method begun below. Its task is to remove the last k nodes from the doubly-linked list whose last node is referred to by `end`. The method assumes that there are at least $(k+1)$ nodes in the list to begin with.

Part of the marks are for making sure that the removed nodes are not pointed to by any node in the doubly-linked list. (This will ensure that they will eventually be garbage collected.)

```
// Precondition: There are at least (k+1) nodes in the list.

public static void remove (Node end, int k) {
```

Question 2

[4 marks in total]

a. [2 marks]
Is it a good idea or a bad idea to implement a stack using a doubly-linked list? Circle one answer.

Good idea Bad idea

Explain your answer:

b. [2 marks]
Say we are implementing a stack as a linked list (singly- or doubly-linked), and that we're keeping a reference to the node representing the top of the stack. Is it a good idea or a bad idea to keep an additional reference to the bottom of the stack? Circle one answer.

Good idea Bad idea

Explain your answer:

Question 3

[10 marks in total]

Assume that the following class has been defined.

```
class Floop {
   public int silly;

   public boolean equals (Floop another) {
           return (silly == another.silly);
   }
}
```

If we run the `Tester` class which is written on the next page (*i.e.* if we run its `main` method), it runs without crashing. Show what the output would be.

```
class Tester {
   public static void reset (Floop f) {
         f.silly = 0;
   }
   public static void increment (Floop f) {
         f.silly ++;
   }
   public static void reRoute (Floop f) {
         f = new Floop();
         f.silly = -47;
   }
   public static void main (String[] args) {
        Floop f1 = new Floop();
        f1.silly = 44;
        Floop f2 = new Floop();
        f2.silly = 3;
        Floop f3 = f1;
        f3.silly = 3;
        System.out.println (f1.silly + " " + f2.silly + " " + f3.silly);

        if (f1==f2) System.out.println("f1 and f2 ==");
        if (f2==f3) System.out.println("f2 and f3 ==");
        if (f1==f3) System.out.println("f1 and f3 ==");
        if (f1.equals(f2)) System.out.println("f1 and f2 equals");
        if (f2.equals(f3)) System.out.println("f2 and f3 equals");
        if (f1.equals(f3)) System.out.println("f1 and f3 equals");

        f1.silly = 1;
        reset(f1);
        System.out.println ("After reset:" + " " + f1.silly);
        f1.silly = 1;
        increment(f1);
        System.out.println ("After increment:" + " " + f1.silly);
        f1.silly = 1;
        reRoute(f1);
        System.out.println ("After reRoute:" + " " + f1.silly);
   }
}
```

OUTPUT:

Question 4

[13 marks in total]

Consider the following code fragment that finds the length of the longest run of consecutive equal elements in the array A between positions 0 and $n-1$ inclusive.

```
int longestSoFar = 1;
int startOfRun = 0;
int current = 1;
while (current < n){
    if (A[current] == A[startOfRun])
         if (current - startOfRun == longestSoFar)
            longestSoFar++;
    else
         startOfRun = current;
    current++;
}
System.out.println(longestSoFar);
```

a. [3 marks]

Assume that n is 10, and that array A has been constructed and contains the following values:

0	1	2	3	4	5	6	7	8	9
0	5	5	1	2	2	2	5	1	5

Trace the code fragment and show the values of each variable as it executes, using the following table.

Variable	Value as the code executes $\cdots$
longestSoFar	
startOfRun	
current	

What would the output be?

b. [2 marks]

State necessary preconditions for this code fragment to behave properly.

c. [6 marks]
For each of the following statements, explain why it is **not** a loop invariant for the while-loop.

- `current < n`

- `(current - startOfRun) < longestSoFar`

- `longestSoFar` is the length of the longest run of consecutive equal elements in A up to and including location `current`.

d. [2 marks]
Give a valid loop invariant involving some of the program variables.

Question 5
[12 marks in total]

Consider the following algorithms (on the next page), each of which operates on an n-by-n array of integers A. We are only interested in the running times of these algorithms, not what they do. For each algorithm, circle the best upper bound on its time complexity.

Helpful information:

- sorting a list of n items using "bubble sort" takes $O(n^2)$ time
- sorting a list of n items using merge sort takes $O(n \log n)$ time.
- $\log(n^2) = 2 \log n$.

a. Sort each row of A using bubble sort, then sort each column using bubble sort.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ $O(n^2 \log n)$ $O(n^3)$ $O(n^3 \log n)$ $O(n^4)$ $O(2^n)$

b. Sort each row of A using merge sort, then sort each column using merge sort.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ $O(n^2 \log n)$ $O(n^3)$ $O(n^3 \log n)$ $O(n^4)$ $O(2^n)$

c. Sort each row of A using merge sort, then sort each column using bubble sort.

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ $O(n^2 \log n)$ $O(n^3)$ $O(n^3 \log n)$ $O(n^4)$ $O(2^n)$

d. First copy each integer in A into a 1-dimensional array B of size n^2 using the following fragment of Java code:

```
for (int r=0; r<n; r++)
    for (int c=0; c<n; c++)
        B[r*n+c] = A[r][c];
```

Then sort the array B using bubble sort and, finally, copy B back into A using the following fragment of Java code:

```
for (int r=0; r<n; r++)
    for (int c=0; c<n; c++)
         A[r][c] = B[r*n+c];
```

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ $O(n^2 \log n)$ $O(n^3)$ $O(n^3 \log n)$ $O(n^4)$ $O(2^n)$

e. First copy each integer in A into a 1-dimensional array B of size n^2 as in algorithm (d). Then sort the array B using merge sort and, finally, copy B back into A as in algorithm (d).

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ $O(n^2 \log n)$ $O(n^3)$ $O(n^3 \log n)$ $O(n^4)$ $O(2^n)$

f. This final algorithm assumes that all the integers in A are between 0 and 999 inclusive. It first copies each integer in A into a 1-dimensional array B of size n^2 but uses a different algorithm for doing so than in (d):

```
int i=0;
for (int k=0; k<1000; k++)
    for (int r=0; r<n; r++)
        for (int c=0; c<n; c++)
            if (A[r][c] == k)
                B[i++] = k;
```

This algorithm then copies B back into A as in algorithm (d).

$O(1)$ $O(\log n)$ $O(n)$ $O(n \log n)$ $O(n^2)$ $O(n^2 \log n)$ $O(n^3)$ $O(n^3 \log n)$ $O(n^4)$ $O(2^n)$

CONTINUED

Question 6

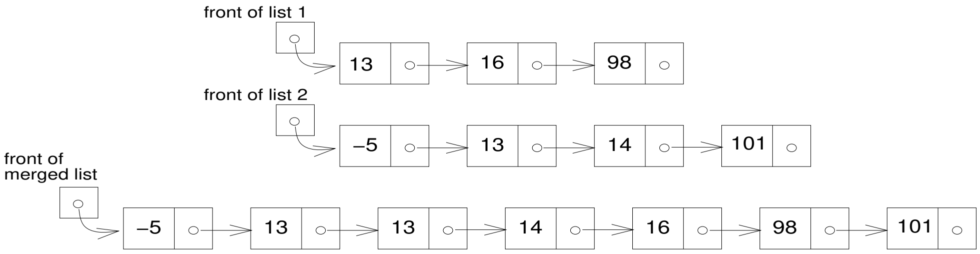
[7 marks in total]

Assume that the following Node class has already been defined for making nodes in a linked list:

```
class Node {
    public String s;
    public Node link;
}
```

Suppose that we are going to write a method whose purpose is to take two sorted linked lists called list 1 and list 2, and create a new sorted linked list that contains the combined contents of list 1 and list 2.

Say that we wish the method to make *copies* of the nodes from list 1 and list 2, rather than reuse the existing ones. For example, if list 1 and list 2 are as shown below, the method should create the bottom linked list shown.



a. [2 marks]
Should the method return the resulting linked list through the parameters, or through a return value? Circle one answer.

Through the parameters Through a return value

Why?

b. [2 marks]
Write an appropriate signature for this method.

c. [3 marks]
Let n represent the number of nodes in list 1 and m represent the number in list 2. Use big-oh notation to give a good upper bound on the time complexity of this method, assuming that a reasonable algorithm is used.

Justify your answer:

Question 7

[11 marks in total]

In Assignment 3 you built a Lexicon class. It implemented a list of words using a trie made out of the following kind of nodes:

```
class TrieNode {
   public static final int ALPHABET_SIZE = 26;

   public char letter;
   public boolean isWord;
   public TrieNode [] children;

   // Constructor.  Makes a trieNode with the given character 'c' in it,
   // with isWord set to the boolean value 'b', and with null children.
   public TrieNode (char c, boolean b) {
       letter = c;
       isWord = b;
       children = new TrieNode[ALPHABET_SIZE];
   }
}
```

a. [3 marks]
Which of the following operations can only be done easily with recursion? For each, circle the appropriate answer and then justify it.
Determining whether or not a given string is in the trie.

Can only be done easily with recursion   Can be done easily without recursion

JUSTIFICATION:

Adding a given string to the trie.

Can only be done easily with recursion   Can be done easily without recursion

JUSTIFICATION:

Determining the number of strings in the trie.

Can only be done easily with recursion   Can be done easily without recursion

JUSTIFICATION:

b. [8 marks]
Complete the following method, which returns the length of the shortest string in the trie rooted at node `r`, or 0 if there are no strings in it. **Hint:** Use recursion.

You will recall that the root node of the trie contains a blank. This character should not be considered part of any string in the trie. For example, if the shortest string in the trie is "hello", your method should return 5, even though the blank at the root might be considered to contribute an additional character.

```
public int shortest (TrieNode r) {
```

Question 8

[7 marks in total]

A *palindrome* is a string that reads the same way forwards and backwards. The following is a recursive Java method that determines if a string is a palindrome.

```
public static boolean isaPalindrome (String s) {
    if (s.length() <= 1)
        return true;
    else if (s.charAt(0) == s.charAt(s.length()-1))
        return isaPalindrome (s.substring(1, s.length()-1));
    else
        return false;
}
```

a. [5 marks]
Give a recurrence relation that expresses the number of comparisons performed by a call to `isaPalindrome` in the worst case. Define any variables that you use, and remember to include initial conditions for your recurrence.

b. [2 marks]
Make a table of values for your recurrence relation, and use it to find a solution to your recurrence relation. Do *not* prove your solution to be correct.

TABLE OF VALUES:

SOLUTION TO THE RECURRENCE RELATION:

Question 9

[19 marks in total]

Suppose you want to implement a binary search tree (BST) of integers where each node has *three* references as well as its data. Two of the references refer to the node's left and right children, as with a regular binary tree, and the third one refers to the node's parent. The node at the root of the tree should have a null parent reference. This kind of node can be described by the following Java class.

```
class NWP {        // NWP stands for "Node With Parent".
   int data;
   NWP left, right, parent;

   // constructor
   public NWP() {
      data = 0;
      left = null;  right = null;  parent = null;
   }
}
```

a. [6 marks]

Write a Java method called `insert`, which takes as parameters a reference `t` to a `NWP` node that is the root of a binary search tree, and an integer `x`. The method should insert `x` into the tree pointed to by `t`, and return a reference to the root of the revised tree. If `x` is already in the tree, the method should not insert it again; it should simply return a reference to the root. The signature of `insert` is given for you.

```
public static NWP insert (NWP t, int x) {
```

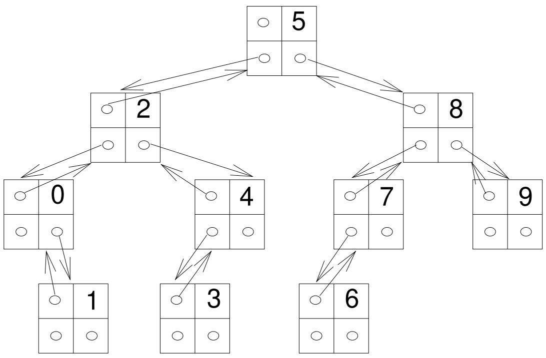
b. [1 mark]
Describe what modifications, if any, you would make to the regular BST search in order for it to work on a BST with parent pointers.

c. [2 marks]
Write a Java method, `isRightChild`, which takes a non-root node `t` from a binary tree with parent pointers, and returns true if `t` is the right child of `t`'s parent and false otherwise. The signature and specifications of isRightChild are given for you.

```
// Preconditions: t is a non-root node from a binary tree with parent pointers.
// Postconditions: returns true if t is the right child of its parent
//                 and false otherwise.
private static boolean isRightChild (NWP t) {
```

d. [10 marks]
On the next page, write a Java method called `next`, which takes as parameter a reference `n` to a node within a binary search tree with parent pointers, and returns the reference to the node containing the next largest integer to the one stored in the node pointed to by `n`. If there is no next largest integer in the tree, the method should return null.

For example, consider the following BST.



If `n` refers to node 4, then `next(n)` should return node 5.
If `n` refers to node 5, then `next(n)` should return node 6.
If `n` refers to node 9, then `next(n)` should return null.

CONTINUED

Hints:

- Consider the cases where n has and does not have a right child.
- Consider using the method in part (b). You may assume, for this part, that your method in part (b) works as specified.

The signature of `next` is given for you.

```
public static NWP next (NWP n) {
```

Question 10

[12 marks in total]

Suppose W is an array of n floating point numbers, each denoting the weight of a coin. Exactly one of the coins is fake and weighs less than the others. All real coins weigh the same amount. Assume that we already have a method with the following signature that correctly represents the action of a balance scale:

```
public static int weigh (float[] W, int i, int j, int h, int k)
```

If $0 \leq i \leq j < h \leq k \leq n-1$, then $weigh(W, i, j, h, k)$ compares the weights $W[i] + \cdots + W[j]$ and $W[h] + \cdots + W[k]$. It returns -1 if the first weight is smaller than the second; it returns 1 if the first weight is larger than the second; and it returns 0 if the two weights are equal.

The following recursive method uses `weigh` to identify the fake coin:

```
public static int findFake (float[] W, int first, int last) {
   if (last - first == 0)
       return first;
   else {
       int m = (last - first + 2) / 3;      // The '/' does integer division.
       int v = weigh (W, first, first+m-1, first+m, first+2*m-1);
       if (v == -1)
           return findFake (W, first, first+m-1);
       else if (v == 1)
           return findFake (W, first+m, first+2*m-1);
       else
           return findFake (W, first+2*m, last);
   }
}
```

For $0 \leq i \leq j \leq n-1$, let $W[i..j]$ denote the set of array elements $\{W[i], \ldots, W[j]\}$.

Let $S(r)$ denote the statement

> "If $0 \leq first \leq last \leq n-1$, $last - first = r$, and $W[first..last]$ contains the weight of the fake coin, then $findFake(W, first, last)$ returns the location containing the weight of the fake coin."

On the next two pages is a proof that $S(r)$ is true for all $r \geq 0$. Complete the proof by filling in the missing parts of the base case and induction step. You may not need to use all the space available. Do not prove that array accesses are in bounds — assume that this has already been done.

If there is a step that you need to prove but are unable to, **just say so and carry on** as if you had proven it.

Base Case: Prove that $S(0)$ is true.

Suppose that $0 \leq first \leq last \leq n-1$, $last - first = 0$, and $W[first..last]$ contains the weight of the fake coin.

Therefore if $0 \leq first \leq last \leq n-1$, $last - first = 0$, and $W[first..last]$ contains the weight of the fake coin, then $findFake(W, first, last)$ returns the location containing the weight of the fake coin.

Therefore $S(0)$ is true.

Let k be an arbitrary integer ≥ 0.

Induction Hypothesis: Assume that $S(t)$ is true for $t = 0, \ldots, k-1$.

In other words, if $0 \leq first' \leq last' \leq n-1$, $0 \leq last' - first' \leq k-1$, and $W[first'..last']$ contains the weight of the fake coin, $findFake(W, first', last')$ returns the location containing the weight of the fake coin.

Induction Step: Prove that $S(k)$ is true. That is, prove that "If $0 \leq first \leq last \leq n-1$, $last - first = k$, and $W[first..last]$ contains the weight of the fake coin, then $findFake(W, first, last)$ returns the location containing the weight of the fake coin."

Suppose that $0 \leq first \leq last \leq n-1$, $last - first = k$, and $W[first..last]$ contains the weight of the fake coin.

Since $last - first = k > 0$, the first test in the method is false, so m is assigned the value $\lfloor (last - first + 2)/3 \rfloor = \lceil k/3 \rceil$. Note that, if $k \geq 2$, then $2\lceil k/3 \rceil \leq k$.

Hence $S(k)$ is true.

By induction, $S(r)$ is true for all integers $r \geq 0$.

END OF EXAM; Total pages = (17)