PLEASE HAND IN

**UNIVERSITY OF TORONTO**
**Faculty of Arts and Science**

**DECEMBER EXAMINATIONS 2001**

**CSC 148H1 F and CSC 150H1 F**
**St. George Campus**

**Duration — 3 hours**

**Aids allowed: none**

PLEASE HAND IN

**Student Number:** |_|_|_|_|_|_|_|_|_|

**Last Name:** ______________________________

**First Name:** ______________________________

**Lecture Section:** ____________ **Instructor:** ______________

*Do **not** turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below.)

---

| | |
|---|---|
| # 1: | ______/ 10 |
| # 2: | ______/ 10 |
| # 3: | ______/ 10 |
| # 4: | ______/ 10 |
| # 5: | ______/ 10 |
| # 6: | ______/ 10 |
| # 7: | ______/ 24 |
| # 8: | ______/ 12 |
| # 9: | ______/ 10 |
| TOTAL: | ______/106 |

This examination consists of 9 questions on 14 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.* If you need more space for one of your solutions, use the reverse side of the page and *indicate clearly the part of your work that should be marked.*

Comments are not required except where indicated, although they may help us mark your answers. They may also get you part marks if you can't figure out how to write the code.

Write your student number at the bottom of pages 2-14 of this test.

*Good Luck!*

## Question 1. [10 MARKS]

Below are 3 different proofs by induction. We have left out the bodies of the proofs, because we are interested only in the structure.

Each is missing one part. Your job, in each case, is to complete the missing component. In some cases, there are multiple correct answers. Choose the simplest.

Write only in the underlined areas.

### Part (a) [2 MARKS]

Base case: Proof that `S(1)` is true.

Let `k >= 1` be an arbitrary integer.

Induction hypothesis: Assume `S(k)`.

Induction step: Proof that `S(k+2)` is true.

Conclusion: ______________________________________________

### Part (b) [4 MARKS]

Base case: Proof that ____________ is true.

Let `x >= 18` be an arbitrary integer.

Induction hypothesis: Assume ____________ is true.

Induction step: Proof that `P(x)` is true.

Conclusion: `P(x)` is true for all `x >= 17`.

### Part (c) [4 MARKS]

Base cases: Proof that `A(0)`, `A(1)`, and `A(2)` are true.

Let `i >=` ____________ be an arbitrary integer.

Induction hypothesis: Assume ________________________ .

Induction step: Proof that `A(i+3)` is true.

Conclusion: `A(i)` is true for all `i >= 0`.

## Question 2. [10 MARKS]

Prove that $3x^2 - 2x + 123,456,789$ is $\mathtt{O}(x^2)$.

## Question 3. [10 MARKS]

A *prime number* is a number that is divisible only by itself and 1. For example, the numbers 2, 3, 5, 7, and 11 are the first five prime numbers. (1 is not considered to be prime.)

Write a class `PrimeEnumeration` that implements `Enumeration`, and returns the prime numbers as a sequence of `Integer` objects, in ascending order.

**You must use the following algorithm:** keep a `Vector` containing every `Integer` you have previously returned. To find the next prime, count up from the largest of the previous primes until you find a number that is not divisible by any of the items currently in the `Vector`.

## Question 4. [10 MARKS]

Given a sequence of numbers $X = \langle x_0, \ldots, x_{n-1} \rangle$ where $n \geq 0$, we define a *suffix* of $X$ to be any sequence of the form $\langle x_i, \ldots, x_{n-1} \rangle$ where $0 \leq i \leq n$. For example, the suffixes of the sequence $\langle 1, 7, 7, 6 \rangle$ are: $\langle 1, 7, 7, 6 \rangle$, $\langle 7, 7, 6 \rangle$, $\langle 7, 6 \rangle$, $\langle 6 \rangle$ and $\langle \rangle$ (the empty sequence).

Also, given a sequence of numbers $X = \langle x_0, \ldots, x_{n-1} \rangle$, we say that $X$ is a *decreasing* sequence if $x_i > x_{i+1}$ for every $i$ where $0 \leq i \leq n-2$. For example, $\langle \rangle$, $\langle 4 \rangle$ and $\langle 9, 6, 3, 0 \rangle$ are decreasing sequences, but $\langle 5, 6 \rangle$, $\langle 8, 5, 5 \rangle$ and $\langle 4, 2, 1, 3 \rangle$ are not.

For this question, we regard a singly linked list as a sequence and we use the following `IntNode` class for a linked list of integers.

```
public class IntNode {
    public int data;
    public IntNode link;
}
```

Write the body of the following method. Full marks will be given only for an `O(`$n$`)` algorithm; an `O(`$n^2$`)` algorithm will receive a maximum of 75% of the marks.

```
// Find the longest decreasing suffix of the sequence referred to by front.
// Return the first node of the longest decreasing suffix, or null if the
// longest decreasing suffix is empty.
// Requires: front points to a valid linked list.
// Ensures: the list referred to by front is not modified.
public static IntNode longestDecreasingSuffix(IntNode front) {




}
```

## Question 5. [10 MARKS]

Recall Assignment 4, the binary search tree OrderedList assignment. The size of each subtree was stored in each node. Consider this implementation, which uses inheritance:

```
class BSTNode {
    Comparable value;
    BSTNode left, right;
}
```

```
class CounterBSTNode extends BSTNode {
    /**
     * Size of the subtree rooted at this node --starts at
     * 1 because a leaf is the root of a subtree of size 1.
     */
    int subtreeSize = 1;
}
```

Complete method `countGreaterThan`, below, without using any loops. You may write helper methods. For full marks you must use the fact that this is a binary search tree and that each subtree knows its size.

```
public class BSTWithSize {
    private CounterBSTNode root;

    /** Return the number of keys in the tree that are greater than c.
     *  Requires: c != null. */
    public int countGreaterThan(Comparable c) {
```

## Question 6. [10 MARKS]

Consider the following `Node` class for a doubly-linked list of `Comparable`s.

```
class Node {
    public Comparable data; // The value in this Node.
    public Node prev; // The previous node in the list.
    public Node next; // The next node in the list.
}
```

Write the following method **recursively.** (Without a helper method.)

```
    /** Remove all items in the *unsorted* linked list pointed to by t that
     *  are in the range [c1, c2) and return the front of the new list.
     *  (This modifies the existing list, it doesn't produce a new one.)
     *  Requires: c1 != null && c2 != null. */
    private static Node removeRange(Node t, Comparable c1, Comparable c2) {









    }
```

## Question 7. [24 MARKS]

At your new job working for JavaOS, the trendy company building an operating system totally in Java, you have been asked to create a new collection data type. The preliminary design documents call for the ability to add objects to your collection using a priority: objects will be removed based on the priority and insertion order, with the highest priority items first. (Note that this is equivalent to a queue if all the items have the same priority.)

(Perhaps the most visible reason to have your collection is a print queue: users' word processing software would add print jobs to the collection, and the print server would remove them.)

**Part (a)** [2 MARKS] What bad thing can can happen to low-priority items when using this ADT in a busy system?

**Part (b)** [12 MARKS]

In the table below, analyze the running time of insertion and removal of a *single* item using various data structures. Using big-oh notation, give the tightest bound that you can, basing your answers on the following variables:

- `n`, the number of items in the collection.
- `m`, the maximum number of items of a particular priority.
- `p`, the maximum number of priorities.

For all data structures that use arrays, assume the implementation doubles the size whenever it runs out of space, that that this doubling is `O(1)`, and that the implementation uses at most 2 indices per array.

You can assume that each item knows its priority.

| **Data structure** | **Insertion** | **Removal** |
|---|---|---|
| A singly-linked list sorted by priority and then by insertion order, with both a head and tail pointer. | | |
| A doubly-linked list sorted by priority and then by insertion order, with only a head pointer. | | |
| A binary search tree, comparing first by priority and then by insertion order. | | |
| An array sorted only by insertion order. | | |
| A linked list of arrays, where each array holds all items of a single priority in the order of insertion. The linked list has nodes for each existing priority, and is sorted by priority. | | |
| An array of singly-linked lists (without tail pointers), where each linked list holds all items of a single priority in the order of insertion. The array has one entry for each priority. | | |
| A `BucketPriorities` collection, as described on the next page. | | |

**Part (c)** [10 MARKS] In one implementation of your ADT there are only 19 priorities, 0-18. Your boss suggests that you keep "buckets" of objects, where each bucket holds objects of a particular priority, in the order in which they are inserted. (The index indicates the priority.) Each bucket should contain a singly-linked list of items (with head and tail pointer), so there are 19 linked lists.

Complete class `BucketPriorities`.

```
class Node {
    public Object data;
    public Node link;
    public Node(Object o) {
        data = o;
    }
}
```

```
class Bucket {
    public Node head;
    public Node tail;
}
```

```
public class BucketPriorities {
    // Assume the constructor initializes this to refer to 19 empty Buckets.
    private Bucket[] buckets = new Bucket[19];

    public void insert(Object o, int priority) {









    }

    public Object getNext() {









    }
}
```

## Question 8. [12 MARKS]

Consider the following two classes:

```
public class Test {
    public static void main(String[] args) {
        APair start = new APair(4, -2);                              // line 1
        start.print();                                               // line 2
    }
}

public class APair {
    private int myValue, myIncrement;
    private static int myCount = 0;
    private APair myPair = null;

    public APair(int value, int increment) {
        myCount++;                                                   // line 1
        myValue = value;                                             // line 2
        myIncrement = increment;                                     // line 3

        if (myValue > 0) {                                     // #1 // line 4
            if (myIncrement < 0) {                                   // line 5
                myPair = new APair(myValue + myIncrement, 1);        // line 6
            } else {                                                 // line 7
                myPair = new APair(myValue + myIncrement, -2);       // line 8
            }
        }
    }

    public void print() {
        if (myPair != null) {                                        // line 1
            myPair.print();                                          // line 2
            String values = "" + myValue + " " + myIncrement;        // line 3
            System.out.println(values);                        // #2 // line 4
        }
    }
}
```

### Part (a) [2 MARKS]

List the exact output the results from executing the program.

For both of the following memory model drawings you may omit any Java API classes in the static space, and you may omit anything to do with `main`'s parameter `args`.

**Part (b)** [5 MARKS]

Draw the memory model to illustrate the state of the program the *first* time the line labelled `// #1` is reached, but *before* that line has begun to execute. (The topmost line number should be 4.)

**Part (c)** [5 MARKS]

Draw the memory model to illustrate the state of the program the *first* time the line labelled `// #2` is reached, but *before* that line has begun to execute. (The topmost line number should be 4.)

## Question 9. [10 MARKS]

The following method prints the contents of a stack. Rewrite it so that it doesn't use any loops. You may use helper methods if you wish. (We recommend using 2 helper methods.)

```
public static void printStack(Stack s) {
    Stack temp = new Stack();

    // Flip s into temporary stack temp.
    while (!s.isEmpty()) {
        temp.push(s.pop());
    }

    // Flip temp back into s, printing as we go.
    while (!temp.isEmpty()) {
        Object o = temp.pop();
        System.out.println(o);
        s.push(o);
    }
}

public static void noLoopPrintStack(Stack s) {
```

## **Reference sheet** —you may detach this page

## Relevant Java APIs

All information is `public`.

```
interface Enumeration {
    boolean hasMoreElements() // = true if this Enumeration has more elements.
    Object nextElement() // the next element in this Enumeration.
}

class Integer {
    Integer(int v) // An Integer wrapping v.
    Integer(String s) // An Integer wrapping the value in s.
    int intValue() // = this Integer's wrapped value.
}

interface Comparable {
    // Return < 0 if this Comparable is less than o,
    //        = 0 if this Comparable is equal to o,
    //        > 0 if  this Comparable is greater than o.
    boolean compareTo(Object o)
}

class Vector {
    void add(Object o) // Add o to the end of this Vector.
    void add(int i, Object o) // Add o to this Vector at location i.
    Object get(int i) // Return the item at location i in this Vector.
    Object remove(int i) // Remove the item at location i in this Vector.
    int size() // Return the number of items in this Vector.
}
```

---

## Inductive proof outline

**Base Case**: Prove $S(\boxed{\phantom{0}})$.

Let $k \geq \boxed{\phantom{0}}$ be an arbitrary integer.

**Induction Hypothesis**: Assume $\boxed{\phantom{0000000}}$ is true.

**Induction Step**: Prove $S(\boxed{\phantom{0}})$ is true.

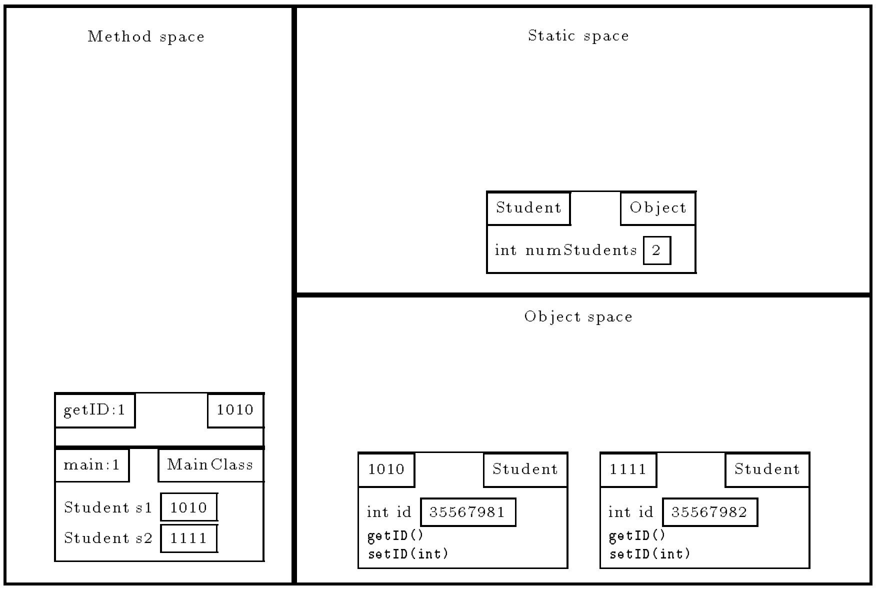
**Conclusion**: $S(n)$ is true for all $n \geq \boxed{\phantom{0}}$.

## 10 step for developing a recursive method

1. How can you reduce the problem to one or more simpler sub-problems of the same form?
2. What information is needed as input and output for the recursion?
3. Write the method header.
4. Write a method specification that explains exactly what it will do, in terms of the parameters. Include any necessary preconditions.
5. When is the answer so simple that we know it without recursing? What is the answer in the base case(s) (also called "degenerate")?
6. Write code for the base case(s).
7. Describe the answer in the other case(s) in terms of the answer on smaller inputs.
8. Simplify if possible.
9. Write code for the recursive case(s).
10. Put it all together.

---

## Memory model example




Total Marks = 106