PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

APRIL/MAY EXAMINATIONS 2003

CSC 148H1 S and CSC A58H1 S
St. George Campus

Duration — 3 hours

Aids allowed: none

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.)

This examination consists of 8 questions on 16 pages (including this one). *When you receive the signal to start, please make sure that your copy of the examination is complete.*

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.

You need not throw or catch exceptions, unless explicitely asked to.

Helper methods are always allowed.

# 1:	______/10
# 2:	______/10
# 3:	______/10
# 4:	______/10
# 5:	______/10
# 6:	______/10
# 7:	______/10
# 8:	______/10
TOTAL:	______/80

Good Luck!

Question 1. [10 MARKS]

Examine the following header for the merge operation of mergesort:

```
/**
 * Merge list[s...m] with list[m+1...e]
 * Requires: 0 <= s <= m < e < list.length &&  ______________________
 * Ensures: list[s...e] is sorted in ascending order
 */
private static void merge(Comparable[] list, int s, int m, int e) { ... }
```

Part (a) [2 MARKS]

The Requires clause is missing a piece. What else do you need to know about the array contents for merge to work correctly?

Part (b) [5 MARKS]

Below is the body of merge with some lines missing. On the next page is a list of Java statements. Fill in the blanks with the letters of the correct lines.

```
private static void merge(Comparable[] list, int s, int m, int e) {

  ___ // index of current candidate in list[s...m]
  ___ // index of current candidate in list[m+1...e]

  ___ // Temporary storage to accumulate the merged result
  int p=0;  // index to put next element into merged

  // merged[0...p] contains list[s...p1-1] merged with list[m+1...p2-1]
  ---
  ---
  ---
  ---
  } else {
  ---
  ---
  ---
  ---
  }
  ---
  System.arraycopy(list, p1, list, s+p, m+1-p1);
  ---
  ---
}
```

Lines:

A) `++p;`

B) `++p1;`

C) `++p2;`

D) `if (p1 != m + 1) {`

E) `if (list[p1].compareTo(list[p2]) < 0) {`

F) `int p1= s;`

G) `int p2= m + 1;`

H) `merged[p]= list[p1];`

I) `merged[p]= list[p2];`

J) `while (p1 != m + 1 && p2 != e + 1) {`

K) `Comparable[] merged= new Comparable[e - s + 1]`

L) `System.arraycopy(merged, 0, list, s, p);`

M) `}`

N) `}`

Part (c) [3 MARKS]

Consider a call to `mergesort` with array `b`, below. What will the contents be just before the very last call to `merge` during the execution of `mergesort`? Enter your answer in the blank array provided below.

```
   +---+---+---+---+---+---+---+---+---+---+
b: | 8 | 0 | 5 | 4 | 1 | 2 | 7 | 6 | 9 | 3 |
   +---+---+---+---+---+---+---+---+---+---+
```

Answer:

```
   +---+---+---+---+---+---+---+---+---+---+
   |   |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+---+
```

Question 2. [10 MARKS]

Consider the following BST:

```
        M
      /   \
     E     S
      \   / \
       G O   W
```

Definition: the *height* of a tree is the number of nodes on the longest path from the root to a leaf. In this example, the height is 3.

The example can be represented using an array to hold the keys, as follows:

```
index:     0     1     2     3     4     5     6
        +-----+-----+-----+-----+-----+-----+-----+
keys:   | "M" | "E" | "S" |null | "G" | "O" | "W" |
        +-----+-----+-----+-----+-----+-----+-----+
```

Class `ArrayBST` below stores a tree in an array. Write the representation invariant, the body of the constructor, and the body of `linkedRepresentation`. For `linkedRepresentation` use the following class:

```
  public BSTNode {
    public BSTNode left;
    public BSTNode right;
    public Comparable key;
  }
```

You might find this method from class `Math` helpful in your constructor:

```
  public static double pow(double a, double b) // Return a raised to the power b.
```

```
public class ArrayBST {

  /** REPRESENTATION INVARIANT:









   */
  private Comparable[] keys;

  /** An empty ArrayBST that will always represent a tree of height <= maxHeight.
    * Requires: maxHeight >= 0. */
  public ArrayBST(int maxHeight) {






  }
  /** Return the root of a BST made of BSTNodes that represents the
    * same tree as this tree does. */
  public BSTNode linkedRepresentation() {









  }

  // Insert, remove, and contains not shown.

  // You may write any helper method(s) here:










}
```

Question 3. [10 MARKS]

Consider a Java class which is used to hold the names of individuals ranked according to some ranking scheme that we don't care about.

```
/**
 * A list of names of individuals ordered by a ranking determined by the
 * World Federation of Combined Soccer and Golf. Each name appears only
 * once in the list.
 */
public class OrderedList {

  private int size;
  private String[] list;

  public OrderedList(int capacity) {
    list= new String[capacity];
    size= 0;
  }

  /**
   * Returns the number of elements in this OrderedList.
   * @return number of elements
   */
  public int numElements() {
    return size;
  }

// Some other methods go here, e.g. insert a name, remove a name, etc.
// which don't concern us now.
```

The class also provides a method `findInRange`, which appears on the next page. The method comment for `findInRange` is incomplete, possibly wrong, and does not follow the principles of Design by Contract.

Rewrite the comment so that it describes exactly what the method does; it should follow the principles of Design by Contract and include appropriate pre- and post-conditions.

```
/**
 * Loops through the storage array, and if name is in the range specified
 * by start and end, return the index of the element where name is
 * stored. Returns false otherwise.
 * @param start beginning of range
 * @param end end of range
 */
public int findInRange(String name, int start, int end)  {
  int pos= -1;
  int i= start + 1;
  while (i <= end && pos == -1) {
    if (list[i].compareTo(name) == 0) {
      pos= i;
    }

    i++;
  }

  return pos;
}
```

Rewritten comment:

Question 4. [10 MARKS]

You are working on a program that helps people learn the base 8 number system, which uses only the digits 0 ... 7.

Your current task for the program is to create appropriate methods to get user input, and to reject input containing digits that are not used in the base 8 system.

For this program, `BufferedReader`'s `readLine()` method is guaranteed to return a `String` containing only digits 0 .. 9 (your users have access to a numeric keypad only).

It may help you to know that class `Exception` has a constructor with a `String` argument, and a method `getMessage()` that returns that `String`.

Part (a) [3 MARKS]

Write a `private static void` method `checkDigits(String)` that throws a `WrongDigitException` if and only if a wrong digit is encountered in the input `String`.

Part (b) [3 MARKS]

Write class `WrongDigitException`, a subclass of `Exception`:

Part (c) [4 MARKS]

Write a `public static` method `getInput(BufferedReader)` that reads a line from the `BufferedReader` and calls `checkDigits` on that input. `getInput()` returns the input `String` if it is valid and otherwise returns "`X is not a proper digit in base 8`", where `X` is the first wrong digit found.

Question 5. [10 MARKS]

Part (a) [4 MARKS]

Consider the following set of statements:

1) If A is true, then B and C are true.
2) If B is true, then A is false and D is true.
3) Only one of C and D is true.

Circle all the statements below that are implied by the statements shown above.

- $A \Longrightarrow$ **not** D
- $B \Longrightarrow$ **not** A
- $C \Longrightarrow$ **not** B
- $D \Longrightarrow$ **not** C

Part (b) [6 MARKS]

Consider the following theorem:

2^n is always less than $n!$ for integer values of n greater than or equal to ____________.

1) Fill in the blank with the smallest correct value.
2) Use induction to prove this theorem.

Question 6. [10 MARKS]

Part (a) [6 MARKS]

What is the running time for the following operations, in `O(..)` notation? Give as small and simple a bound as possible.

1) Finding out whether a linked list with n nodes contains any duplicate elements: _______
2) Printing the nth element of a sorted array: _______
3) Calculating the height of a binary tree with n nodes: _______
4) Printing the contents of a binary tree with n nodes in ascending order, without using any kind of list or another tree: _______
5) Printing the contents of a binary search tree with n nodes in ascending order: _______
6) Finding the smallest element in a full binary search tree with n nodes: _______

Part (b) [4 MARKS]

A method used to find all prime numbers that are less than n runs as follows:

1) Make a table of integers from 2 to n.
2) For every value of i from 2 to $\sqrt{n}$, cross out the entries $2 * i$, $3 * i$, $4 * i$ up to n.
3) Once $i > \sqrt{n}$, print out all entries that have not been crossed out.

Circle the tightest bound for the questions below.

A) What is the running time for Step 1?

```
O(1)            O(n)
O(n^2)          O(log n)
O(sqrt(n))      O(n*sqrt(n))
```

B) What is the running time for Step 2?

```
O(1)            O(n)
O(n^2)          O(log n)
O(sqrt(n))      O(n*sqrt(n))
```

C) What is the running time for Step 3?

```
O(1)            O(n)
O(n^2)          O(log n)
O(sqrt(n))      O(n*sqrt(n))
```

D) What is the total running time for this method? _______

Question 7. [10 MARKS]

Consider this code.

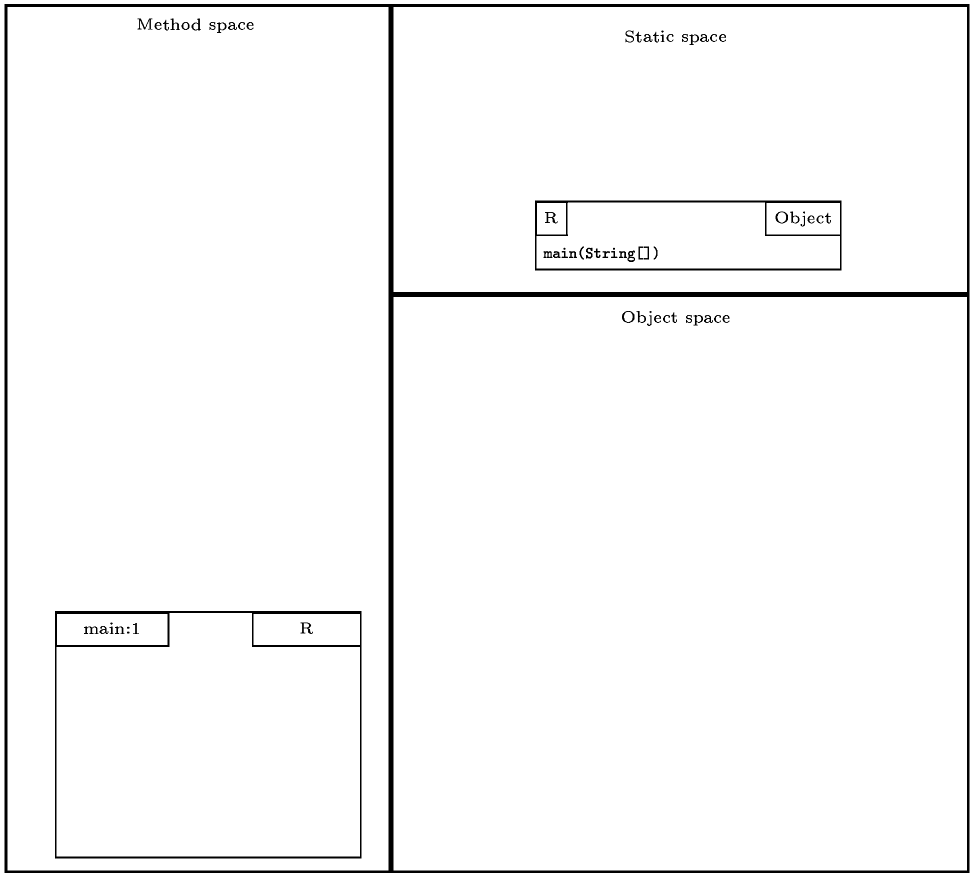
```
public class R {
  private R next;

  public R(R n) {
    next= n;                  // 1
  }

  public void m(int i) {
    if (i != 0) {             // 1
      if (i % 2 == 0) {       // 2
        m(i - 1);             // 3
      } else {                // 4
        next.m(i - 1);        // 5
      }
    }
  }

  public static void main(String[] args) {
    R last= new R(null);      // 1
    R middle= new R(last);    // 2
    R first= new R(middle);   // 3
    first.m(2);               // 4
  }
}
```

Below is a memory model trace paused just after `main` has been called. Show the state of memory when line 1 of method `m` has been reached and parameter `i` is `0`.



CONT'D...

Question 8. [10 MARKS]

Consider the following two classes:

```
public Node {
  public Node next;
  public Object value;
}
```

```
public DoubleNode {
  public Node next;
  public Node previous;
  public Object value;
}
```

Write the bodies of the `reverse` methods below. They modify the original lists, they don't make copies.

You may only declare variables of type `Node` or `DoubleNode`, and you must never use `new`.

Part (a) [5 MARKS]

```
/** Reverse the order of the nodes in the linked list <code>list</code> and return the list.
  * @param  list  first node of a linked list, null if the list is empty.
  * @return the first node of the modified list. */
public static Node reverse(Node list) {
```

Part (b) [5 MARKS]

```
/** Reverse the order of the nodes in the linked list <code>list</code> and return the list.
  * @param  list  first node of a linked list, null if the list is empty.
  * @return the first node of the modified list. */
public static DoubleNode reverse(DoubleNode list) {
```

This page is left mostly blank for rough work.

This page is left mostly blank for rough work.

Reference sheet

```
interface Iterator:                       interface Comparable:
  next() // the next item.                  // < 0 if this < o, = 0 if this is o, > 0 if this > o.
  hasNext() // = "there are more items".    int compareTo(Object o)

class Integer:                            class Vector:
  Integer(int v) // wrap v.                 void add(Object o) // Add o to the end.
  Integer(String s) // wrap s.              void add(int i, Object o) // this[i] = o.
  int intValue() // = the int value.        Object get(int i) // = this[i].
                                            Object remove(int i) // Remove & return this[i].
                                            int size() // Return the number of items.
class String:
  char charAt(int i) // = the char at index i.
  compareTo(Object o) // < 0 if this < o, = 0 if this == o, > 0 otherwise.
  compareToIgnoreCase(String s) // Same as compareTo, but ignoring case.
  endsWith(String s) // = "this String ends with s"
  startsWith(String s) // = "this String begins with s"
  equals(String s) // = "this String contains the same chars as s"
  indexOf(String s) // = the index of s in this String, or -1 if s is not a substring.
  indexOf(char c) // = the index of c in this String, or -1 if c does not occur.
  substring(int b) // = s[b .. ]
  substring(int b, int e) // = s[b .. e)
  toLowerCase() // = a lowercase version of this String
  toUpperCase() // = an uppercase version of this String
  trim() // = this String, with whitespace removed from the ends.

interface Stack:                               interface Queue:
  void push(Object o) // Add o to the top.       void enqueue(Object o) // Add o to the end.
  Object pop() // Remove & return the top.       Object dequeue() // Remove & return the front.
  Object top() // Return the top.                Object front() // Return the front.
  boolean isEmpty() // = "there are no items".   boolean isEmpty() // = "there are no items".

class Object:
  String toString() // return a String representation.
  boolean equals(Object o) // = "this is o".

class System:
  arraycopy(Object a1, int i, Object a2, int j, int n) // Copy a1[i..i+n-1] to a2[j..j+n-1]
```

Inductive proof outline

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

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

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

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

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

Partial list of Exception classes

ArrayIndexOutOfBoundsException
ClassCastException
Exception
IOException
IndexOutOfBoundsException
MalformedURLException
NoSuchElementException
NullPointerException
StringIndexOutOfBoundsException

Total Marks = 80