PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

DECEMBER EXAMINATIONS 2005

CSC 148/150H1 F
St. George Campus

Duration — 3 hours

Aids allowed: none

Student Number:	
Family Name:	
Given Name:	

*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 7 questions on 10 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 and throwing exceptions are not required except where indicated.

Write your student number at the bottom of pages 2-10 of this test.

# 1:	______/14
# 2:	______/14
# 3:	______/12
# 4:	______/14
# 5:	______/10
# 6:	______/14
# 7:	______/12
TOTAL:	______/90

Good Luck!

Question 1. [14 MARKS]

For each of the following operations briefly describe the most efficient (in the big-O sense) algorithm. Give the worst-case runtime efficiency in big-O (using the smallest, simplest expression).

Part (a) [2 MARKS] Given a balanced binary search tree of n nodes containing integers, return true if it contains the number 32.

Algorithm:

Runtime efficiency:

Part (b) [2 MARKS] Given a linked list of n nodes containing integers, move the node with the largest value to the end of the list.

Algorithm:

Runtime efficiency:

Part (c) [2 MARKS] Given two arrays of n integers, check if every integer in the first array also appears in the second array.

Algorithm:

Runtime efficiency:

Part (d) [2 MARKS] Given a balanced binary search tree of n nodes containing integers, return the biggest difference between any node and its successor.

Algorithm:

Runtime efficiency:

Part (e) [2 MARKS] Given a sorted array of n integers, return the smallest integer in the array which is larger than 42 (if any).

Algorithm:

Runtime efficiency:

Part (f) [2 MARKS] Given an array of n integers, determine if there are two elements in this array which sum to 100.

Algorithm:

Runtime efficiency:

Part (g) [2 MARKS] Given an array of n integers, check if there is a subset of this array whose elements sum to 0.

Algorithm:

Runtime efficiency:

Question 2. [14 marks]

Complete the method `has100Run` below. The only objects you may create are instances of the class `CircularQueue` from the lectures. This class has the constructor `CircularQueue(int capacity)` and implements the following interface:

```
public interface Queue {
    void enqueue(Object o);
    Object head();
    Object dequeue();
    int size();
    int capacity();
}
```

Recall the Integer class has a method `int intValue()` which returns the value. Also recall the `Iterator` interface has the methods:

```
public interface Iterator {
    Object next();
    boolean hasNext();
}
```

```
/** Returns whether i has a contiguous (i.e. with no gaps) subsequence of elements
 *  whose values add to exactly 100.
 *  Precondition: i produces only Integer objects, with strictly positive values. */
public static boolean has100Run(Iterator i)
```

Use the back of this page if you need more space.

Question 3. [12 MARKS]

Consider the following recursive method:

```
public static int m(int a, int b, int c) {
  if (a > b) {
    return m(b, a, c);
  } else if (b > c) {
    return m(a, c, b);
  } else {
    return c;
  }
}
```

Part (a) [4 MARKS]

Give a good Javadoc method comment for `m`.

Part (b) [2 MARKS]

Give an example of concrete values `a`, `b` and `c` for which `m(a, b, c)` results in as deep a recursion as possible.

Answer:

Part (c) [6 MARKS]

Show the runtime stack after calling `m` with your values of `a`, `b` and `c`, just before `return c` executes for the first time.

Question 4. [14 MARKS]

Consider the following class for nodes in a Linked List:

```
class ListNode {
  public Comparable data;
  public ListNode link;
}
```

Complete the following method. You **must not** use recursion. You can use non-recursive helper methods. Do not use any other datatypes such as arrays.

Recall that the `Comparable` interface consists of the method `int compareTo(Object o)`.

```
/** Merge two linked lists that are sorted in non-decreasing order into one
 *  sorted list containing all of the elements in the two given lists,
 *  including any duplicate elements.  Returns the head of the merged list.
 *  This method reuses the nodes in the original two lists.
 *  @throws java.lang.IllegalArgumentException (a RuntimeException) if
 *          either argument list h1 or h2 is not sorted.  */
public static ListNode merge(ListNode h1, ListNode h2) {
```

Question 5. [10 MARKS]

Consider the following class for nodes in a Binary Search Tree:

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

Complete the following method. **Do not use recursion**. You can use helper methods, but they cannot be recursive. Do not use any other datatypes such as arrays or lists.

Recall that the `Comparable` interface consists of the method `int compareTo(Object o)`.

```
/** Return the second smallest key in the binary search tree rooted at t.
    Precondition: t contains at least two keys, and all keys are distinct. */
public static Comparable secondSmallest(BSTNode t) {
```

Question 6. [14 marks]

Consider the following class for nodes in a Binary Search Tree:

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

Complete the following method. You can use a helper methods. You **must** use recursion. Do not use any other datatypes such as arrays or lists.

Recall that the `Comparable` interface consists of the method `int compareTo(Object o)`.

```
/** Return true exactly when the tree rooted at t is a binary search tree.
 *  Precondition:  All the keys in the tree are distinct.  */
public static boolean isBST(BSTNode t) {
```

Question 7. [12 MARKS]

Part (a) [6 MARKS]

Complete the body of `mergeSort` below with a **non-recursive** implementation. You must make an appropriate use of the specified `for` loop. You may assume `merge` specified below is available and in the same class.

```
/** Merge a[s .. m] with a[m+1 ..e].
    Precondition: the two sublists are sorted in non-decreasing order.
    Postcondition: a[s .. e] sorted in non-decreasing order. */
private static void merge(int[] a, int s, int m, int e);

/** Merge sort a in non-decreasing order.
    Precondition: a.length is a power of 2. */
public static void mergeSort(int[] a) {



  for (int step = 2; step < a.length; step *= 2) {
```

Question continues on next page.

Part (b) [3 MARKS]

Carefully explain the order of the running time of your code from part (a) (assume an efficient implementation of `merge`). Note: even if you don't complete part (a) you can reason about the rest of the code.

Part (c) [3 MARKS]

What is the maximum number of methods that can appear on the runtime stack when a traditional **recursive** implementation of `mergeSort` is called from a main method? Explain briefly. You can ignore any methods called before the main method, for example, those which are part of an IDE such as Dr.Java. You can assume the length of the original array is a power of 2.

For scratch work.

Total Marks = 90