UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

E&CE 250 – ALGORITHMS AND DATA STRUCTURES

Midterm Examination — Instructor: R.E.Seviora — 1.5 hrs, Oct 27, 2000

SOLUTIONS (INCOMPLETE, UNCHECKED)				Student id:
1.	2.	3.	4.	Total:

Do all problems. The number in brackets denotes the relative weight of the problem (out of 100). If information appears to be missing from a problem, make a reasonable assumption, state it and proceed. If the space to answer a question is not sufficient, use the last (overflow) page.
Closed book. Only numeric display calculators are allowed.

PROBLEM 1 [20]

A. Running Time Analysis

Determine the running times predicted by the simplified model of the computer for the following program fragment

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

statement	time
1a	2
1b	$3(n+1)$
1c	$4n$
2a	$2n$
2b	$3(I+n)$
2c	$4I$
3	$4I$
TOTAL	$11/2n^2 + \frac{35}{2}n + 5$

The number of iterations of the inner loop is I where

$$\begin{aligned} I &= \sum_{i=0}^{n-1}(n-i) \\ &= \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i \\ &= n^2 - \frac{(n-1)(n)}{2} \\ &= \frac{n(n+1)}{2}. \end{aligned}$$

B. Solving Recurrences

Solve the following recurrence:

$$T(n) = \langle \begin{array}{ll} O(1) & n = 1 \\ aT(\lfloor n/a \rfloor) + O(n) & n > 1, a >= 2 \end{array}$$

You may assume that n is a power of a. Show all your work.

drop the $O(\cdot)$s and assume that $n = a^m$:

$$\begin{aligned} T(a^m) &= aT(a^{m-1}) + a^m, \quad n > 0 \\ &= a(aT(a^{m-2}) + a^{m-1}) + a^m \\ &= a(a(aT(a^{m-3}) + a^{m-2}) + a^{m-1}) + a^m \\ &\vdots \\ &= a^k T(a^{m-k}) + ka^m \\ &\vdots \\ &= a^m T(a^0) + ma^m, \quad m - k = 0 \\ &= n + n\log_a n \\ &\leq 2n\log_a n \quad n \geq a. \end{aligned}$$

Therefore, $T(n) = O(n \log n)$.

PROBLEM 2 [25]

A. Big Oh Analysis

Do a big Oh analysis of the following algorithm:

```
01 public class Example
02 {
03     public static int geoSeriesSum (int x, int n)
04     {
05          int sum = 0;
06          for (int i = 0; i <= n; ++i)
07          {
08             int prod = 1;
09             for (int j = 0; j < i; ++j)
10                prod *= x;
11             sum += prod;
12          }
13          return sum;
14     }
15 }
```

Stmt	Time
5	O(1)
6a	O(1)
6b	O(n)
6c	O(n)
8	O(n)
9a	O(n)
9b	$O(n^2)$
9c	$O(n^2)$
10	$O(n^2)$
11	O(n)
13	O(n)

TOTAL = O(n^2)

B. Miscellanea

(a) The course introduced three different types for representation of integers: `int`, `Integer` and `Int`. Summarize briefly the differences between these three types.

```
int is a Java primitive type; the value of int is stored directly

Integer is the Java wrapper for int; it wraps int inside an object

Int is an E&CE250 wrapper for int; it was introduced because of the need to
view Int as an instance of the type Comparable (defined in E&CE 250)which In-
teger does not subtype
```

(b) What is the difference between `extends` and `implements` in Java?

```
extend:
- relationship between classes;
- a class C which extends superclass S inherits from C all the fields and
methods visible to it;
- also a relationship between interfaces
- only one class (or interface) can be extended

implements:
- relationship between a Java class and a Java interface
- a class implementing an interface must implement all the methods in the in-
terface (unless they are declared abstract in the class)
- a class may implement several interfaces
```

C. Visitors

Consider a container class whose instances will contain objects of the type `Float`. The value encapsulated in each object can be obtained using the method `float floatValue()` of the class `Float`.

Implement a sum-of-squares `Visitor` whose `visit` method computes the sum of squares of the values in the objects in the container, i.e.

$$S = \sum_{objects} V_k^2$$

where V_k is the value encapsulated in the k-th object. The computed value of the above sum should be accessible through the method `float getSumOfSquares()`of the `Visitor`. Note that the `Visitor` ‘s method `isDone()` is inherited from the class `AbstractVisitor`. It will always return `false`, which is appropriate for the case considered.

```
public class SumOfSquaresVisitor extends AbstractVisitor {
  //fields

   protected float sumSq = 0.0

  //methods
  public void visit (Object obj) {
    float val = ((Float) obj).floatValue();
    sumSq += val*val;
  }



  public float getSumOfSquares () {
    return sumSq;
  }
}
```

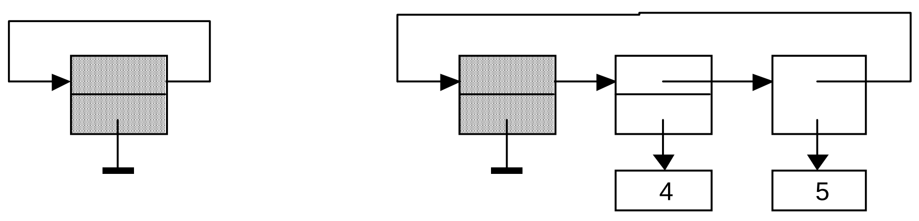
PROBLEM 3 [25]

A. Singly-linked List

(a) The singly linked list class considered in the lectures had the structure shown below. Fill in the dotted (. . .) entries.

```
   public class LinkedList {
      protected Element head;
      protected Element tail;
      public final class Element {
        Object datum;
        Element next;
        Element (Object datum, Element next) {
          this.datum = datum;
          this.next = next;
        }
        //etc
      }
   }
```

(b) A more uniform implementation of a singly liked list could have just a single `Element`, called the header node, plus other nodes if the list is nonempty. A diagram showing an empty list and a list with two `Integer` objects is below. The header node is shown in gray.



(i) Show the fields of the class `LinkedListHN` (linked list based on header node).

```
   protected Element headerNode;
```

(ii) Show the no-arg (i.e. no parameters) constructor for `LinkedListHN`.

```
   public LinkedListHN () {
     headerNode = new Element();
     headerNode.next = headerNode;
   }
```

(iii) Show the algorithm for the method `void prepend (Object obj)` of `LinkedListHN`.

```
   void prepend (Object obj) {
     Element tmp = new Element (obj, headerNode.next);
     headerNode.next = tmp;
   }
```

B. Project 2

The objective of Project 2 was to represent a polynomial in x

$$a_n x^n + a_{n-1} x^{n-1} + .. + a_1 x + a_0$$

where n ≥ 0 is the degree of the polynomial, using an <u>array</u>. The implementation was required to be reasonably lean in terms of computing time, and minimal: the array length at all times was to be *n+1*.

The PolynomialAsArray class declaration looked as follows:

```
class PolynomialAsArray {
   double[] a;               //array of coefficients
   PolynomialAsArray () {    //constructs the polynomial 0x⁰
   ...}
   //etc
}
```

(a) Draw a diagram that shows the objects stored in the memory of the computer after each of the following statements. <u>Identify</u> the objects that are the same (in the = = sense).

```
PolynomialAsArray p, q;
```

```
p = new PolynomialAsArray();
```

```
p.setCoefficient(3,13.0);
```

```
p.setCoefficient(0, 10.0);
p.setCoefficient(2, 12.0);
```

```
p.setCoefficient(3, 0.0);
```

```
q = p.plus(p);
```

(b) Derive an expression for the amount of memory required to represent a polynomial with degree m. Write your expression in terms of the following parameters: `sizeof(double), sizeof(int), ...`

PROBLEM 4 [30]

A. Enumeration

In the lectures, we showed the `getEnumeration()` method for `StackAsArray`. Show an implementation of `getEnumeration()` for `QueueAsArray`.

```
public class QueueAsArray extends AbstractContainer implements Queue {
    protected Object[] array;
    protected int head;
    protected int tail;
    // ...
    //Enumeration methods: nextElement() and hasMoreElements()
    public Enumeration getEnumeration () {
    {
      return new Enumeration () {
        protected int position = -1, enumCount = 0;

        public Object nextElement () {
          if (enumCount++ >= getCount()) throw new NoSuchElementException ();
          while (array[++position]==null) {};
          return array[position];
        }

        public boolean hasMoreElements() {
          return (enumCount < getCount ());
        }
```

B. Stacks

An application you are designing will use the abstract data type `Stack` (with operations `push, pop, getTop`). Furthermore, the application will need to frequently find the largest item on the stack. You are to develop a new class `StackMax`, with an additional method `findMax()` which returns a reference to the largest element on the stack.

In view of the anticipated frequency of invocations of this method, it is required that `findMax()` has the worst case running time of O(1). Note that `push, pop and getTop` should remain O(1).

Show the fields and the algorithms for `push`, `pop` and `findMax` for `StackMax`. You are not required to show the implementation for remaining methods. It is suggested that you base your implementation on one of the stack class implementations discussed in the course, specifically `StackAsLinkedList`, as opposed to implementing all methods from scratch.

If you are unable to devise an O(1) solution for `findMax()`, show a slower algorithm for part mark.

```
public class StackMax extends AbstractContainer implements Stack {

   //fields
    protected Stack s1 = new StackAsLinkedList ();
    protected Stack s2 = new StackAsLinkedList ();
   //methods

   public Object getTop ()
     { return s1.getTop (); }

   public Object findMax ()
     { return s2.getTop (); }

   public void push (Object arg)
   {
     Comparable object = (Comparable) arg;
     s1.push (object);
     if (object.isLT ((Comparable) s2.getTop ()))
       s2.push (object);
     else
       s2.push (s2.getTop ());
   }
   public Object pop ()
   {
     s2.pop ();
     return s1.pop ();
   }
```

C. Queues - Running Times

The `QueueAsLinkedList` class presented in the lectures was implemented using the `LinkedList` class of Chapter 4 of the text. The latter had both the head and the tail pointer in its implementation.

(a) In the table below, show the big Oh bounds for the running time for the methods given.
(b) Consider the case when the underlying `LinkedList` class was replaced by another implementation in which the `tail` field was dropped. In the table below, show the big Oh bounds for this case.

	Base: LinkedList with head,tail fields	Base: LinkedList with head field only
constructor	O(1)	O(1)
purge	O(1)	O(1)
enqueue	O(1)	O(n)
dequeue	O(1)	O(1)
getHead	O(1)	O(1)

OVERFLOW SHEET [Please identify the question(s) being answered.]