UNIVERSITY OF WATERLOO

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

E&CE 250 – ALGORITHMS AND DATA STRUCTURES

Midterm Examination Instructors: R.E.Seviora and L.Tahvildari 1.5 hrs, Feb. 14, 2001

Name: **SOLUTIONS [not checked]**						Student ID:		
1. A	1. B	2. A	2. B	3. A	3. B	4. A	4. B	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. No calculators allowed.

PROBLEM 1 [25]

A. Algorithm Analysis

Consider the Java program fragments given below. Assume that m, n, and k are non-negative **int**s and that the methods P, Q, R, and S have the following characteristics:

The worst running time for P(n ,m, k) is O(1) and it returns a value between 1 and (n + m + k).
The worst running time for Q(n, m, k) is O(nm + k).
The worst running time for R(n, m, k) is O(mk).
The worst running time for S(n, m, k) is O(n + k).

Determine a tight Big Oh expression for the worst-case running time of each of the following program fragments:

I. P(n, 10, 1)
O(1)

II.
III.

```
1 for (int i = 0; i < n; i++)
2    Q(n, m, k)
```

Statement	Time
1a	O(1)
1b	O(1)*n
1c	O(1)*n
2	O(nm+k)*n
Total	O($n^2m + kn$)

IV.

```
1 for (int i = 0; i < P(n, 10, 1); i++)
2    R(n, m, k)
```

Statement	Time
1a	O(1)
1b	O(1)*(n+10+k)
1c	O(1)*(n+10+k)
2	O(mk)*(n+10+k)
Total	O(nmk+10mk+m k^2)

IV.

```
1 for (int i = 0; i < n; ++i)
2  for (int j = 1; j < n; ++j)
3        S(n, m, k)
```

Statement	Time
1a	O(1)
1b	O(1)*n
1c	O(1)*n
2a	O(1)*n
2b	O(1)*O(n^2)
2c	O(1)*O(n^2)
3	O(n+k)* O(n^2)
Total	O($n^3 + n^2k$)

B. Asymptotic Bounds

Consider the functions e(n)...q(n), and complete the table showing their asymptotic relationship.

e(n) = n

f(n) = $3n^2 + 2n - 4$

g(n) = n^2

$$h(n) = \begin{cases} n^2 & n \ is \ even \\ 0 & n \ is \ odd \end{cases}$$

$$k(n) = \begin{cases} n & n \ is \ even \\ n^2 & n \ is \ odd \end{cases}$$

m(n) = log n

p(n) = log (n+1)

q(n) = ln n

True/False	Statement
True	f(n) = O(g(n))
False	h(n) = O(n)
True	h(n) = O(n^2)
False	h(n) = O(n log n)
False	k(n) = Ω(g(n))
True	k(n) = Ω(e(n))
True	f(n) + h(n) = O(n^2)
True	m(n) = O(p(n))
True	q(n) = O(m(n))
True	m(n) = O(q(n))

PROBLEM 2 [25]

A. Running Times

Determine the running time predicted by the detailed and the simplified computer model presented in the lectures for the following program fragment (where $n \geq 1$):

```
1    for (int i= 0; i < n; ++i)
2      for (int j = 0; j < i * i; ++j)
3        ++k;
```

Statement	*Running Time: Detailed Model*	*Running Time: Simplified Model*
1a	$t_{fetch} + t_{store}$	2
1b	$(2t_{fetch} + t_{<})*(n+1)$	3(n+1)
1c	$(2t_{ftech} + t_{+} + t_{store})*n$	4n
2a	$(t_{fetch} + t_{store})*n$	2n
2b	$(3t_{fetch} + t_{*} + t_{<})*(I+n)$	5(I+n)
2c	$(2t_{fetch} + t_{+} + t_{store})*I$	4I
3	$(2t_{fetch} + t_{+} + t_{store})*I$	4I
TOTAL	$(\frac{7}{3}t_{fetch} + \frac{2}{3}t_{+} + \frac{1}{3}t_{*} + \frac{1}{3}t_{<} + \frac{2}{3}t_{store})*n^3 - (\frac{7}{2}t_{fetch} + t_{+} + \frac{1}{2}t_{*} + \frac{1}{2}t_{<} + t_{store})$ $*n^2 + (\frac{55}{6}t_{fetch} + \frac{4}{3}t_{+} + \frac{7}{6}t_{*} + \frac{13}{6}t_{<} + \frac{7}{3}t_{store}) + (3t_{fetch} + t_{<} + t_{store})$	$\frac{13}{3}n^3 - \frac{13}{2}n^2 + \frac{97}{6}n + 5$

B. Solving Recurrences

Solve the following recurrence. You may assume that n is a power of a. Show all your work.

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

Show all your work. Drop O(.) and assume that $n = a^m$

$T(a^m)$ = a T(a^{m-1}) + 1

= a (a T(a^{m-2}) + 1) + 1

= a (a (a T (a^{m-3}) + 1) + 1) + 1

= a^k T(a^{m-k}) + $\sum_{i=0}^{k-1} a^i$

= a^m T(1) + $\sum_{i=0}^{m-1} a^i$ if m-k =0 ➔ m = k

= a^m + $\frac{a^m - 1}{a-1}$

= O(a^m) → T(n) = O(n)

PROBLEM 3 [25]

A. Space Requirement of Data Structures

NOTE: In the following, assume that an object reference occupies 4 bytes.

1. Consider a class `Array` class with two fields as follows:

```
public class Array   {
   protected Object[] data;
   protected int base;
   public Array (int n, int m)  {
       data = new Object[n];
       base = m;                }
// etc
}
```

Consider a particular instance of `Array, new Array (10, 5)`. How much space does this array instance occupy?

sizeof(Array) = sizeof(Object[N]) + sizeof(int)

= sizeof(int) + N*sizeof(Object ref) + sizeof(int)

= 2 * sizeof(int) + N * sizeof(Object ref)

= 2* 4 + 10 * 4

= 48 bytes

2. Consider the `LinkedList` class defined below:

```
public class LinkedList         {
   protected Element head;
   protected Element tail;
   public final class Element   {
     Object datum;
     Element next;
     //etc
     }
     //etc
   }
```

Consider a particular instance of `LinkedList` with 10 instances of `Integer`. How much space does the entire structure occupy (including `Element`'s and `Integer`'s)?

= sizeof(LinkedList) + n * sizeof(Element) + n * sizeof(Integer)

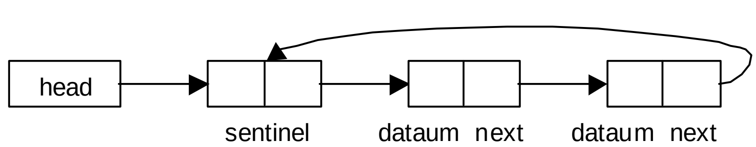
= 2 * sizeof(Object ref) + n * sizeof(Object ref) + n * sizeof(Object ref) + n * sizefof(Integer)

= 2 * 4 + 10 * 4 + 10 * 4 + 10 * 4

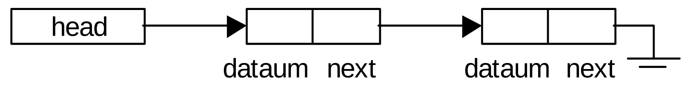
= 128 bytes

B. Singly-Linked List

A singly-linked list is simply a sequence of dynamically allocated objects, each of which refers to its successor in the list. Despite this simplicity, there are many implementation variants. One way is to add an extra element at the head of the list called the **sentinel** as shown below.



1. Explain the main advantage(s) and disadvantage(s) of this implementation in the comparison with the basic singly-linked list where we have only `head` field



Using the sentinel simplifies the programming of certain operations. However, the extra space is required because of sentinel. Also, sentinel need to be created when the list is initialized.

2. Write an implementation of the `prepend` method of the `LinkedList` class when the circular list with a sentinel as shown above is used. Assume that the constructor of `LinkedList` creates both the head and the sentinel and makes head refer to the sentinel. Fill in the dotted (…) entries

```
public class LinkedList
{
   protected Element head;
   public final class Element
   {
     Object datum;
     Element next;
     //etc
    }
    public void prepend (Object item)
    {
        head.next = new Element (item, head.next);
    }
}
```

PROBLEM 4 [25]

A. Visitor

Consider a container class whose instances will contain objects of the type `Int`. `Int` is a wrapper class which was introduced in the lecture. The int value encapsulated in each `Int` object can be obtained using the method `int intValue()` of the class `Int`, as illustrated below.

```
public class Int  extends AbstractObject
{
   protected int value;
   public Int (int value)
   {
      this.value = value;
   }
   public int intValue ()
   {
      return value;
    }
   // etc
}
```

Implement a `SummingVisitor` whose `visit` method computes the sum of all of the integer values stored in the objects in a container, i.e.

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

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 `int getSum()`of the `SummingVisitor`.

```
public class SummingVisitor extends AbstractVisitor {

  //fields
  protected int sum = 0;
```

```
  //methods
  public void visit (Object o)
  {
     sum += ((Int)o).intValue();
  }

  public boolean isDone()
  {
     return false;
  }

  public int getSum()
  {
    return sum;
  }
}
```

B. Big Oh Analysis [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 $a_n \neq 0$, n ≥ 0 is the degree of the polynomial, using Java array. The implementation was required to be reasonably lean in terms of computing time, and minimal in terns of space: the array length at all times was to be *n+1*.

The `PolynomialAsArray` class definition included:

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

One of the methods you were to provide was `double eval (double x)`. This method takes a single `double` argument, `x`, and computes the value of the polynomial for the given `x`. Devise an algorithm for `eval( )` whose tight big-oh expression for the running time is O(n) and write it down in Java.

```
public double eval (double x)
{
     double y = coefficient[degree];
     for (int i = getDegree() –1; i >= 0; i--)
          y = y * x + a[i];
      return y;
}
```

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