SYSC 2100, Fall 2004 Midterm
Sample Solutions

Question 1. Algorithm Complexity (10 marks)

For each of the following code segments, estimate worstTime(n) using Big-O notation or plain English. In each segment, S represents a sequence of statements in which there are no n-dependent loops.

1)
```
for (int i = 0; i * i < n; i++)
        S
```

The worstTime(n) is O($\sqrt{n}$). (3 marks)

2)
```
for (int i = 0; Math.sqrt (i) < n; i++)
        S
```

The worstTime(n) is quadratic in n: O(n^2). (3 marks)

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

The worstTime(n) is O(2^n). (4 marks)

Question 2. The Java Collections Framework (10 marks)

1. Identify each of the following as either an interface or a class: (2 marks)
   Collection: interface
   LinkedList: class
   Iterator: interface
   AbstractSet: class
   Map: interface

2. What is the difference between an interface and an abstract class? (2 marks)
   The methods in an interface must be abstract. An abstract class may have fully defined methods as well as abstract methods.

3. Of what value is an abstract class? That is, to what extent can an abstract class make a programmer more productive? (3 marks)
   An abstract class allows a developer to override, in a subclass of
   the abstract class, only those methods the developer chooses to.

4. What is a list? What is a set? What is a map? (3 marks)
   A ***list*** is an object in a class that implements the List interface. That is, a list can invoke index-related methods to insert an element, delete an element, and search for an element in the list.
   A ***set*** is an object in a class that implements the Set interface. That is, a set is a Collection object in which duplicate elements are not allowed.
   A ***map*** is an object in a class that implements the Map interface. That is, a map is a collection in which each element consists of two parts: a unique key and a value.

Question 3. ADT List (10 marks)

1. Suppose we added each of the following methods to the ArrayList class:
   - public boolean addFirst (Object element)
   - public boolean addLast (Object element)
   - public Object getFirst()
   - public Object getLast()
   - public Object removeFirst()
   - public Object removeLast()

   Estimate worstTime(*n*) for each method. (2 marks)

   addFirst: linear in *n*
   addLast: linear in *n* (because of the possibility of resizing)
   getFirst: constant
   getLast: constant
   removeFirst: linear in *n*
   removeLast: constant

2. In the Java Collections Framework, the LinkedList class is designed as a circular, doubly-linked list with a dummy entry (pointed to by the header field). What is the main advantage of this approach over a circular, doubly-linked list with head and tail fields? (2 marks)

   In the LinkedList class, the main advantage of having a dummy entry is that every entry, even the first or last entry, has a predecessor and a successor, so there is no need for a special case to insert or delete at the front or back of a LinkedList object.

3. Suppose we have the following:

```
LinkedList weights = new LinkedList();
ListIterator itr;
weights.add (new Double(5.3));
weights.add (new Double(2.8));
itr = weights.listIterator();
```

   Bad question: adding (as an iterator method) is not discussed in the course notes

   Hypothesize which of the following sequences of messages (i.e., method invocations) would now be legal: (6 marks)

```
a. itr.add (new Double(8.8)); itr.next(); itr.remove();
b. itr.add (new Double(8.8)); itr.remove(); itr.next();
c. itr.next(); itr.add (new Double(8.8)); itr.remove();
d. itr.next(); itr.remove(); itr.add (new Double(8.8));
e. itr.remove(); itr.add (new Double(8.8)); itr.next();
f. itr.remove(); itr.next(); itr.add (new Double(8.8));
```

   After sequences a and d, weights will contain 8.8 and 2.8, in that order. Sequences b and c throw IllegalStateException objects because remove requires that lastReturned not be equal to header, but itr.add (new Double(8.8)) sets lastReturned to header just before remove is called. Sequences e and f throw IllegalStateException objects because lastReturned is initialized to header just prior to the call to remove.

Question 4. Stacks and Queues (10 marks)

1. Suppose that elements "a", "b", "c", "d", "e" are pushed, in that order, onto an initially empty stack, which is then popped four times, and as each element is popped, it is enqueued into an initially empty queue. If one element is then dequeued from the queue, what is the *next* element to be dequeued? (2 marks)

   The next element to be dequeued is "d".

2. The array-based implementation of the Queue interface had *elementData*, *size*, *front* and *back* instance variables. Show that the *size* instance variable is redundant. Specifically, show that the *size*() method can be implemented using only the *back*, *front* and *elementData* variables. Assume that entries in *elementData* that do not contain an element are set to *null*. (3 marks)

```
public int size( )
{
    int result = back – front + 1;
    if (back == CAPACITY) {
            // check wether queue is completely full or completely empty
            if (elementData[front] == null)
                    return 0;
            else
                    return CAPACITY;
    if (result < 0) // back was actually smaller than front, account for wrapping around
            result = result + CAPACITY;
    return result;
} // method size
```

3. An expression in postfix notation can be evaluated at run time by means of a stack. For simplicity, assume that the postfix expression consists of integer values and binary operators only. For example, we might have the following postfix expression:
   8 5 4 + * 7 -

   The evaluation proceeds as follows: When a value is encountered, it is pushed onto the stack. When an operator is encountered, the first and second elements on the stack are retrieved and popped, the operator is applied (the second element is the left operand, the first element is the right operand) and the result is pushed onto the stack. When the postfix expression has been processed, the value of that expression is the top (and only) element on the stack. For example, for the above expression, the contents of the stack would be as follows:

			4				
		5	5	9		7	
	8	8	8	8	72	72	65
-----	-----	-----	-----	-----	-----	-----	-----

   Convert the following expression into postfix notation and then use a stack to evaluate the expression:
   5 + 2 * (30 - 10 / 5) (5 marks)

   The postfix notation is 5 2 30 10 5 / - * +

				10
			30	30
		2	2	2
	5	5	5	5
-----	-----	-----	-----	-----
5				
10	2			
30	30	28		

2	2	2	56	
5	5	5	5	61
-----	-----	-----	-----	-----