ECE 250
Data Structures and Algorithms
MIDTERM EXAMINATION

2005-10-27/4:30-6:00

The examination is out of 56 marks.

Instructions:

No aides.
Turn off all electronic media and store them under your desk.
If there is insufficient room, use the back of the previous page.
You may ask only one question during the examination:

"May I go to the washroom?"

If you think a question is ambiguous, write down your assumptions and continue.
Do not leave during the first 30 minutes of the examination.
Do not leave during the last 15 minutes of the examination.
Do not stand up until all exams have been picked up.

Attention:

The questions are in the order of the course material, not in order of difficulty.

I have read and understood all of these instructions:

Name: ____

Signature: ____

Runtime Analysis

1. [3] Indicate which of the functions

$3n + 5 \quad n^2 - 10n \quad \ln(n) \quad 1 \quad 10n^2 \quad n\log(n) \quad \log_2(n) \quad n$

are big-**O** of each other. You can simply circle and connect matching functions.

2. [2] Show, using limits, that $n \ln(n) = \mathbf{O}(n^{1.585})$.

3. [3] Assume that the run time of an algorithm is given by

$$\mathrm{T}(n) = \begin{cases} 1 & n = 1 \\ \mathrm{T}(n/2) + 2n & n > 1 \end{cases}$$

Assuming that $n = 2^k$, find the asymptotic (big-**O**) run time of $\mathrm{T}(n)$. You may require one of the following formulae:

$$\sum_{i=0}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

4. [3] A binary search is useful in searching an arbitrary array of ordered entries. Suppose that you have the additional information that the array is sorted and evenly distributed, that is, **`array[i] ~ ci`** for some integer constant **`c`**. If the size of the array is **`N`** and the range of the objects stored in the sorted array is between **`0`** and **`c*N`**, suggest a better strategy than bisection to check if an element **`x`** is in the array. You only need sketch the outline of your algorithm. Assume that you know the values of **`N`** and **`c`**. Will the asymptotic behaviour of your modified algorithm be any faster than that of a binary search?

5. [5] Determine the asymptotic (big-**O**) run times in terms of *n* of the following code fragments:

```
int sum = 0;
for ( int i = 0; i < n; ++i ) {
     for ( int j = 0; j < n; ++j ) {
          ++sum;
     }
}
```

Let `int f( int n, int m )` be a routine which runs in **O**(*n* + *m*) time.

```
int sum = 0;
for ( int i = 0; i < n; ++i ) {
     for ( int j = 0; j < n; ++j ) {
          sum += f( j, 10 );
     }
}
```

```
int sum = 0;
for ( int i = 1; i < n; i *= 2 ) {
     ++sum;
}
```

Formulae:

$$\sum_{i=0}^{n} 1 = n+1 \qquad \sum_{i=0}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

Linked Lists

6. [2] Figure 6 graphically shows the values of the elements in a non-empty linked list assigned to the variable `list`.

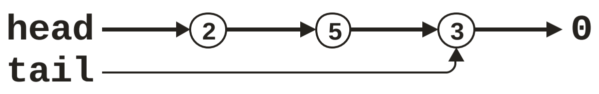

Figure 6. A non-empty singly-linked list.

Using the same style of diagram, indicate the state of the linked list after the following five operations (performed in the given order):

```
list.pop_front();
list.pop_front();
list.push_front( 4 );
list.push_back( 7 );
list.pop_front();
```

head ———
tail

7. [4] Given a doubly-linked list class **`DoubleList`** with three class members

```
DoubleNode * head;
DoubleNode * tail;
int count;
```

which are all initially assigned **`0`**, implement the method **`push_back`** which places an object in a new **`DoubleNode`** at the end of the doubly-linked list.

```
template <class Object>
void DoubleList<Object>::push_back( const Object & obj ) {











    ++count;
}
```

The class **`DoubleNode`** is provided for reference:

```
template <class Object>
class DoubleNode {
   Object element;
   DoubleNode * nextNode;
   DoubleNode * prevNode;

   DoubleNode(
      const Object & obj, DoubleNode * n = 0, DoubleNode * p = 0
   ):element( obj ), nextNode( n ), prevNode( p ) {
      // does nothing
   }

   friend class DoubleList<Object>;
};
```

8. [4] Given a singly-linked list class **`SingleList`** with the three class members

```
      SingleNode * head;
      SingleNode * tail;
      int          count;
```

which are all initially set to **`0`**, implement the method **`bool check()`** which returns **`true`** if the internal state of the linked list is consistent, that is:

1. the count is correct,
2. the tail pointer is pointing to the correct node, and
3. the last **`nextNode`** pointer is 0.

As soon as you can determine that one of these conditions is not met, you should return **`false`**. Your routine must run in **O(** **`count`** **)** time.

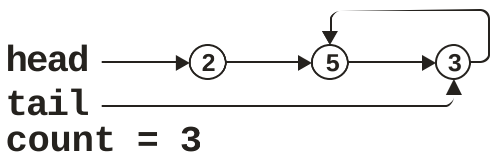

Figure 8. A linked-list with an inconsistent internal state.

```
template <class Object>
bool SingleList<Object>::check() {













}
```

The class **`SingleNode`** is provided for reference.

```
template <class Object>
class SingleNode {
      Object       element;
      SingleNode * nextNode;

      SingleNode(
            const Object & obj, SingleNode * n = 0
      ):element( obj ), nextNode( n ) {
            // does nothing
      }

      friend class SingleList<Object>;
};
```

Stacks and Queues

9. [8] With the addition of a new method **`void makeEmpty()`** which empties the stack (you don't have to implement it), the public interface of the drop-off stack-as-array class which you implemented in Project 2 is given here:

```
template<class Object>
class DropOffStackAsArray
{
        public:
                int size() const;
                bool empty() const;
                Object top() const;

                void push( const Object & );
                void pop();
                void makeEmpty();   // empty the stack
};
```

The **`UndoRedoStack`** class uses two of these stacks as its private members:

```
template <class Object>
class UndoRedoStack {
     private:
          DropOffStackAsArray<Object> undo_stack;
          DropOffStackAsArray<Object> redo_stack;
};
```

Assume that everything is appropriately initialized in the constructor.

You must implement the following five member functions:

`bool can_undo()`
Returns **`true`** if the undo stack is not empty and **`false`** otherwise.

`bool can_redo()`
Returns **`true`** if the redo stack is not empty and **`false`** otherwise.

`void event( const Object & obj )`
Places the object on the undo stack and empties the redo stack.

`Object undo()`
If the undo stack is empty, **`throw UnableToUndo()`**, otherwise pop the top object off of the undo stack, push it onto the redo stack and return the object.

`Object redo()`
If the redo stack is empty, **`throw UnableToRedo()`**, otherwise pop the top object off of the redo stack, push it onto the undo stack, and return the object.

Implement these member functions in the space provided on the next page.

DO NOT REIMPLEMENT STACKS. USE THE OBJECTS AND METHODS PROVIDED!

You may detach this page.

```
template <class Object>
class UndoRedoStack {
     private:
          DropOffStackAsArray<Object> undo_stack;
          DropOffStackAsArray<Object> redo_stack;
     public:
          // ...
};

template <class Object>
bool UndoRedoStack<Object>::can_undo() {




}

template <class Object>
bool UndoRedoStack<Object>::can_redo() {




}

template <class Object>
Object UndoRedoStack<Object>::undo() {








}

template <class Object>
Object UndoRedoStack<Object>::redo() {








}

template <class Object>
Object UndoRedoStack<Object>::event_performed(
                                 const Object & event ) {








}
```

Tree Traversals

10. [5] Perform pre- and post-order depth-first traversals and a breadth-first traversal of the tree in Figure 10. Print the nodes in the order in which they are visited in the tables provided.

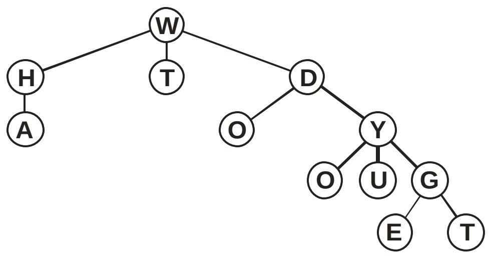

Figure 10. An arbitrary tree.

Table 10a. Pre-order depth-first traversal.

Table 10b. Post-order depth-first traversal.

Table 10c. Breadth-first traversal.

...when you multiply...

11. [3] By doing a post-order traversal of the expression tree shown in Figure 11, write down the post-fix expression corresponding to the expression tree. Evaluate the expression.

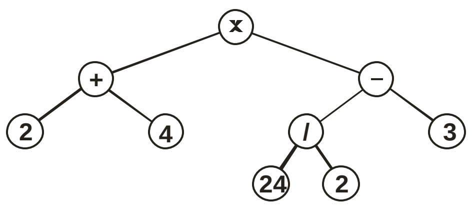

Figure 11. An expression tree.

Binary Search Trees

12. [3] Without balancing, simply insert the elements 1, 9, 5, 3, 8, 7, 2, 0, 4 into a binary search tree with a root node containing 6.

13. [2] Show the result of deleting the root node from the binary search tree shown in Figure 13.

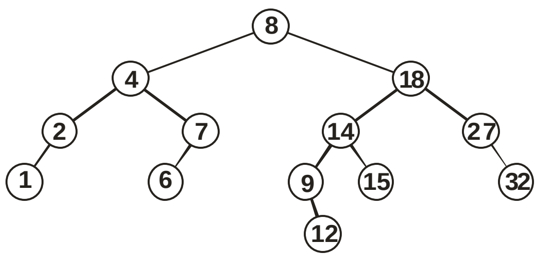

Figure 13. A binary search tree.

AVL Trees

14. [3] Given the AVL tree in Figure 14, indicate the locations where a new node may be inserted without unbalancing any of the nodes in the tree. (-0.5 for each incorrect placed or missing node.)

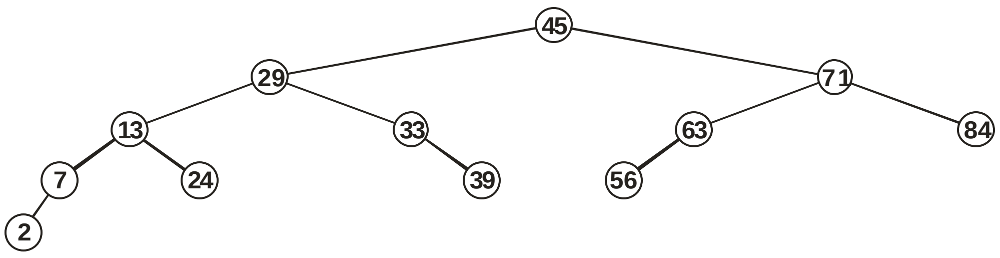

Figure 14. An AVL-balanced tree.

15. [6] For each of the following five AVL trees, perform the required insertion or deletion, making any rotations necessary to maintain the AVL balance. If the insertion or deletion does not result in an unbalanced tree, you may simply indicate this by adding or crossing out the appropriate node in the given figure.

Insert 25 into the AVL tree shown in Figure 15a.

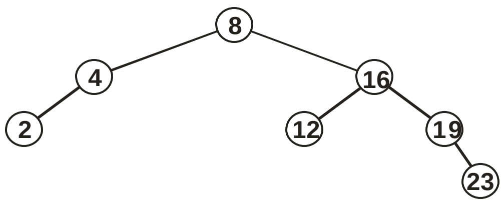

Figure 15a.

Insert 25 into the AVL tree shown in Figure 15b.

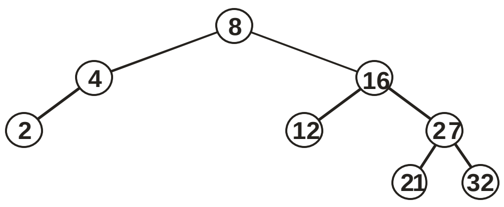

Figure 15b.

Remove 14 from Figure 15c.

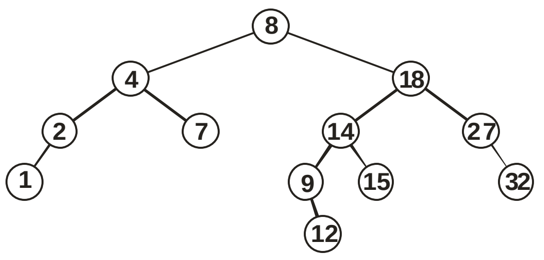

Figure 15c.

Remove 31 from Figure 15d.

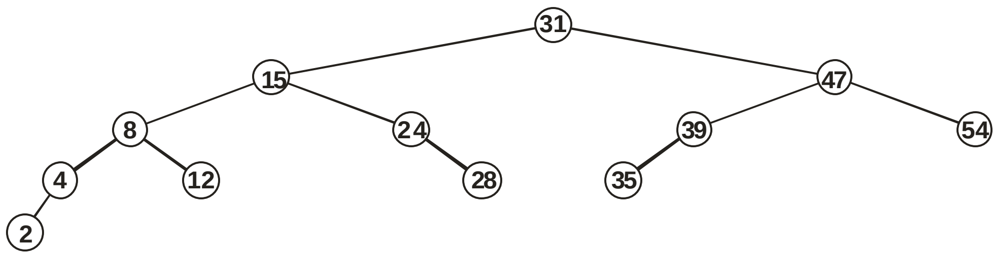

Figure 15d.

16. [2] Suppose after an insertion into an AVL tree, a node becomes unbalanced and a rotation is required to balance that node. Is it possible for the very next insertion to cause that same node to become unbalanced? Justify your answer.