# ECE 250 *Algorithms and Data Structures*
## Sections 001 and 002
## MIDTERM EXAMINATION

Douglas Wilhelm Harder dwharder@uwaterloo.ca EIT 4018 x37023

## 2014-3-1T09:00:00P1H20M
## Rooms: DC-1350 and DC-1351

Initial each instruction:

_____ There are 44 marks.
_____ 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?"
_____ Asking **any** other question **will** result in a deduction of 5 marks from the exam grade.
_____ If you think a question is ambiguous, write down your assumptions and continue.
_____ **Do not leave during first hour or after there are only 15 minutes left.**
_____ Write your UW User ID on the last page for a bonus mark.
_____ Do not stand up until all exams have been picked up.
_____ If a question asks for an answer, you do not have to show your work to get full marks; however, if your answer is wrong and no rough work is presented to show your steps, no part marks will be awarded.

**Attention:**

**The questions are in the order of the course material.**

**THIS BLOCK MUST BE COMPLETED USING ALL CAPITAL LETTERS (1 mark for getting it right)**

| Last name as appearing in Quest (e.g., HARDER) | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | A | R | D | E | R | | | | | | | | | | | | | | | |
| Legal Given names as appearing in Quest (e.g., DOUGLAS WILHELM) | | | | | | | | | | | | | | | | | | | | |
| D | O | U | G | L | A | S | | W | I | L | H | E | L | M | | | | | | |
| Common or nick name (entirely optional; e.g., DOUG) | | | | | | | | | | | | | | | | | | | | |
| D | O | U | G | | | | | | | | | | | | | | | | | |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| UW Student ID Number | **2** | **0** | 1 | 2 | 3 | 4 | 5 | 6 | |
| UW User ID | | | | | | | | | @uwaterloo.ca |

I have read the above instructions:

Signature: _____

**Asking any question other than that question noted above.**

**-5**

**A.1 [2]** Every hierarchical order is a partial order. Argue why every partial is not necessarily a hierarchical order **or** give an example of a partial order on a set that is not a hierarchical order.

**A.2 [2]** Using l'Hopital's rule and algebra, show that $n \ln(n) = o(e^n)$.

**A.3 [2]** Why does the asymptotic analysis we use to relate the run-times of algorithms and the memory usage of data structures define a weak order as opposed to a linear order? Justify your answer.

**A.4 [2]** What are the run times of the following code segments?

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

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

**B.1 [4]** Implement a `truncate( int n )` function for a doubly linked list that limits the size of the linked list to *n*. If the size is more than *n*, all nodes following the $n^{th}$ node are deleted from the linked list.

You may use the `list_head`, `list_tail`, `list_size` member variables and the corresponding functions `head()`, `tail()` and `size()` in the `Double_list` class, and the `next_node` and `previous_node` member variables and the corresponding member functions `next()` and `previous()` of the `Double_node` class.

```
template <typename Type>
void Single_list<Type>::truncate( int n ) {
    if ( n < 0 ) {
        throw illegal_argument();
    }
```

**B.2 [3]** Recall that a reverse Polish expression may be evaluated using a stack by pushing operands onto the stack and, for an operator, popping the last two operands off the stack, applying the operator, and pushing the result back onto the stack. At the end of the expression, the size of the stack should be one. Which of the following are valid reverse Polish expressions?

3 5 2 + + 4 * 3 5
5 4 + 2 * 5 6 + *
3 4 7 + + + 8 9 +

**B.3 [3]** Based on the previous question, can you find a relationship between the number of operands in a valid revere Polish expression and the number of operators? How does this relate to the relationship between the number of leaf nodes and internal nodes in a full binary tree (each node has either zero or two children)?

**C.1 [2]** Define the depth of a node and the height of a tree in terms of path lengths.

**C.2 [3]** What are the breadth-first and pre- and post-order depth-first traversals of the tree shown in Figure C.2?

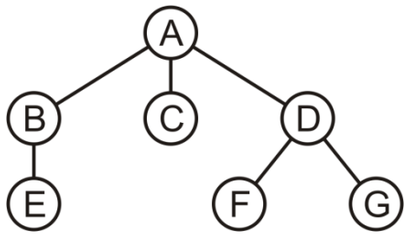


Figure C.2. A tree.

**C.3** [**2]** A developer has already written an efficient implementation of the member function

```
template <typename Type>
bool Simple_tree<Type>::is_ancestor_of( Simple_tree<Type> *treeptr ) const;
```

which returns `true` if `this` node in the tree is an ancestor of the node pointed to by the argument `treeptr` by following the list of `parent_node` pointers of the argument. You have to extend the interface by implementing a member function that returns `true` if this node in the tree is a descendant of the node pointed to by the argument `treeptr`. Implement this function efficiently, too. There are no restrictions on what you may do or what member functions you may call to implement this member function.

```
template <typename Type>
bool Simple_tree<Type>::is_descendant_of( Simple_tree<Type> *treeptr ) const {
```

**D.1 [2]** An *N*-ary tree is a tree that has *N* specific children (which may or may not exist). What are the maximum and minimum number of nodes in an quaternary tree of height 3?

**D.2 [3]** Give a proof by induction that

$$\sum_{k=0}^{h} k3^k = \frac{3+(2h-1)3^{h+1}}{4}.$$

**D.3 [5]** In class, we saw a node-based implementation of a binary tree. We also saw that you could store a complete binary tree as an array by using a breadth-first traversal. Given an example of how a binary tree (of size at least 6) can be stored in an array. Compare and contrast these two implementations with respect to memory usage and the run-time of the operations of finding the children and parent of a given node. If you did not have a `parent_node` member variable in the node-based class, how could you find the parent of a node?

**E.1 [2]** In the order given, insert the numbers 34, 15, 65, 59, 72, 42, 40, 80, 50, 5 into an initially empty binary search tree.

**E.2 [2]** Delete the root node of the tree you generated in Question F1. Explain what you are doing.

**E.3 [1]** Do we have to worry about maintaining AVL balancing in any functions that are defined `const` (as shown below)? Why or why not?

```
return_type AVL_node::function_name( … ) const;
```

**E.4 [4]** In class, we discussed two different cases for correcting imbalances caused by insertions into an AVL tree. Demonstrate these two cases by indicating a value that could be added into the AVL tree in Figure E.4 that results in each of these two cases. Next, for each insertion, perform the correct operation for that corresponding case to maintain the AVL balance.

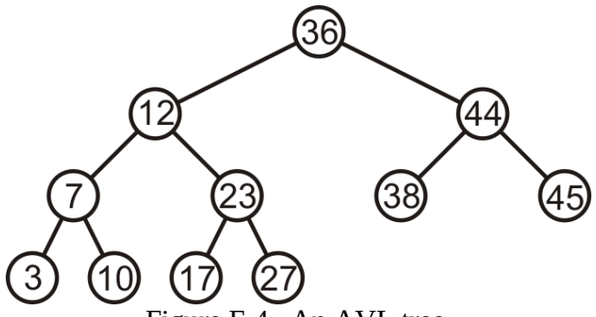


Figure E.4. An AVL tree.