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

Douglas Wilhelm Harder dwharder@uwaterloo.ca EIT 4018 x37023

2015-2-25T17:30:00P1H20M

Rooms:	MC-2034	20349155-20480690
	MC-2038	20480972-20508699
	MC-2054	20508814-20516557
	MC-4020	20516850-20525565
	MC-4021	20526073-205988681

1. There are 42 marks.
2. No aides.
3. Turn off all electronic media and store them under your desk.
4. If there is insufficient room, use the back of the previous page.
5. You may ask only one question during the examination: "May I go to the washroom?"
6. Asking **any** other question **will** result in a deduction of 5 marks from the exam grade.
7. If you think a question is ambiguous, write down your assumptions and continue.
8. **Do not leave during first hour or after there are only 15 minutes left.**
9. Write your UW User ID on the last page for a bonus mark.
10. Do not stand up until all exams have been picked up.
11. 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.
12. Remember that $\sum_{k=1}^{n} k^d \approx \frac{n^{d+1}}{d+1}$.

Attention:

The questions are in the order of the course material.

THIS BLOCK MUST BE COMPLETED USING ALL CAPITAL LETTERS

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

Student ID Number	2	0	1	2	3	4	5	6
User ID									@uwaterloo.ca

I have read the above instructions:

Signature:

____

Asking any question other than that question noted above.

-5

A.1 [1] The structure of classes related to each other through inheritance in C++ is represented by what type of ordering?

A.2 [2] Using l'Hopital's rule and algebra, show that $n \ln(n^3) = o(n^{1.5})$.

A.3 [3] In your own words, why is asymptotic analysis used in the analysis of algorithm and data structure run times and memory usage? That is, why not just execute the algorithms and calculate the run times and memory usage? One mark will be awarded for each supporting point.

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 < i; ++j ) {
            sum += i + j;
      }
}

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

B.1 [4] Implement a `rotate()` function for a double sentinel linked list that moves the node at the front of the linked list to the back in Θ(1) time. If the list size is less than or equal to one, do nothing.

You may use the `list_head`, `list_tail`, `list_size` member variables and the corresponding member functions `head()`, `tail()` and `size()` in the `Double_sentinel_list` class, and the `next_node` and `previous_node` member variables and the corresponding member functions `next()` and `previous()` of the `Double_node` class. You may also use the `std::swap` function which swaps its two arguments.

```
template <typename Type>
void Double_sentinel_list<Type>::rotate() {
    if ( size() <= 1 ) return;
```

B.2 [1] Calculate the following expression in reverse Polish notation:

```
3  5  2  +  +  3  *  3  5  +  *
```

B.3 [2] You are implementing a design using a queue where it is required that every push and pop operation must be Θ(1), but the size of the queue is not bounded. Would you use a dynamically resizing queue with a single array or a queue using a linked list of arrays? Why?

B.4 [2] Do the following descriptions refer to stacks, queues or neither:

first-in—last-out ____________________________

first-in—first-out ____________________________

last-in—first-out ____________________________

last-in—last-out ____________________________

B.5 [4] In class we saw that if you double the size of an array each time it is filled, then the amortized number of copies per insertion is Θ(1), as after inserting $n = 2^m$ objects, the number of copies is

$$\frac{\sum_{k=0}^{m-1} 2^k}{2^m} = \Theta(1) \; .$$

The number of calls to new[] is Θ(ln(n)).

If we increased the size of the array by a fixed amount, we saw that inserting n objects into the array results in an amortized Θ(n) copies per insertion with Θ(n) calls to `new[]`.

Suppose, instead, each time the array is full, we increase the size of the array to the next perfect square, so the sizes of the array are 1, 4, 9, 16, 25, 36, 49, etc. Given n insertions, what is the amortized number of copies per insertion? Asymptotically speaking, what is the number of calls to `new[]` given n insertions?

C.1 [3] What are the breadth-first and pre- and post-order depth-first traversals of the tree shown in Figure C.1?

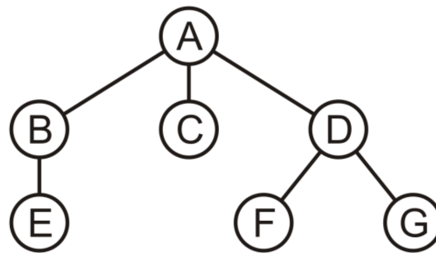

Figure C.1. A tree.

C.2 [**1**] When is the deepest common ancestor of two in a tree nodes (the ancestor shared by both nodes that is deepest within the tree) one of those two nodes?

C.3 [4] A more complex structure than a tree is a family tree, where each individual has two parents and zero or more children. Suppose that we have a four iterators that are returned by member functions as follows:

```
Family_tree::iterator parent_begin();
Family_tree::iterator parent_end();
Family_tree::iterator children_begin();
Family_tree::iterator children_end();
```

You can iterate through the parents of a node as follows:

```
for ( Family_tree::iterator itr = parent_begin(); itr != parent_end(); ++itr ) {
    Family_tree:iterator grand_itr = itr->parent_begin();
                                // allows you to access
                                // an iterator to the first of the parent's parents
}
```

with similar means of iterating through the children. Write a function that determines the number of cousins that an individual has. Be sure not to count you or your siblings. You may assume that there are four unique grandparents, that there was no inter-marrying of your parents, aunts and uncles and that all children are shared by both parents (i.e., you don't have to worry about half cousins or step cousins).

```
int Family_tree::cousin_count() const {
    int count = 0;



    return count;
}
```

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

$$\sum_{k=0}^{h}(2k+1) = (h+1)^2 .$$

E.1 [2] In the order given, insert the numbers 15, 65, 59, 68, 42, 40, 80, 50, 65 into an initially empty binary search tree.

E.2 [2] Explain the purpose of using an AVL trees versus using a more general binary search trees that performs simple insertions and erases as described in class.

E.3 [2] What is the maximum and minimum number of nodes that can be stored in an AVL tree of height 4?

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.

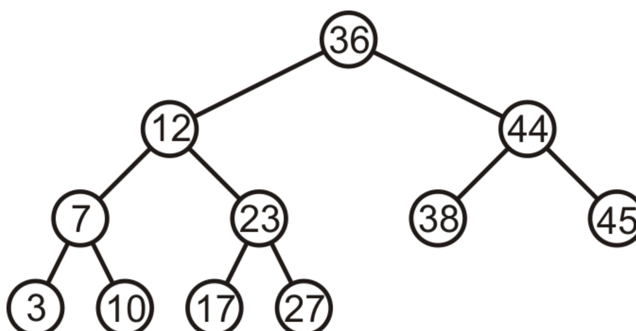

Figure E.4. An AVL tree.