University of Waterloo
Department of Electrical and Computer Engineering
ECE 250 Data Structures and Algorithms

Final Examination
2008-04-17T09:00

Instructor: Douglas Wilhelm Harder
Time: 2.5 hours
Aides: none
18 pages

Surname	Given name(s)
UW Student ID Number	Signature (in pen but read instructions first)

You may only ask one question: "May I go to the washroom?"
If you are unsure of a question, write down your assumptions and continue.
If you have insufficient space to answer a question, use the back of the previous page.
I acknowledge that I have read all these questions and will receive -5 if I fail to follow them.
Write your UW User ID in the left half of the signature block and sign in the right half for one bonus mark.
The examination is out of 130 marks.

The -5 box for T.A.s to check if a student asks a question other than the one specified.

A. Relationships

A.1 **[3]** The relationship $f(n) = Q(g(n))$ defines an equivalence relationship on functions. If we consider only functions of the form $n^k \ln(n)$ or n^k where $k = 0$, what kind of relationship does $f(n) < g(n)$ if $f(n) = \mathbf{o}(g(n))$ define?

What are the two smallest elements according to this relationship?

A.2 **[3]** Indicate the truth (write T or F next to each) of each of the following statements:

a. Directed acyclic graphs are used to store hierarchically-ordered data.
b. Graphs are used to store partially-ordered data.
c. General trees are used to store unordered data.
d. Hash tables are used to store well-ordered data.
e. Arrays are used to store adjacency relations.

B. Asymptotic Analysis

B.1 **[3]** The following functions are ordered according to $f_k(n) = \mathbf{o}(f_{k+1}(n))$. Fill in the missing places with functions which continue to satisfy this relation. (There are multiple answers.)

	$\ln(n)$		$n^{1/2}$		$n\ln(n)$		$n^{\lg(3)}$		n^2	

B.2 **[3]** Determine the relationship $n^2 = \mathbf{?}(n \ln(n))$ by using the limit definition and, where appropriate, l'Hopital's rule.

C. Algorithm Analysis

C.1 **[3]** Given that the function `int f(int n)` randomly returns a number between 0 and n^2 and the function `int g(int n)` runs in $\mathbf{O}(n^3)$ time, write a function which runs in $\mathbf{O}(n^5)$ time.

```
void h( int n ) {




}
```

D. Stacks, Queues, and Deques

D.1 **[5]** Implement a member function `void double_capacity()` which doubles the size the array of a stack when a stack is full. Assume that the first element placed into the stack is placed into array location 0.

```
template <typename Object>
class Stack {
      private:
            Object *array;
            int array_capacity;
            int array_size;
      // ...
};

template <typename Object>
int Stack<Object>::size() const {
      return array_size;
}

template <typename Object>
void Stack<Object>::double_capacity() {







}
```

D.2 **[4]** If the size of the array is doubled each time it is filled, you can calcluate the number of copies required to insert $n = 2^m$ items by summing

$$\sum_{k=0}^{m-1} 2^k = 2^m - 1 = n - 1$$

If, however, you increase the size of the array by a multiple d then the number of copies required to insert $n = d^m$ elements is

$$\sum_{k=0}^{m-1} d^k = \frac{d^m - 1}{d - 1} = \frac{n-1}{d-1}$$

For example, if you multiply the array capacity by $d = 4$ each time the array is full, we would need only 1/3rd the number of copies, *i.e.*, $(n - 1)/3$, as compared to when we double the size of the array, *i.e.*, $n - 1$.

Why not always use a larger value of d?

Are there circumstances where we might want a value of $d < 2$?

E. General Trees, Binary Trees, and Tree Taversals

E.1 **[3]** Prove by using induction that the number of leaves in a perfect *N*-ary tree of height *h* is N^h.

E.2 **[5]** Write the insert function for a binary node class. If the argument already appears in the tree, do not insert the element.

```
template <typename Object>
class BinaryNode {
    private:
        Object element;
        BinaryNode *left;
        BinaryNode *right;
        // ....
    public:
        BinaryNode( const Object & obj ):element(obj){};
};

template <typename Object>
void BinaryNode::insert( const Object & obj ) {





}
```

E.3 **[2]** Perform pre-order traversal of the tree shown in Figure E.3 by placing your answer in Table E.3.

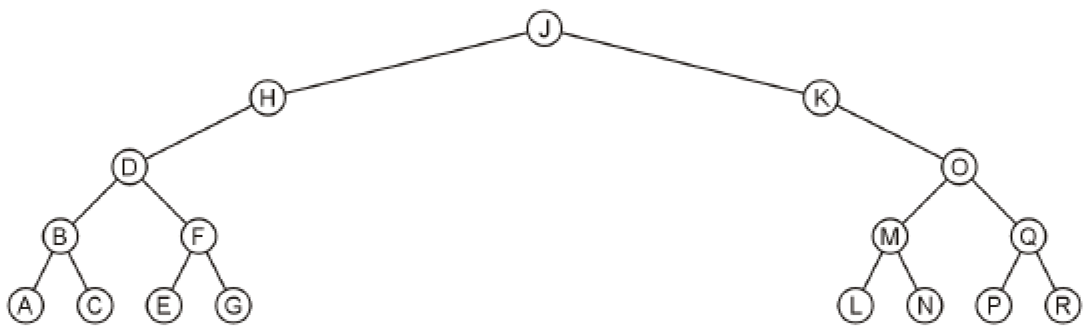

Figure E.3 A binary tree.

Table E.3

E.4 **[2]** Given the same tree in Figure E.3, perform a post-order traversal, **but visit only the left sub-tree if the current node contains a vowel**. Place your answer in Table E.4.

Table E.4

F. AVL Trees

F.1 **[3]** What are the defining properties of an AVL tree?

F.2. **[8]** Starting with the AVL tree shown in Figure F.2, perform the following three insertions.

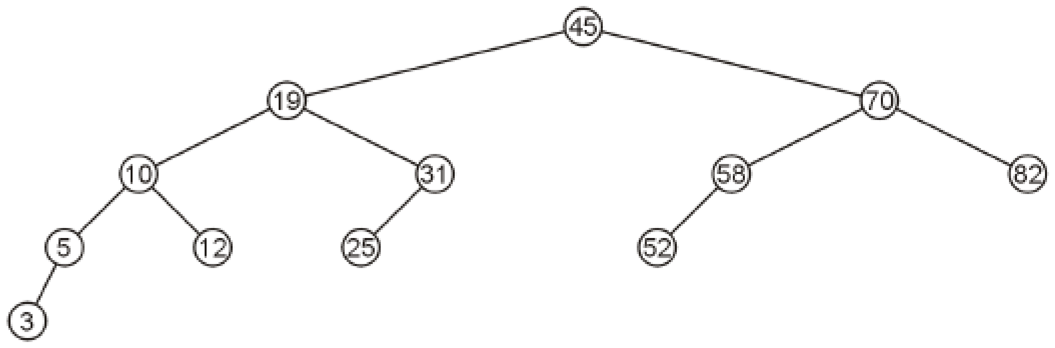

Figure F.2. An AVL tree.

F.2a Insert the value 2 into the AVL tree shown in Figure F.2.

F.2b Insert the value 1 into the result of Question F.2a.

F.2c Insert the value 4 into the result of Question F.2b.

F.3 **[5]** Which nodes can be removed (individually) from the AVL tree shown in Figure F.3 without causing an AVL imbalance? Recall that, when deleting appropriate nodes, we copy up the minimum element of the right sub-tree. Other cases are easier.

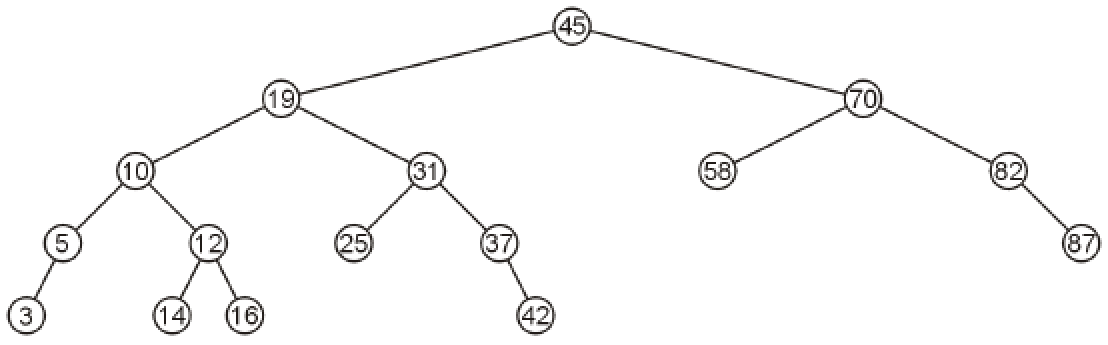

Figure F.3 An AVL tree.

G. B Trees

G.1 **[10]** Come up with a number of test cases for your B-Tree implementation. You should test as many possible **different** scenarios as possible. I want you to restrict your testing to the following `BTreeTester` commands

```
empty b         empty           the result is the boolean value b
root            root            the root is not 0 and sends the next instructions
                                (until the command exit) to BTreeNodeTester for the root
insert n        insert          the element can be pushed onto the front (always succeeds)
```

and the following `BTreeNodeTester` commands

```
leaf b          leaf            the result is the boolean value b
subtree n       subtree         the nth subtree is not 0 and sends the next
                                instructions (until the command exit) to BTreeNodeTester
                                for that subtree
retrieve n m    retrieve        the nth element in the tree is m
```

You should recall that you need `new`, `delete`, and `exit`, and your B-Tree had $M = L = 5$. You will be tested on the thoroughness of your

H. Hash Tables

H.1 **[3]** Insert the elements 532, 412, 582, 984, 471, 458, 874, 172, 283 (in this order) into the initially-empty hash table Table H.1 containing 16 bins using linear probing. The hash function is the least-significant digit.

Table H.1

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

H.2 **[3]** Insert the elements 532, 412, 582, 984, 471, 458, 874, 172, 283 (in this order) into an initially-empty hash table shown in Table H.2 containing 16 bins using double hashing. The first hash function is the least-significant digit while the second hash function (the jump) is an appropriately modified variation of the middle digit.

Table H.2

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

H.3 **[2]** Why did I choose a hash table of size 16 for Question H.2?

J. Priority Queues and Heaps

J.1 **[6]** Suppose you wanted to implement a priority queue, and you had a choice of using:

a. an appropriately-modified queue using an array,
b. an appropriately modified AVL tree which has an **`Object get_min() const`** member function, and
c. a binary min heap.

Show the average run times of the following operations on these three data structures in Table J.1. Note that there are two possible answers for using the array implementation. You should choose the *better* implementation.

Table J.1

	`void enqueue(..)`	`Object dequeue()`	`Object head()`
queue-as-array			
AVL tree			
binary min-heap			

K. Sorting Algorithms

K.1 **[4]** This implementation of insertion sort has two logical errors. Indicate the errors and write down the correct code. A sample unsorted array is provided.

```
int insertion( int * array, int n ) {
        for ( int i = 1; i < n; ++i ) {
                for ( int j = i; j >= 0; --j ) {
                        if ( array[j - 1] > array[j]  ) {
                                int tmp = array[j];
                                array[j - 1] = array[j];
                                array[j - 1] = tmp;
                        } else {
                                break;
                        }
                }
        }
}
```

0	1	2	3	4	5
31	14	36	15	57	7

K.2 **[4]** Convert the array shown into Table K.2a into a max-heap using in-place heapification and place your answer in Table K.2b. Note that in this case, we do not leave the array entry 0 blank.

Table K.2a

32	18	93	49	72	84	31	53	27	33	42	10

Table K.2b

K.3 **[3]** Justify or argue against the following statement: because merge sort is the fastest of the **O**(n ln(n)) algorithms, it is always the optimal choice for performing sorts under any circumstances in a small embedded system.

K.4 **[5]** Perform **one step** of quick sort on the list of 15 integers shown in Table K.4a using the median-of-three selection of the pivot. Place your answer in Table K.4b. I've marked the middle entry and you have two blank tables for your rough work.

Table K.4a

							x							
85	43	53	72	41	95	47	23	95	52	84	32	53	72	82

pivot = _____

Table K.4b

K.5 **[1]** If you wanted to use bucket sort to sort an array of *n* numbers which have values between 0 and *m* – 1, what is the additional memory requirement (using Landau symbols)?

M. Graph Algorithms

M.1 **[4]** The following are a list of ECE courses with their prerequisites.

ECE 100	
ECE 150	
ECE 222	(Prereq ECE 150 and ECE 223)
ECE 223	(Prereq ECE 150)
ECE 250	(Prereq ECE 150)
ECE 241	(Prereq ECE 100)
ECE 342	(Prereq ECE 100, ECE 241, MATH 211)
ECE 327	(Prereq ECE 222, ECE 223)
MATH 117	
MATH 119	(Prereq MATH 117)
MATH 211	(Prereq MATH 119)
MATH 212	(Prereq MATH 211)

Draw a DAG where (ECE *NNN*) ? (ECE *MMM*) indicates that ECE *NNN* is a prerequisite of ECE *MMM*.

M.2 **[3]** Do a topological sort on the DAG you created in Question M.1 to show how an arts student could take one course per term to take all these courses in such an order as to satisfy all of the prerequisites. Place your answer into Table M.2. The student should, where possible, finish all first-year courses before starting second year courses, and finish all 2nd-year courses before taking the third-year course.

Table M.2

M.3 **[6]** Figure M.3 is a weighted undirected graph with edges labeled A-V together with numeric weights. Perform Prim's algorithm to find the minimum spanning tree (starting with the darkened vertex) and place the edges you added to the minimum spanning tree in the order you found them into Table M.3.

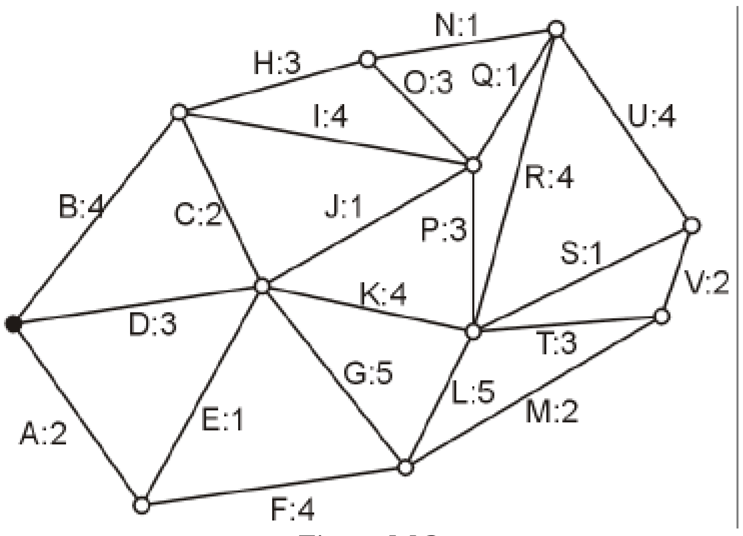

Figure M.3

Table M.3

N. Algorithms Design

N.1 **[4]** For the general divide-and-conquer recurrence relation $\mathrm{T}(n) = a\mathrm{T}(n/b) + \mathbf{O}(n^k)$, is it always true that $a = b$? If your answer is "yes", provide a proof.

N.2 **[3]** Show how we can go from the summation

$$\mathrm{T}(b^m) = a^m \sum_{\ell=0}^{m} \left(\frac{b^k}{a}\right)^{\ell}$$

to the formula $\mathrm{T}(n) = \mathbf{O}(n^k \ln(n))$ when $a = b^k$.

N.3 **[4]** Suppose that in a wireless network (a graph), one computer A is aware of the physical location of:

a. all computers with which it can directly communicate, and
b. the computer B to which it wishes to send a message.

Assume that it may not be possible for the computer A to directly communicate with computer B. Suggest a greedy algorithm for getting the message to computer B in an efficient (if not optimal) manner. What information must be passed on?

N.4 **[4]** Draw a picture of a graph where your greedy algorithm proposed in Question N.3 would not find the shortest path (as counted by the number of hops from computer to computer). Comment as to the likelihood of such a scenario occurring.

N.5 **[3]** Indicate (by circling) which pointers (arrows) would be accessed (either to follow or to check if they are 0) when determining if the value 50 is in the skip list shown in Figure N.5.

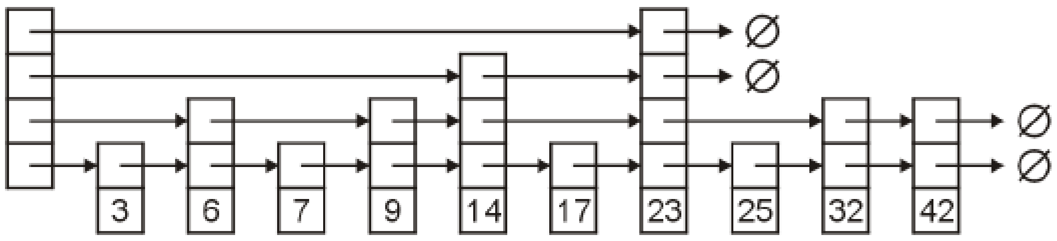

Figure N.5. A skip list.

P. Sparse Matrices

P.1 **[4]** Given the following square 4 × 4 sparse matrix using the old-Yale representation, recreate the original matrix and place your answer in Figure P.1.

0	1	2	3	4
0	3	5	7	9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	1	2	1	2	2	3							

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1.2	3.4	5.6	7.8	9.0	2.1	4.3	6.5	8.7							

$$\begin{pmatrix} \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \\ \_\_ & \_\_ & \_\_ & \_\_ \end{pmatrix}$$

Figure P.1

P.2 **[2]** What would change in the representation of the matrix in Question P.1 if you set the entry $a_{33} = 0$?

Q. Unix (bonus)

Q. 1 **[4]** Half a mark for each Unix command for a maximum score of 4. -1/2 for each DOS command which is not a Unix command.

R. **Self Study (bonus)**

R.1 **[2]** The following array is used to represent a disjoint set.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Recall that the disjoint sets {0}, {1}, {2}, {3}, {4}, ..., {15} is represented by

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	1	2	12	13	14	1	7	8	3	10	7	12	5	14	5

Are 3 and 4 in the same set?

Are 4 and 5 in the same set?

Recall that the disjoint collection of sets {0}, {1}, {2}, {3}, {4}, ..., {15} is represented by the array entries

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

while the disjoint collection of sets {0, 1}, {2, 3}, {4, 5}, {6, 7}, {8, 9,10,11,12,13,14,15} could be represented by

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	3	3	4	4	7	7	9	9	9	9	9	10	9	13