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

Final Examination
2009-04-15T9:00P2H30M PAC 5, 6

Instructor: Douglas Wilhelm Harder
Time: 2.5 hours
Marks: 158 + 4 bonus

Surname (last name)	Given name(s)
UW Student ID Number	UW User ID @engmail

Instructions:

This cover page must be filled in with pen.
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.
If you have insufficient space for your answer, continue on the back of the previous page.
Do not leave during the first hour 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.
Bonus of 1 for obeying all these instructions.
You must stop writing when you are told that the examination is finished. No exceptions, not even for your name. (-5 mark penalty.)

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

I have read and understand these instructions.

Signature: ____

A. Relationships, Asymptotic and Algorithm Analysis

A.1 [4] For every set of statements, circle the correct one.

a. every partial ordering is a hierarchical ordering
   every hierarchical ordering is a partial ordering

b. every well ordering is a hierarchical ordering
   every hierarchical ordering is a well ordering

c. every equivalence relation is an adjacency relation
   every adjacency relation is an equivalency relation
   neither of these two previous statements is true

d. every equivalence relation is a partial ordering
   every partial ordering is an equivalence relation
   neither of these two previous statements is true

A.2 [3] Fill in the most appropriate Landau symbol:

$n =$ ______$(n^{\ln(2)})$

$n =$ ______$(n^{\ln(4)})$

$n^{\ln(2)} =$ ______$(2^{\ln(n)})$

$\ln(n) =$ ______$(\lg(n))$

A.3 [3] Verify, using limits and the appropriate mathematical rules, that $n = \mathrm{W}(\lg(n))$. You may use information from Question A.2 if it will help.

B Linked Lists and Arrays

B.1 [5] Suppose we begin with a linked list `List` class where only the `Node *list_head` is stored in the linked list class and each node is singly linked. Suppose the class has the following member functions:

```
empty  size  member  front  push_front  pop_front  back  push_back  pop_back
```

Which member functions will run in **O**(1) time?

Which member functions will run in **O**(1) time only after a `int list_size` member variable (initially 0) is added to the `List` class?

Which member functions will run in **O**(1) time only after a `Node *list_tail` member variable is added to the `List` class?

Which member functions will run in **O**(1) time only after each node is made into a doubly linked node?

Which member functions will run in **O**(ln(n)) time only if the list is made into a skip list?

B.2 [5] With any data structure stored as an array, we can double the size of the array if the array is full or we can increase the size of the array by one if the array is full. Using asymptotic analysis, which is a better design?

With many of the data structures covered in class which are stored as an array, if the array is full, we can double the size and then if the array is too empty, we can halve the size of the array. Is it better to halve the size of the array when the array is half empty or is it better to halve the size of the array when the array is three-quarters empty? Use asymptotic analysis in your answer.

B.3 [5] Write a function `Object Stack_as_array::pop()` which removes the top of the stack and returns that object. If, after removing the top, the array size is one quarter the capacity **and** the capacity is greater than `initial_capacity`, then resize the array to be one-half the current capacity.

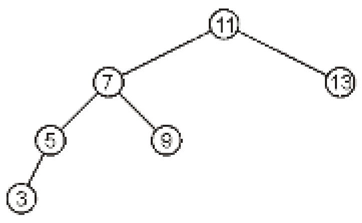
```
template <typename Object>
class Stack_as_array {
       private:
              int stack_capacity;      // int capacity() returns this
            int initial_capacity;
              Object *array;
              int stack_size;          // int size() returns this
       public:
              Stack_as_array( int n ):array_size(n), initial_capacity(n),
              array( new Object[array_size] ), stack_size( 0 )
              { /* empty */ }

              Object pop();
};

template <typename Object>
Object Stack_as_array<Object>::pop() {













}
```

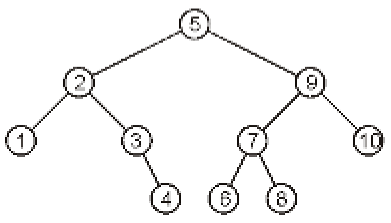
B.4 [2] What does it mean for a member function to be declared `const`? Why could the member function in Question B.3 not be declared as `Object pop() const`?

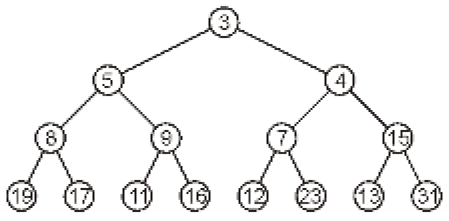
C. Trees

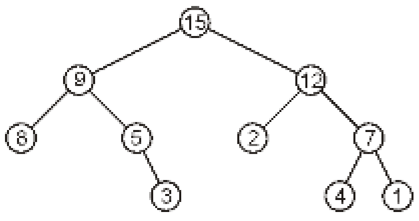
C.1 [3] Write a proof by induction that the number of leaves in a perfect tree of height *h* is 2^h.

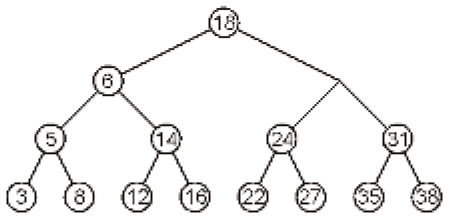
C.2 [4] For each of the following trees, indicate whether they are binary search trees, AVL trees, min-heaps, max-heaps, or none of these. Write down the best answer below the corresponding the tree. -1 per wrong answer.

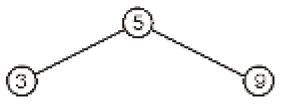












C.3 [5] Insert the seven values 6, 2, 1, 4, 3, 5, 7 in this order, into an initially empty AVL tree. Show the result of each insertion.

C.4 [4] Give the definition of a B-tree where internal nodes contain *M* next pointers and external nodes contain *L* entries. What are the constraints on the number of entries in the nodes, both internal, external, and the root? What does a B-tree with only one entry look like?

C.5 [2] Explain why all information associated with a key is stored in the leaf nodes of a B-tree.

D. Hash Tables

D.1 [5] What are all the problems with the following code which attempts to double the size of the array used in a hash table using linear probing? Assume `capacity()` returns the member variable `array_size`. Some errors may have different marking weights.

```
void Hash_table::double_array_size() const {
    Object *new_array = new Object[2*capacity()];

    for ( int i = 0; i <= capacity(); ++i ) {
        new_array[i] = array[i];
    }

    delete array;
    array = new_array;
    array_size *= 2;
}
```

D.2 [2] In double hashing, why do we require that the jump size is relatively prime to the number of bins. Give an example of what happens if they are not relatively prime.

D.3 [8] The following code shows the constructor of a chained hash table class:

```
template <typename Object>
Chained_hash_table::Changed_hash_table( int n ):
array_size( std::max( n, 1 ) ),
array( new Single_list<Object>[array_size] ) {
    // does nothing
}
```

Assume that the function

```
   int Chained_hash_table::hash( Object const &obj, int array_size ) const;
```

returns a value between `0` and `array_size – 1`.

Implement the functions `bool insert( Object const & )`, `bool member( Object const & ) const`, and `bool remove( Object const & )` with the stated constraints.

```
// return false if the object is already in the hash table
// and do not insert it; otherwise insert at the front of the
// appropriate linked list and return true
template <typename Object>
bool Chained_hash_table::insert( Object const & obj) {















}

// determine if the object is in the hash table (true or false)
template <typename Object>
bool Chained_hash_table::member( Object const & obj ) const {











}

// return false if the object is not in the hash table; otherwise
// remove the object from the appropriate linked list and return
// true
template <typename Object>
bool Chained_hash_table::remove( Object const & obj ) {















}
```

E. Heaps and Priority Queues

E.1 [3] Describe all the benefits of using an array (starting at array location 1) to store a complete binary tree to represent a heap as opposed to using an AVL tree to store the entries of a priority queue. Comment on memory requirements and on the run times of the `enqueue`, `dequeue_min`, and `head` (of the priority queue) member functions.

E.2 [3] Insert the three values 3, 9, 6 into the binary min heap shown in Table E.2a. Place your answer in Table E.2b. You must show your work for partial marks if your answer is not correct.

Table E.2a. A binary min heap.

	1	8	5	17	12	24	7								

Table E.2b Your answer.

E.3 [4] Dequeue the minimum element three times from the binary min heap shown in Table E.3a. Place your answer in Table E.3b. You must show your work for partial marks if your answer is not correct.

Table E.3a. A binary min heap.

	2	4	6	9	7	25	8	18	10	13					

Table E.3b Your answer.

E.4 [2] Which integer values could be placed in the two question marks (?) to make the array in Table E.4 a min heap?

Table E.4 A partial binary min heap.

	8	?	?	19	14	26	11								

F. Sorting

F.1 [3] What are the maximum and minimum run times required to sort an array of size n with insertion sort? Describe what the entries in the array are like before sorting takes place to achieve these run times. Write down the formula which relates run time to the size of the array being sorted and the number of inversions d.

F.2 [3] Why do we have to know that $\ln(n!) = Q(n \ln(n))$ in order to analyze the asymptotic run time of heap sort?

F.3 [6] Show how you would apply quick sort on the following unsorted array. If a sub-array has only four or fewer entries, just sort the entries for the next step. Indicate the steps you are taking, your choice of pivot (median of three with correct integer division, *i.e.*, use $(a + b)/2$ where $a = 0$ and $b = n - 1$ to start with), *etc.* A number of tables are given for you to show your work.

0	1	2	3	4	5	6	7	8	9	10	11
10	17	3	16	18	5	8	1	9	15	0	14

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

0	1	2	3	4	5	6	7	8	9	10	11

F.4 [4] What algorithm does this implement? Briefly explain what is happening.

```
void function( int *a, int b, int c ) {
    int *d = new int[b];
    int e = 1;
    for ( int f = 0; f < c; ++f, e <<= 1 ) {
        int g = 0, h = b – 1;
        for ( int i = 0; i < b; ++i ) {
            if ( a[i] & e ) {
                d[h] = a[i];
                --h;
            } else {
                d[g] = a[i];
                ++g;
            }
        }

        int i = 0;
        for ( ; i < g; ++i ) {
            a[i] = d[i];
        }

        for ( int j = b – 1; j > h; --j, ++i ) {
            a[i] = d[j];
        }
    }

    delete [] d;
}
```

G. Graphs Algorithms

G.1 [3] What is the purpose of a topological sort on a DAG defined by a number of tasks and constraints on the order in which the tasks can be performed?

G.2 [5] Figure G.2 shows a DAG where each node is associated with a run time given in seconds. If these tasks were run on a single processor, the run time would be the sum of the individual run times; however, if there are multiple processors, it is possible to run multiple tasks on different processors simultaneously where the tasks do not depend on each other. What is the minimum time in which these processes can be run assuming we can use as many processors as we need?

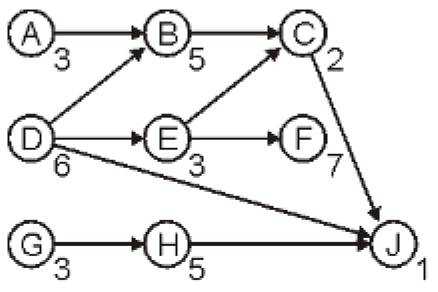


Figure G.2 A DAG of nine processes with run times.

G.3 [4] Justify or draw a counter-example for the following claim: In finding the shortest path between two vertices in a graph, all one needs do is find a minimum spanning tree and because the edges of that minimum spanning tree allow for only one connection between any two vertices, the shortest connection on the minimum spanning tree will be the shortest path between the two nodes in the original graph.

G.4 [1] In a graph with *n* vertices, assume that each edge has weight 1. What is the significance if the minimum spanning tree starting at one vertex has weight $n - 1$?

H. Algorithms

H.1 [4] Given a 30-day project period with 9 possible projects as listed below (with estimated completion times and estimated expected revenue), determine which projects you may consider placing into your schedule.

Project	**Expected Time** (days)	**Expected Revenue**
A	6	30000
B	3	21000
C	8	64000
D	6	36000
E	4	26000
F	5	32000
G	10	62700
H	1	3000
J	5	15000

H.2 [5] Given the divide-and-conquer run time of

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

where a, b, and k are fixed, what are the conditions on a, b, and k where $n = b^m$ to produce run times of $\mathrm{T}(n) = \mathbf{O}(n^{\log_b(n)})$ and $\mathrm{T}(n) = \mathbf{O}(\log_b(n)n^{\log_b(n)})$, respectively?

H.3 [4] Show how you would simply the multiplication of these two numbers using the divide-and-conquer method shown in class which has a run time of $\mathrm{T}(n) = 3\mathrm{T}(n/2) + \mathrm{O}(n)$.

$$42002500 \times 40002000$$

Recall that $ad + bc = ac + bd - (a - b)(c - d)$. You need only apply one step of this method after which you can just multiply the two numbers.

H.4 [2] Given the random number generator `random()` which returns a random number between 0 and $2^{31} - 1$, write a function which returns a random double-precision floating-point number which falls on the open range (0, 1). The variable `INT_MAX` stores the value $2^{31} - 1$.

J Sparse Matrices (no I)

J.1 [4] Find the 6×6 matrix represented by the three arrays `IA`, `JA`, and `A` and write your answer in the matrix provided by Table J.1.

`IA`

0	1	2	3	4	5	6
0	2	4	5	7	7	9

`JA`

0	1	2	3	4	5	6	7	8
2	3	1	5	4	1	2	0	4

`A`

0	1	2	3	4	5	6	7	8
5	10	15	20	25	30	35	40	45

Table J.1

K Bonus Question (do either K.1 or K.2 but not both)

K.1 [4] What are the disjoint sets in Table K.1?

Table K.1

0	1	2	3	4	5	6	7	8	9
0	3	4	3	7	5	9	7	5	3

How would you take the union of the set containing 1 and the set containing 2?

K.2 [4] Answer the following two questions:

When a new element is inserted into a splay tree, where should it be placed?

When an element is accessed in a splay tree, where is it, through a series of rotations, moved to? What are these rotations called with respect to manipulating splay trees?