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

Final Examination
2007-12-07

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

Surname	Given name(s)
Student ID Number	Signature (in pen after reading comments below) x

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 ask any question, you will be given a deduction of 5 marks.
If you have insufficient space to answer a question, use the back of the previous page.
The examination is out of 142 marks. It will be graded out of 130.
Start your signature at the x for 1 bonus mark.
You may not leave in the first hour or during the last 15 minutes of the examination.

1. **[4]** List the following functions in order in the provided table so that $f_i(n) = W(f_{i+1}(n))$. If you get the answer backwards, you will get zero.

$$3 + n, \quad \lg(n), \quad n^2\ln(n), \quad n - n^2, \quad n^{\lg(3)}, \quad n^{\ln(2)}, \quad 5$$

2. **[4]** If `a` is an array of size *m* and where each entry `a[i]` is between 0 and *n* – 1, what is the worst-case run time of the following code fragment?

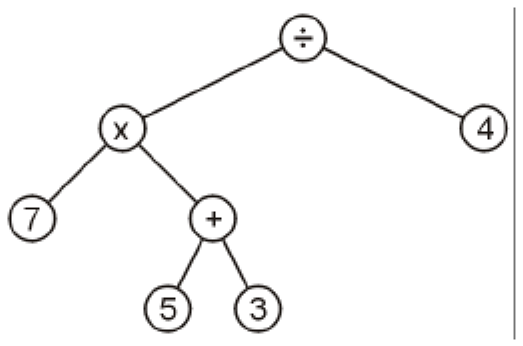
```
k = 0;

for ( int i = 0; i < m; ++i ) {
     for ( int j = 0; j < a[i]; ++j ) {
          b[k] = i;
          ++k;
     }
}
```

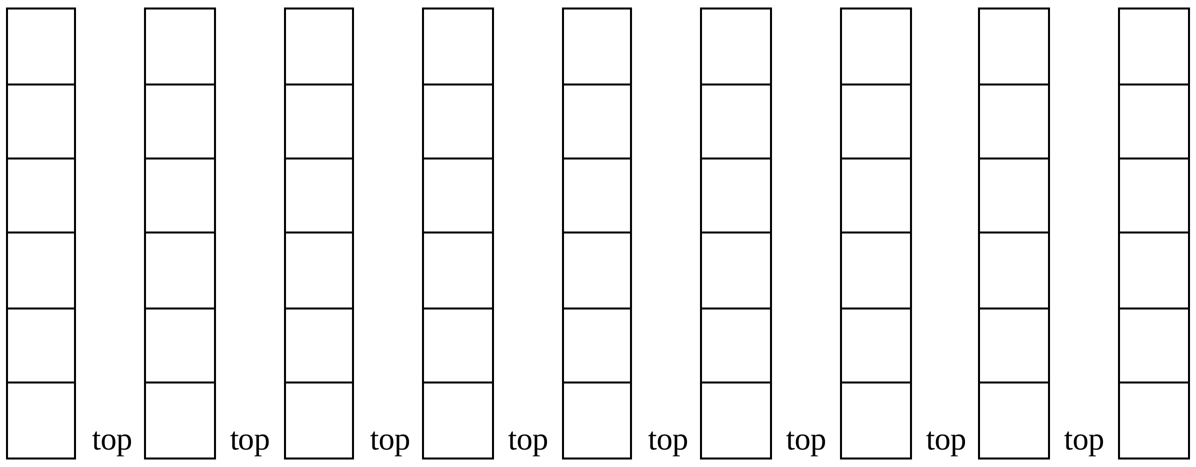
If it is also known that the sum of the entries in the array `a` equals *n*, what is the run time of the above code fragment?

3. **[5]** Implement a function export which returns a new array containing all of the elements in the singly linked with the element at the head of the linked list in the first entry of the array.

```
template <typename Object>
class SingleNode {
    public:
        SingleNode( const Object &, SingleNode * = 0 );
        SingleNode * next() const;
        Object * retrieve() const;
};

template <typename Object>
class SingleList {
    private:
        SingleNode<Object> * list_head;
        SingleNode<Object> * list_tail;
        int count;
    public:
        Object * export() const;
        // ...
};

// your implementation here
```

4. **[3]** Demonstrate how the following expression tree may be evaluated using a stack and the appropriate traversal. Show the state of the stack after any push operation onto the stack in a separate stack, starting with the stack on the left.





5. **[8]** Given a dynamically resizing queue, implement the insert function which doubles the size of the array if the array is full. You may create a helper function on the reverse of the previous page if that will simply your task.

```
template <typename Object>
class DynamicQueueAsArray {
    private:
        Object * array;
        int array_size;
        int initial_size;
        int head;         // stores the index of the head of the queue
        int tail;         // stores the index of the tail of the queue
        int count;
    public:
        void insert( const Object & );
        // ...
};

// place your implementation here
```

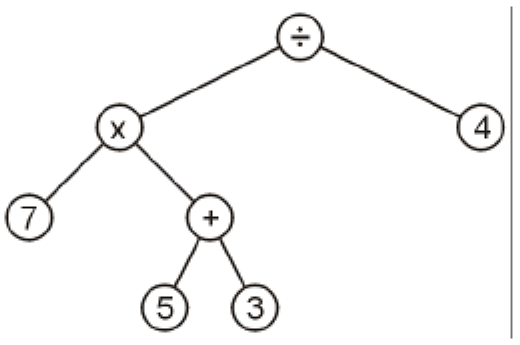
6. **[4]** Circle the mistakes in the following declaration of a general tree:

```
class GeneralTree {
    private
        Object element;
        SingleList<GeneralTree **> children;
    public
        GeneralTree( const Object & );
        Object retrieve() const;
        GeneralTree * get_child( int ) const;
        void append_sub_tree( GeneralTree * ) const;
        void clear() const;
}
```

7. **[3]** Assuming that a perfect N-ary tree of height h has N^h leaf nodes, show that a perfect N-ary tree of height h has $\frac{N^{h+1}-1}{N-1}$ nodes. You may use either induction or an appropriate formula.

8. **[1]** Given a complete binary tree represented as an array, which of the four traversals seen in class may be performed with negligible effort?

9. **[3]** Perform an in-order depth-first traversal of the following tree and print the contents of the node in the table in the order in which the nodes are visited.



10. **[2]** What is the problem with the output as shown in Question 9?

11. **[4]** Using lexicographical ordering, insert the following ten elements, in the given order, into an initially-empty binary search tree:

AB, EA, AC, CB, BC, EB, CA, EE, BA, C

12. **[4]** Create two binary search trees containing seven integers (your choice) where
   a. the first has optimal search properties, and
   b. the second represents a worst-case scenario for searching.

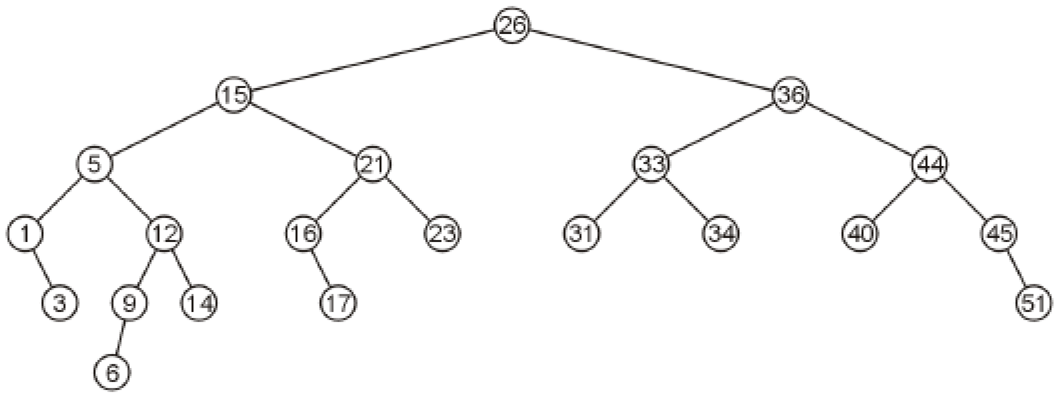
13a. **[4]** Insert the following seven elements, in the given order, into an initially-empty AVL tree. At the least, you must show as a separate tree the result of any rotation.

0, 1, 2, 3, 4, 5, 6

13b. **[2]** From your observation in Question 13a, hypothesize what the result of inserting the following fifteen elements into an initially-empty AVL tree would look like.

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

14. **[5]** Remove 16 from the following AVL tree and rebalance the tree if necessary. You need only redraw those sub-trees which changed.

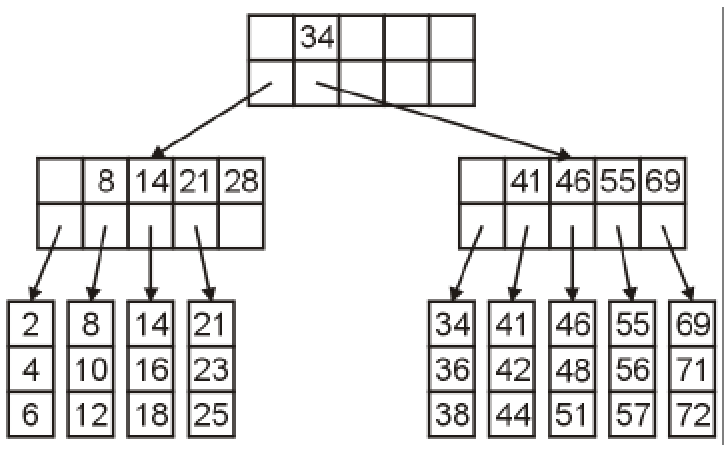


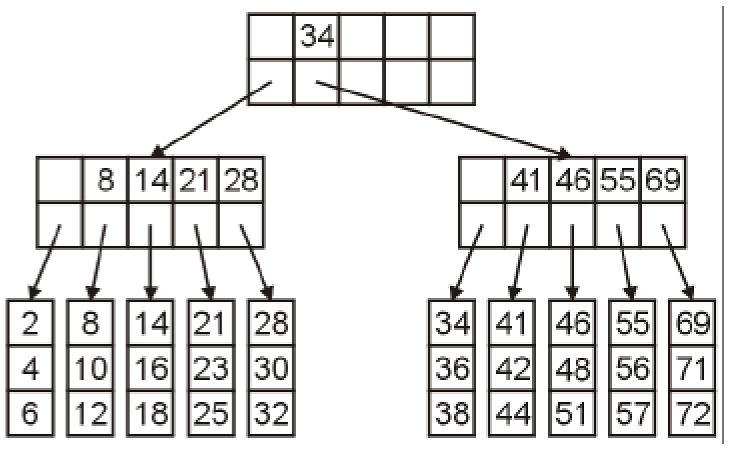
15. **[3]** Describe in words and/or images how the following two implementations of the same AVL-tree single rotation differ.

```
template <typename Object>
void AVLNode<Object>::rotate_to_right() {
     Object a = left() -> retrieve();
     left() -> element = retrieve();
     element = a;

     AVLNode<Object> * b = right();
     right_tree = left();
     left_tree = right() -> left();
     right() -> left_tree = right() -> right();
     right() -> right_tree = b;

     right() -> update_height();
     update_height();
}

template <typename Object>
void AVLNode<Object>::
            rotate_to_right( AVLNode<Object> * & to_this ) {
    to_this = left();
    AVLNode<Object> * a = left() -> right();
    left() -> right_tree = this;
    left_tree = a;

    update_height();
    to_this -> update_height();
}
```

The `element` stored in a node is returned by `Object retrieve() const`.
The `left_tree` sub-tree of a node is returned by `AVLNode * left() const`.
The `right_tree` sub-tree of a node is returned by `AVLNode * right() const`.
The function void `update_height()` updates the height of a node.

16. **[7]** Insert the element 11 into the following two B-trees. You need only redraw those nodes which are modified





17. **[4]** In the following hash table which uses linear probing, remove the element 55. The hash function is the number modulo 10.

0	1	2	3	4	5	6	7	8	9
90	07	62	13	43	25	55	72	38	84

18. **[4]** Insert the two numbers 133 and 546, in that order, into the following hash table which uses double hashing where the bin is the least-significant digit and the step size is derived from the second-least-significant digit.

0	1	2	3	4	5	6	7
	541		743	274		436	887

19. **[6]** Given a min-heap of integers which is implemented as shown in class, implement a function **`void percolate_up( int )`** which:

a. throws an **`illegal_argument()`** exception if the argument does not index a valid entry of the array,
b. if the argument indexes the top of the heap, return, otherwise
c. compare the argument indexed by the argument with its parent, and
    i. if they are out of order, swap them and call the function recursively, otherwise
    ii. return.

```
class MinHeap {
    private:
        int * array;
        int array_size;  // size of the array
        int count;       // number of elements
    public:
        void percolate_up();
        // ...
};

// your implementation here
```

20. **[5]** Dequeue the minimum element from the following three min heaps. Put your answer in the empty table below it.

	1	2	3	4	8	5	6	7	10	9

	1	2	3	6	4	8	10	7	9	5

	1	3	2	7	4	10	6	8	9	5

21. **[1]** Given an array of integers **`int * array`**, define what it means if, when **`i < j`**, the ordered pair of integers (**`array[i]`**, **`array[j]`**) forms an inversion.

22. **[2]** Perform one pass of bubble sort on the following table of unsorted numbers. Put your answer in the table below it. The last entry is provided for you.

5	3	6	2	1	4	8	7	9	0
									9

23. **[3]** The following table represents a max heap. Show the four steps required to convert it into a sorted list:

5	4	2	1	3

24. **[2]** The following table shows an intermediate step of merge sort where there are four sorted sub-arrays. Perform the next merging step and put your answer in the table provided.

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

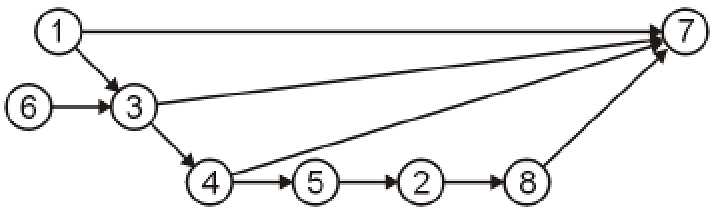
25. **[5]** The following code implements bucket sort for integers from 0 to 9 on an array **`array`** of size **`array_size`** where there may be repetitions. Finish the implementation by assigning to **`array[i]`** the appropriate values.

```
void bucket_sort( int * array, int array_size ) {
    int bucket[10];

    for ( int i = 0; i < 10; ++i ) {
        bucket[i] = 0;
    }

    for ( int i = 0; i < array_size; ++i ) {
        ++bucket[array[i]];
    }







}
```

26. **[5]** When applying radix sort to integers with 2 digits, there is one intermediate step after the first time all the entries are dequeued. The first row of the next table shows a list of unsorted numbers. Enter that intermediate list of integers into the middle row of the following table. The last row is, of course, the sorted list after all the numbers are dequeued the second time.

82	53	95	39	91	57	78	21	54	17
17	21	39	53	54	57	78	82	91	95

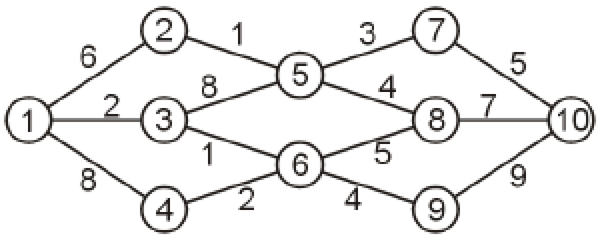
27. **[4]** Topologically sort the vertices of the following directed acyclic graph



Place your answer in this table with the first visited node in the first entry.

28. **[3]** For Prim's and Dijkstra's algorithms, what is the critical difference when we compare and update the table of stored distances?

29. **[6]** Starting with vertex 1, find the minimum spanning tree of the following graph by using Prim's algorithm. List the weights of the edges in the table below in the order in which you add the edges to the minimum spanning tree.



30. **[2]** Use a greedy algorithm to schedule the following six processes on a single processor to minimize the total completion time. Place the order in which the processes are run in the next table.

Process	1	2	3	4	5	6
Run Time	0.5 ms	0.9 ms	0.1 ms	0.4 ms	0.8 ms	0.6 ms

Processor 1						

31. **[2]** Use a greedy algorithm to schedule the same six processes in Question 30 on two processors. Place the order in which the processes are run in the next table:

Processor 1						
Processor 2						

32. **[4]** Under the assumption that $a = b^k$, simplify the formula

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

into an expression of only n, a, and b.

33. **[4]** Assume we are multiplying four matrices A, B, C, and D of sizes 10 × 2, 2 × 20, 20 × 5, and 5 × 30, respectively. In attempting to find the optimal order of the products, explain where the following numbers come from.

	A	**B**	**C**	**D**
A	0 10 × 2	400 **AB** 10 × 20	300 **A(BC)** 10 × 5	1100 **A(BC)D** 10×30
B		0 2 × 20	200 **BC** 2 × 5	500 **(BC)D** 2 × 30
C			0 20 × 5	3000 20 × 30
D				0 **CD** 5 × 30

34. **[6]** Assume that the random number generator **`std::rand()`** returns an integer in the range **`0`** and **`numeric_limits<int>::max()`**. Write a function which approximates the area of the sphere of radius 1 centred at the origin by generating *n* random points in an appropriate box. There are numerous solutions. Be sure to use C++ style casting as shown in class. If *n* < 1 throw an **`illegal_argument()`** exception.

```
double area_of_sphere( int n ) {




}
```

35. **[4]** Add the following two sparse 4 × 4 matrices which are stored using the old Yale format. Place your answer in the third set of three arrays.

1	2	4	6	8							
1	1	2	2	3	3	4					
3.0	1.0	2.0	-1.0	-4.0	2.0	7.0					

1	3	5	7	9							
1	2	1	2	2	3	3	4				
1.0	8.0	2.0	-2.0	2.0	4.0	-2.0	1.0				

1											

Recall that the entries of these arrays are indexed from 1, not 0, as matrices and vectors, in general, have their dimensions range from 1 to *n* and not 0 to *n* – 1.

36. **[3]** Write the commands necessary to:
 a. *mak*e a *dir*ectory **`myproject`**,
 b. *c*hange into that *d*irectory,
 c. edit two files **`MyArray.h`** and **`MySupport.h`** (two commands, your choice of editor),
 d. create a *t*ape *ar*chive using **`tar –cvf Array.tar`** of the two files,
 e. zip the files using *g*nu *zip*.