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

Final Examination
2006-12-21

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

Surname	Given name(s)
Student ID Number | | | | | | | | |	Signature

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.
The examination is out of 108 marks.

A. **Algorithm Analsyis**

A.1 [3] Describe, in terms of limits, what it means for:

a. f(n) = **O**(g(n))
b. f(n) = Q(g(n))
c. f(n) = w(g(n))

A.2 [3] Using limits, show that $n \ln(n) = \mathbf{o}(n^{1.5})$.

A.3 [5] Determine the run times of the following loops where f(n) runs in **O**(n) time and g(n) runs in **O**(n^2) time.

a.
```
for ( int i = 0; i < n; ++i ) {
     for ( int j = 0; j < n; ++j ) {
          f(n);
     }
}
```

b.
```
for ( int i = 0; i < n; ++i ) {
     for ( int j = 0; j < 10; ++j ) {
          f(n);
          g(n);
     }
}
```

c.
```
for ( int i = 0; i < n; ++i ) {
     for ( int j = 0; j < n; ++j ) {
          if ( j % 2 == 0 ) {
               for ( k = 0; k < n; ++k ) {
                    g(n*n);
               }
          }
     }
}
```

B. **Lists, Stacks, Queues, Deques**

B.1 [5] Given that a stack has the following member variables (all appropriately defined and initialized in the constructor):

```
Object * array;
int array_size;
int count;
```

implement the push function which doubles the size of the array if the array is filled.

```
void Stack<Object>::push( const Object & obj ) {

}
```

C. **Trees**

C.1 [6] A depth-first traversal of a directory structure can perform operations either:

1. before the subdirectories of a directory are traversed, or
2. after the subdirectories of a directory are traversed.

For each of the following operations, indicate whether the result must be computed before, after or at either point by placing a check-mark in the correct column. Assume that the only source of information passed down must be through an argument of the traversing function and the only source of information passed up is a return value from the traversing function.

	Before	After	Either
Determine and print the full path name of the directory			
Calculate the memory occupied by all files within the current directory only			
Calculate the memory occupied by all files in or below the current directory			
The depth of the current directory			
The name of the current directory			
Print a list of the files in the current directory			

C.2 [4] Fill in the nodes in the tree in Figure C.2 with the numbers 1 through 14 so that a post-order traversal would visit the nodes in the order 1 through 14.

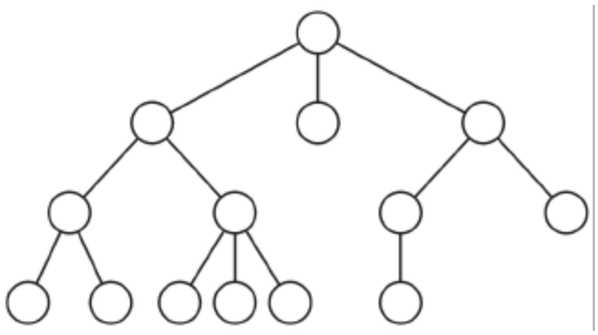


Figure C.2. A general tree.

C.3 [3] Justify each of the three statements of the recurrence relation

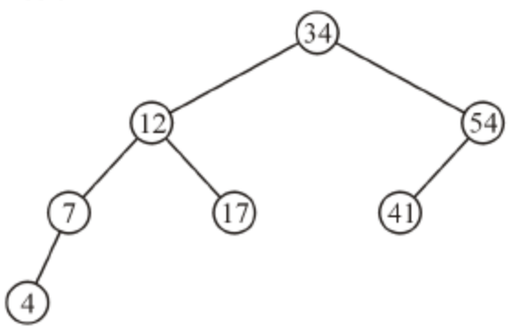
$$F(h) = \begin{cases} 1 & h = 0 \\ 2 & h = 1 \\ F(h-1) + F(h-2) + 1 & h > 1 \end{cases}$$

which describes the number of nodes in the worst-case AVL tree of height *h*.

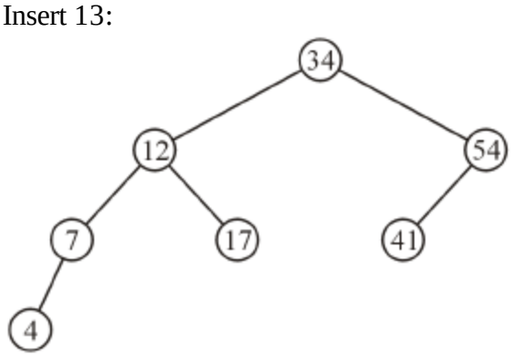
C.4 [2] Given an AVL tree of height *h*, what is the maximum number of nodes which may become unbalanced as a result of an insertion into the tree?

C.5 [9] For each of the following AVL trees, perform the given insertion together with any necessary rotations. You need only redraw that section of the tree which changes. If no rotations are required, simply draw in the new node.

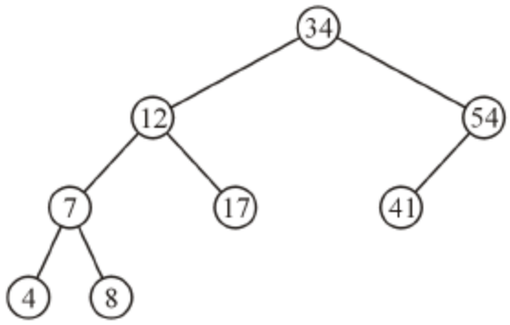
Insert 1



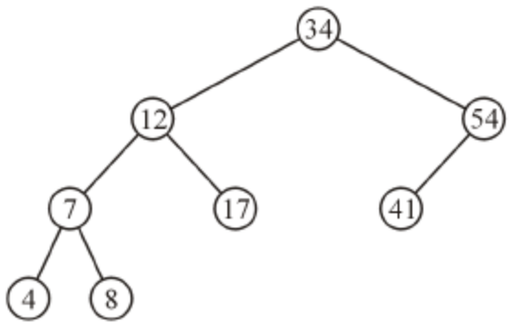
Insert 13:



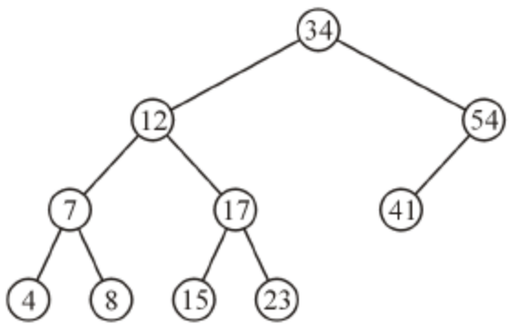
Insert 1:



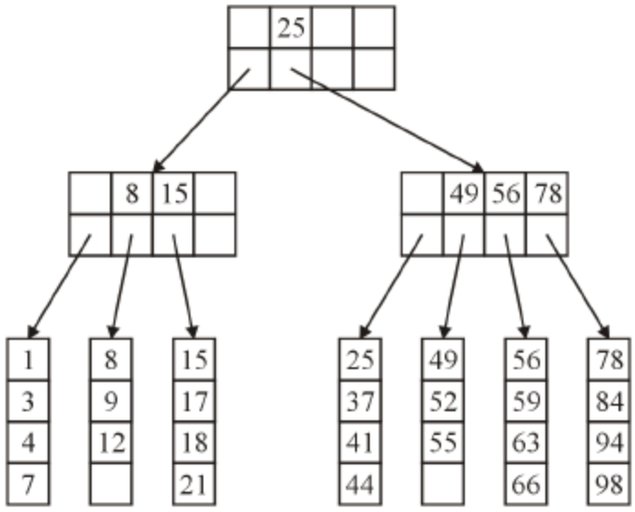
Insert 13:



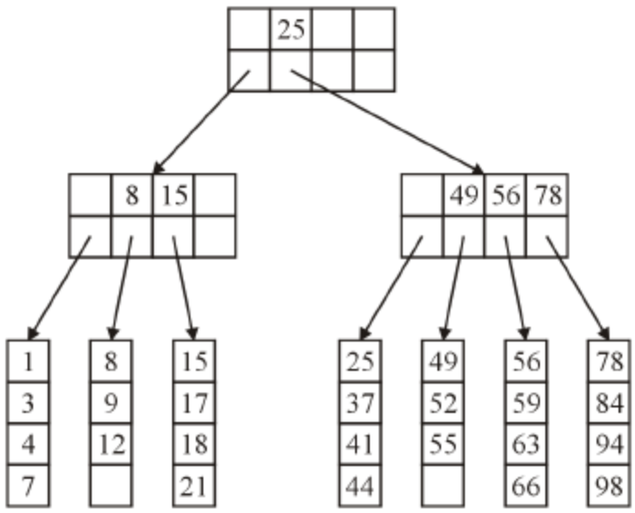
Insert 13:



C.6 [3] Insert 19 into the following B-tree by performing the appropriate split. You only need redraw those blocks which have changed. (Multiple answers are possible.)



C.7 [5] Insert 90 into the following B-tree by performing the appropriate split. You only need redraw those blocks which have changed. (Multiple answers are possible.)



D. **Hash Tables**

D.1 [2] Consider a hash table of size n which uses linear probing. Given a primary cluster of size m (assume $m << n$), what is the probability (assuming a good hash function) that an insertion will cause that cluster to grow to size $m + 1$?

D.2 [3] Remove 32 from the following hash table which uses linear probing and where the least significant digit is the hash function.

0	1	2	3	4	5	6	7
94		32	72	54	93	86	47

Place your answer into this empty table:

0	1	2	3	4	5	6	7

D.3 [4] Insert the values 24, 51, 32, 41, 53, 52, 37, 31, 32, 12 into the following hash table of size 11 using quadratic probing until either all entries are in the hash table or until one of the entries can no longer be placed into the hash table. Use the least significant digit as the hash function and indicate why you stopped.

0	1	2	3	4	5	6	7	8	9	10

D.4 [4] **(Bonus)** Print the (decimal) output from the following four statements.

```
cout << (1 << 5) << endl;
cout << (5234 | 1) << endl;   // 5234 = 1010001110010₂
cout << (5234 & ((1 << 5 ) – 1)) << endl;
cout << (5234 >> 10) << endl;
```

E. **Heaps**

E.1 [6] Using the implementation of a min-heap as discussed in class, insert the following numbers into an initially empty min-heap (draw a picture):

5, 2, 3, 6, 4, 1

Indicate the state of min-heap after each insertion by filling in the appropriate row in this table.

	0	1	2	3	4	5	6
Insert 5							
Insert 2							
Insert 3							
Insert 6							
Insert 4							
Insert 1							

E.2 [6] Suppose that an implementation of a min-heap (as described in class) which stores integers has the following member variables:

```
int * array;
int array_size;
int count;
```

The member variables are all self-descriptive and you may assume that in the constructor, the variables are appropriately assigned.

Implement the enqueue function which inserts a new number *n* into the min heap. Throw an overflow exception if the array is full.

```
void enqueue( int n ) {




}
```

F. **Sorting**

F.1 [3] The following algorithm attempts to implement insertion sort. Find and correct the three mistakes.

```
void  insertion_sort( int * array, int n ) {
    for ( int i = 1; i <= n; ++ i ) {
        int tmp = array[i]

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

F.2 [4] The recurrence relation describing the quick sort and merge sort algorithms both contain a **O**(*n*) term. Describe what the process is which requires this run time and whether or not it occurs before the recursive step. Each can be answered with one or two sentences.

F.3 [4] Apply radix sort to the following 3-bit binary numbers:

010 110 111 001 011 101 100 110

Place the intermediate lists (after dequeuing) into the following three tables. The last (sorted) list is given.

001	010	011	100	101	110	110	111

F.4 [3] **(Bonus)** In class, it was suggested that we could perform a radix sort by starting to sort the most significant digit first. Come up with a counter-example which shows that this is false. This can be done with a reasonably small example.

G. **Graphs**

G.1 [3] For each of the following lists (a-c), indicate whether it is or is not a valid topological sort of the directed acyclic graph shown in Figure G.1. Simply write “yes” or “no” next to each list.

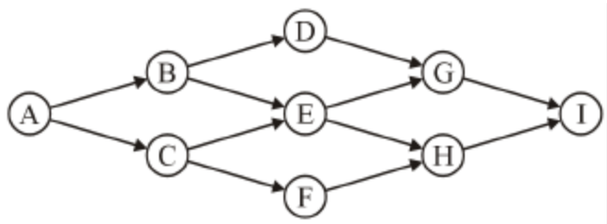
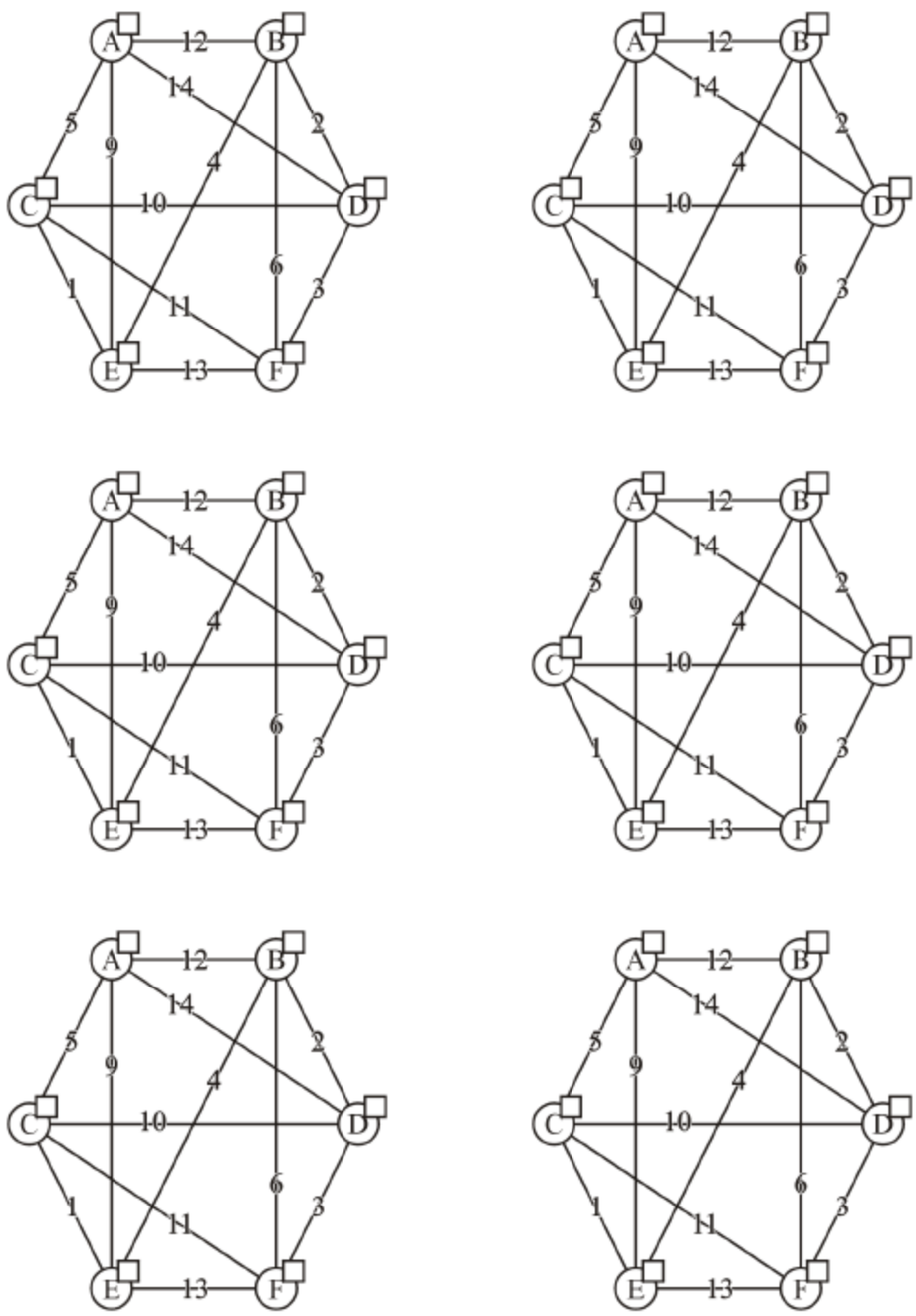


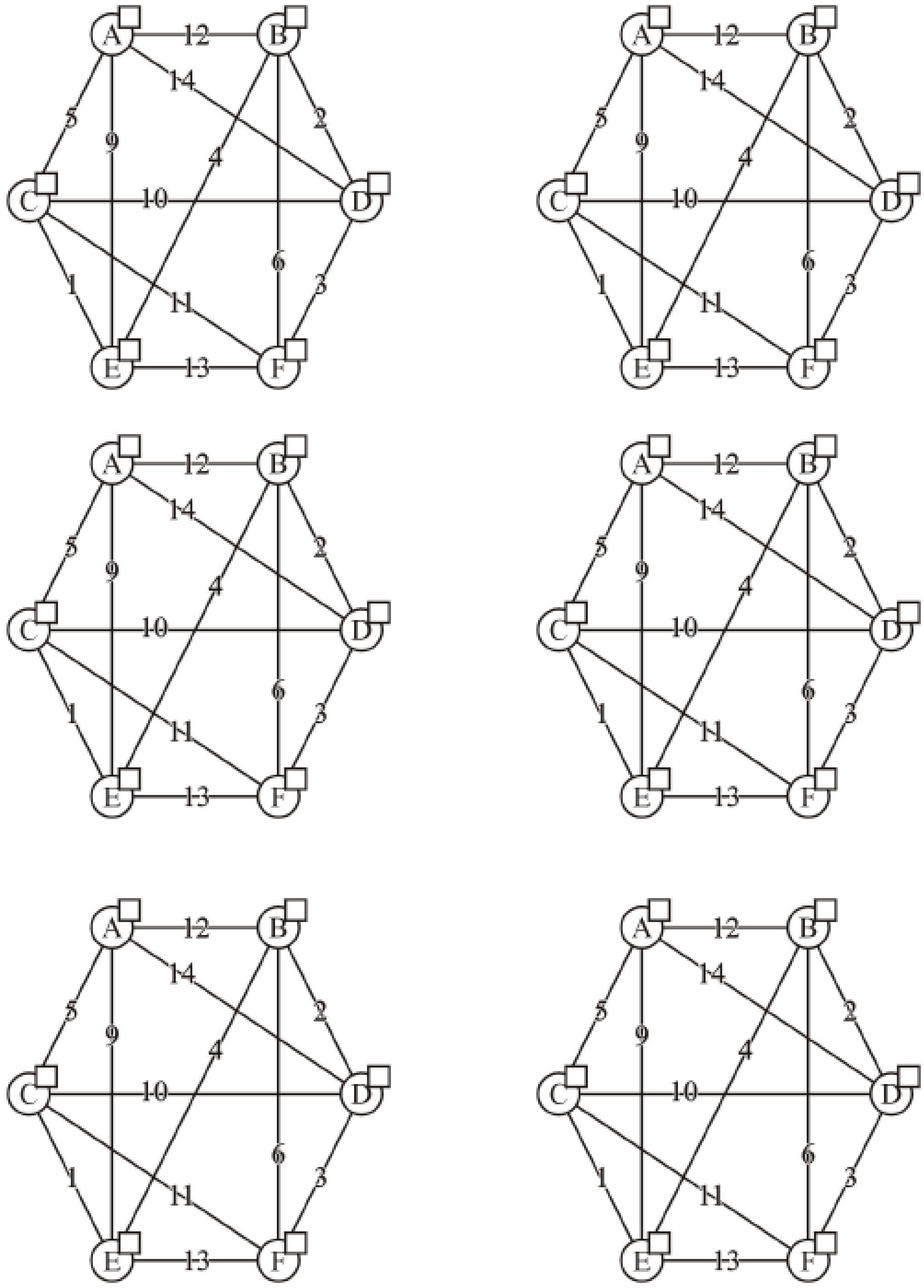
Figure G.1 A directed acyclic graph.

a. A B D C E F G H I
b. A B C D E G F H I
c. A B D C F G E H I

G.2 [6] Implement Dijkstra’s algorithm to find a minimum path from vertex A to vertex F in the following graph. Use one graph for each step. Indicate the current shortest distance for each vertex and whether a vertex has been visited (using the check box), but it is not necessary to record the actual path.



G.3 [4] Implement Prim's algorithm starting at vertex B in the following graph. Use one graph for each step in the algorithm. Draw the minimum spanning tree at the bottom of this page.



H. **Algorithms**

H.1 [4] Suppose a number of projects are being proposed for the next cycle of a product release. There are n projects, where each is associated with a projected revenue and an expected completion time. Justify heuristically why using a greedy algorithm using the highest cost density (expected revenue over completion time) is better than using a greedy algorithm which uses the highest expected revenue.

H.2 [4] For a divide-and-conquer algorithm which has a runtime $T(n) = aT(n/b) + \mathbf{O}(n^k)$ for $n > 1$, assume that $n = b^m$ and that $a = b^k$. Show how we can simplify

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

to see that $T(n) = \mathbf{O}(\log_b(n)\, n^k)$.

H.3 [3] Figure H.3 shows a skip list.

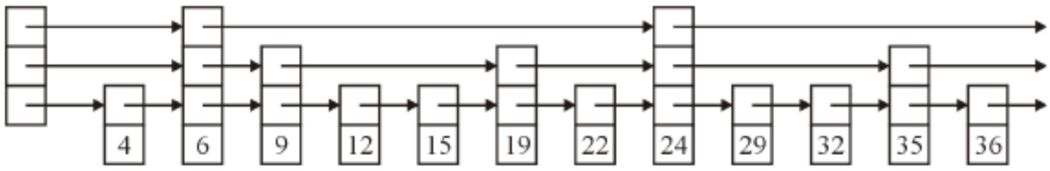


Figure H.3. A skip list.

Insert 31 into the skip list shown in Figure H.3 using the random number 010_2 to determine the height of the new node. You only need redraw the skip list from the node containing 24 to the end.

I. **Sparse Matrices (Bonus)**

I.1 [4] The following sparse matrix

$$\begin{pmatrix} 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 4 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 8 & 0 & 0 & 0 & 9 & 0 \\ 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 11 & 0 & 0 & 12 \\ 0 & 0 & 0 & 13 & 0 & 0 & 0 & 0 \\ 0 & 0 & 14 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

is stored using the following two arrays, the first storing values and the row,

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	3	10	2	4	6	8	14	13	5	11	9	7	12		
1	2	5	1	2	3	4	8	7	2	6	4	3	6		

and second storing column information:

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

Modify the these arrays as necessary to set the matrix entry (3, 4) to the non-zero value of 22. Put your answer in the following two tables:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

1	2	3	4	5	6	7	8	9
1								

J. **Self Study**

J.1 [5] Describe, in your own words, either splay trees or disjoint sets. You should describe the purpose, appropriate implementations, and properties. Provide references.