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

## Final Examination
## 2008-12-17T12:30P2H30M PAC 11, 12

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

| Surname | Given name(s) |
|---|---|
| UW Student ID Number | UW User ID |

You may only ask one question: "May I go to the washroom?"
If you are unsure of a question or if you think a question is ambiguous, write down your assumptions and continue.
If you have insufficient space to answer a question, use the back of the previous page.
No aides.
Turn off all electronic media and store them under your desk.
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.
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. C++ and Unix**

A.1 **[8]** Calling **`new`** and **`delete`** many times can be very expensive as each such call requires a call to the operating system. Suggest how you could use perhaps an array, a stack or queue, and a possibly a hash table to create a class which:

1. At initialization time allocates memory for *n* instances of a type,
2. Each of the *n* instances is designated as being *unallocated*,
3. Each time a request for new memory is made (through a function call), find an unallocated instance, mark it as allocated, and return the address, and
4. Each time memory is freed (by passing the address back to a different function call), mark the memory as unallocated.

Marks are only awarded to implementations where all operations are **O**(1) with the sole exception being the constructor which must run in **O**(*n*) time. You may use either C++ code or pseudo code.

A.2 **[2]** You have just finished implementing the **`Single_list.h`** class and copied all the appropriate files to Unix. Write down the commands which will compile the appropriate driver for and test the executable with the test file **`double.in`**. Bonus **[1]**: pipe the output to **`more`**.

**B. Relationships, Asymptotic and Algorithm Analysis**

B.1 **[4]** Associate each of the following topics in the course with the relationship associated with them:

| | |
|---|---|
| Asymptotic Q notation | |
| Merge sort | |
| Binary Search Trees | |
| Directed Acyclic Graphs | |

B.2 **[3]** Determine, using limits and the appropriate mathematical rules, the asymptotic relationship between $\ln(n^4)$ and $\ln(n)$.

B.3 **[2]** Suppose that a function runs in $T(n) = 3T(n/2) + \mathbf{O}(n)$ for $n > 1$. Why is it not asymptotically any better to improve the runtime of the additional instructions which currently take $\mathbf{O}(n)$ time and reduce them to $\mathbf{O}(1)$ time. Recall $n^{\log_b a}$, $\ln(n)n^{\log_b a} = \ln(n)n^k$, and $n^k$.

B.4 **[6]** What are the best- and worst-case run times for the following loops where the function **`f()`** may return either true or false.

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

```
for ( int i = n; i >= 0; --i ) {
      if ( f() ) {
            for ( int j = i; j <= n; ++j ) {
                  ++sum;
            }

            break;
      }
}
```

```
for ( int i = 1; i < n; ++i ) {
      for ( int j = i; j <= n; j *= 2 ) {
            ++sum;
      }
}
```

**C Linked Lists and Arrays**

C.1 **[6]** The **`Single_list`** class in Project 1 had three member variables: **`list_head`**, **`list_tail`**, and **`count`**. The **`Single_node`** class had two member variables: **`element`** and **`next_node`**.

Implement a **`swap_back`** function which interchanges the last two nodes in the linked list. Throw an **`underflow`** if the linked list has fewer than two elements. You may **not** use any other member functions implemented in the **`Single_list`** class.

```
template <typename Object>
void Single_list<Object>::swap_back() {









}
```

C.2 **[5]** Given a stack class shown below where **`count`** stores the number of elements in the stack and **`count - 1`** indicates the current top of the stack, implement a **`remove`** function which attempts to remove the argument closest to the top of the stack (if there are repeated entries, only to closest to the top should be removed). Return **`true`** if the removal was successful and **`false`** otherwise.

```
template <typename Object>
class Stack_as_array {
      int array_size;
      Object *array;
      int count;

      Stack_as_array( int n ):
            array_size( n ),
            array( new Object[array_size] ),
            count( 0 ) {
      }
};

bool Stack_as_array<Object>::remove( const Object &obj ) {









}
```

**D. Trees**

D.1 **[3]** Write a proof by induction that a perfect tree of height $h$ has $2^h$ leaf nodes.

D.2 **[3]** Given the class **`Binary_tree_node`**, implement a member function **`int print_post_order() const`** which prints the sum of the current node together with the sum of all decendants (using **`cout`**) in a post-order depth-first traversal order. A space should follow the output. The output for the tree in Figure D.2 is shown. The return value, when called on the root node, should be the sum of all entries in all nodes.

```
class Binary_tree_node {
      private:
            int element;
            Binary_tree_node *left_tree;
            Binary_tree_node *right_tree;
      public:
            int print_post_order() const;
};

int Binary_tree_node::print_post_order() const {

}
```

**Figure D.2 A tree and its output.**

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

D.4 **[3]** Remove the element D from the AVL tree shown in Figure D.4.

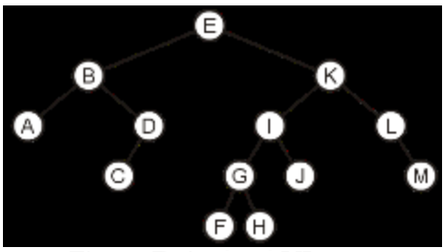



Figure D.4. An AVL tree.

D.5 **[2]** Insert E into the B-tree shown in Figure D.5 where L = M = 5. You must split full nodes.

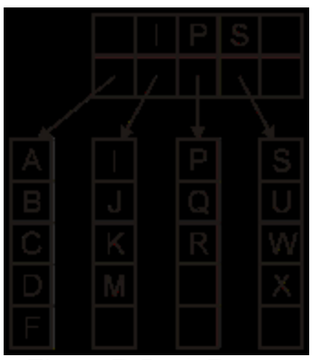



Figure D.5. A B-tree.

D.6 **[3]** Insert R into the B-tree shown in Figure D.6 where L = M = 5. You must split full nodes.

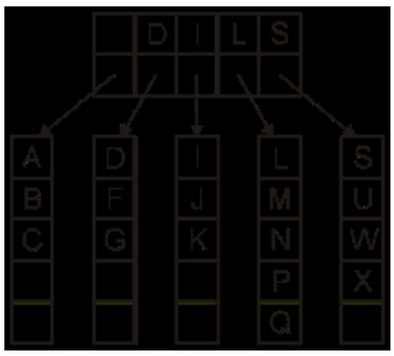



Figure D.6. A B-tree.

**E. Hash Tables**

E.1 **[2]** Insert the following eight numbers, in the given order, into a hash table using linear probing: 30, 62, 46, 21, 55, 42, 47, 60. The hash function is the least significant digit. Place your answer into Table E.1.

Table E.1

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

E.2 **[2]** Remove the number 84 from the hash table shown in Table E.2 which uses linear probing where the hash function is the least significant digit. Place your answer in the following row.

Table E.2

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 45 | 61 | 40 | | 84 | 34 | 76 | 77 |
| | | | | | | | |

E.3 **[3]** Insert the following eight numbers into a hash table using double hashing: 30, 62, 46, 21, 55, 42, 47, 60. The initial bin is the least significant digit while the secondary hash function is an appropriate modification of the most significant digit.

Table E.3

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

E.4 **[2]** Given a 32-bit int, describe what the following function returns.

```
int f( int n ) {
     int m = (n >> 8);

     return m & ((1 << 16) - 1);
}
```

**F. Heaps and Priority Queues**

F.1 **[3]** The first entry of a heap stored as an array is usually left blank (the root of the heap is at array entry 1) to simplify the formula to find the children of entry **`i`** (*i.e.*, **`2*i`** and **`2*i + 1`**) and the formula to find the parent of entry **`i`** (*i.e.*, **`i/2`**). Suppose, for example, in a heap sort, it is not possible to leave the first entry blank, that is, the root of the heap is at array entry 0. Determine formulae for the children and the parent of the entry **`i`**.

F.2 **[3]** The following is an algorithm which implements **`member`** on a binary min heap stored as an array.

```
template <typename Object>
bool Binary_min_heap<Object>::member( const Object &obj ) const {
     for ( int i = 1; i <= count; ++i ) {
          if ( array[i] == obj ) {
               return true;
          }
     }

     return false;
}
```

Describe how this could be considered to be inefficient. Without writing the code, describe how you could improve on the inefficiencies.

F.3 **[4]** Insert the three values 2 and 4, in this order, into the binary min heap shown in Table F.3a.

Table F.3a. A binary min heap.

| | 1 | 3 | 5 | 6 | 18 | 24 | 25 | 42 | 40 | 32 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Table F.3b The binary min heap after inserting 2 into Table F.3a.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Table F.3c The binary min heap after inserting 4 into Table F.3b.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

F.4 **[2]** Dequeue the minimum element from the min-heap shown in Table F.3a and put your answer in Table F.4.

Table F.4 The binary min heap after dequeing the minimum entry from Table F.3a.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**G. Sorting**

G.1 **[2]** Is the following statement true or false and state why: If an array of size *n* has only five inversions, then only five swaps are required with insertion sort. Consequently, the run time of insertion sort, in this case, is Q(1).

G.2 **[3]** Convert the unsorted list in Table G.2a into a max-heap by using the heapification algorithm shown in class. Put your answer in Table G.2b. To help, the children of 5 are 6 and 13.

Table G.2a

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 17 | 10 | 19 | 3 | 16 | 18 | 5* | 8 | 9 | 15 | 0 | 14 | 7 | 6* | 13* |

Table G.2b

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | |

G.3 **[4]** Complete the following implementation of merge sort:

```
void merge_sort( int *array, int lower, int upper ) {
      int N = upper – lower + 1;
     int *tmp_array = new array[N];

      for ( int i = lower, j = 0; i <= upper; ++i, ++j ) {
         tmp_array[j] = array[i];
      }

      merge_sort( tmp_array, 0, N/2 );
      merge_sort( tmp_array, N/2 + 1, N - 1 );

      int lindex = 0, rindex = N/2 + 1, index = 0;  // three indices

      while( left <= N/2 && right <= N – 1 ) {
            if ( tmp_array[left] < tmp_array[right] ) {


            } else {


            }


      }

      for ( int i = left; i <= N/2; ++i ) {




      }

      for ( int i = right; i <= N - 1; ++i {




      }
}
```

G.4 **[2]** In class it was argued that quick sort using the median-of-three strategy tends to divide a list into sizes *n*/3 and 2*n*/3. What happens if the list is already sorted?

G.5 **[3]** What algorithm does this implement and what are the restrictions on the arguments?

```
void function( int *array1, int n, int m ) {
      int array2[m];

      for ( int i = 0; i < m; ++i ) {
            array2[i] = 0;
      }

      for ( int i = 0; i < n; ++i ) {
            ++array2[array1[i]];
      }

      int i = 0;
      for ( int j = 0; j < m; ++j ) {
            for ( int k = 0; k < array2[j]; ++k, ++i ) {
                  array1[i] = j;
            }
      }
}
```

**H. Graphs Algorithms**

H.1 **[4]** Create a DAG of the set of vertices {A, B, C, D, E, F, G, H} where A < B, B< D, E < G, G < C, H < D, F < A, and G < A. Read “X < Y” as “X precedes Y” or “X points to Y”.

H.2 **[2]** In a topological sort of the DAG created in Question H.1, which are the vertices which could appear first?

H.3 **[4]** Find the shortest path from the left-most node to the right-most node (multiple answers may exist). How does the geometry of the graph simplify the problem? Two additional graphs are provided in case you make an error.

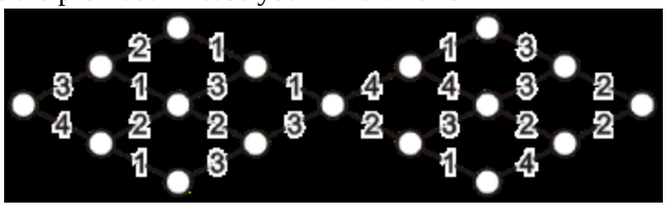
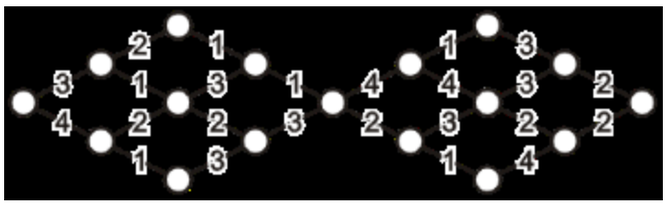
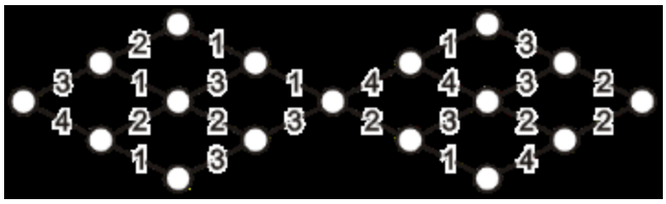


Figure H3. A weighted graph.





H.4 **[4]** Table H.4a shows a partial iteration of Prim's for finding the minimum spanning tree starting at A for the graph shown in Figure H.4. Perform the next two steps of Dijkstra's algorithm and place your answers in Tables H.4b and H.4c. The three rows of the Tables are associated with the three variables which must be tracked. There are two equally valid answers.

Table H.4a

| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **J** |
|---|---|---|---|---|---|---|---|---|
| T | T | F | F | T | F | F | F | F |
| 0 | 3 | 2 | 2 | 1 | infinity | 3 | 2 | infinity |
| - | A | E | B | B | - | E | E | - |

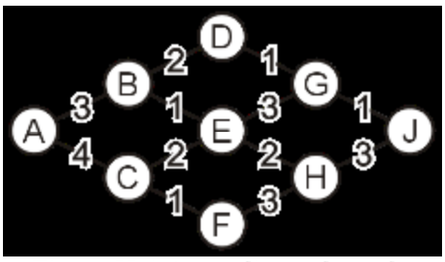



Figure H.4 An undirected graph.

Table H.4b. Vertex visited is: _____

| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **J** |
|---|---|---|---|---|---|---|---|---|
| T | T | | | T | | | | |
| 0 | 3 | | | 1 | | | | |
| 0 | A | | | B | | | | |

Table H.4c. Vertex visited is: _____

| **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **J** |
|---|---|---|---|---|---|---|---|---|
| T | T | | | T | | | | |
| 0 | 3 | | | 1 | | | | |
| 0 | A | | | B | | | | |

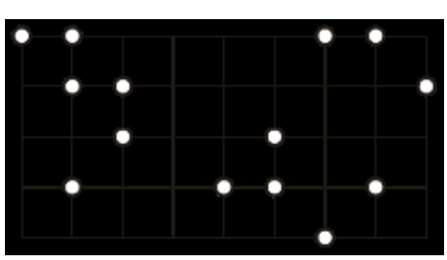
## I. Algorithms

I.1 **[5]** The array **`coins`** contains six entries which contain the values of the six European coins. Suppose change is to be given for a particular amount *n* in cents. Write a function which determines the minimum number of coins necessary to make change for any positive value of *n* cents. You may assume the argument *n* = 0. For example, the change required for *n* = 34 cents is 4 (20, 10, 2, 2). Hint: for while

```
int coins[7] = {1, 2, 5, 20, 50, 100, 200};

int number_of_coins( int n ) {
      int count = 0;




      return count;
}
```

I.2 **[4]** In a wireless network, the probability of a transmission error increases with the distance between the vertices. In constructing *ad hoc* networks, it is desirable to partition the vertices into groups where two vertices are in the same group if the distance between them is less than a fixed distance *d*. In this case, any vertex could communicate with any other vertex in the group by communicating through a sequence of vertices in the same group where successive transmission distances are always less than *d*. Suggest a greedy algorithm to find a partition of a set of vertices.

I.3 **[3]** Apply your algorithm you found in Question I.2 to partition the following 14 vertices where the distance between parallel grid lines is 1 and where two vertices should be in the same group if the distance between then is 2 or less.



I.4 **[5]** Given the divide-and-conquer run time of

$$\mathrm{T}(n) = \begin{cases} 1 & n = 1 \\ a\,\mathrm{T}(n/b) + n^k & n > 1 \end{cases},$$

where $a$, $b$, and $k$ are fixed, show how this can be transformed into the sum

$$\frac{\mathrm{T}(b^m)}{a^m} = 1 + \sum_{\ell=1}^{m} \left( \frac{b^k}{a} \right)^{\ell}$$

when we assume $n = b^m$. This can trivially be transformed into the desired form $\mathrm{T}(b^m) = a^m \sum_{\ell=0}^{m} \left( \frac{b^k}{a} \right)^{\ell}$.

I.5 **[1]** What are the three possible conditions (as a list of three equalities/inequalities) which will ultimately determine the run time of the divide-and-conquer algorithm?

## J Sparse Matrices

J.1 **[4]** Draw 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 | 9 | 9 |

**`JA`**

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

**`A`**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 22 | 11 | 2 | 1 | 3 | 3 | 4 | 33 | 44 |

Table J.1

| | | | | | |
|---|---|---|---|---|---|
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

## K Disjoint Sets

K.1 **[3]** Create a data structure which stores the disjoint sets

{0}, {1, 2, 3, 4}, {5, 6, 7}, {8, 9}

and put your answer in Table K.1.

Table K.1

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

There are multiple possible answers: you may choose the most straight-forward you wish.

K.2 **[2]** Given your representation in Question K.1, take the union of the sets {1, 2, 3, 4} and {5, 6, 7} and put your answer in Table K.2.

Table K.2

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