ECE 250 *Algorithms and Data Structures*
Sections 001 and 002
FINAL EXAMINATION

Douglas Wilhelm Harder dwharder@uwaterloo.ca EIT 4018 x37023

2015-4-15T16:00/18:30
Rooms: PAC 6, 7

IF YOU ARE NOT ENROLLED IN THIS COURSE AND ARE WRITING A SUPPLEMENTAL EXAMINATION TO CLEAR A PAST FAILURE, CHECK HERE: ☐

1. There are 66 marks.
2. No aides.
3. Turn off all electronic media and store them under your desk.
4. If there is insufficient room, use the back of the **previous** page.
5. You may ask only one question during the examination: "May I go to the washroom?"
6. Asking **any** other question **will** result in a deduction of 5 marks from the exam grade.
7. If you think a question is ambiguous, write down your assumptions and continue.
8. **Do not leave during first hour or after there are only 15 minutes left.**
9. Do not stand up until all exams have been picked up.
10. If a question asks for an answer, you do not have to show your work to get full marks; however, if your answer is wrong and no rough work is presented to show your steps, no part marks will be awarded.
11. You may need the sequence 256, 384, 576, 864, 1296, 1944, 2916, 4374, …
12. Remember that $\sum_{k=1}^{n} k^d \approx \frac{n^{d+1}}{d+1}$.

Attention:
The questions are in the order of the course material.

THIS BLOCK MUST BE CORRECTLY COMPLETED USING ALL CAPITAL LETTERS (1 mark)

Surname (last name) as appearing in Quest (e.g., HARDER)
Legal given name(s) as appearing in Quest (e.g., DOUGLAS WILHELM)
Common or nick name (optional, e.g., DOUG)

uWaterloo User ID									@uwaterloo.ca
Student ID Number	2	0							

I have read the above instructions:

Signature:

____

Asking any question other than that question noted above.

-5

A. Relations, asymptotic analysis and algorithm analysis

A.1 [1] What differentiates a weak ordering from a linear ordering?

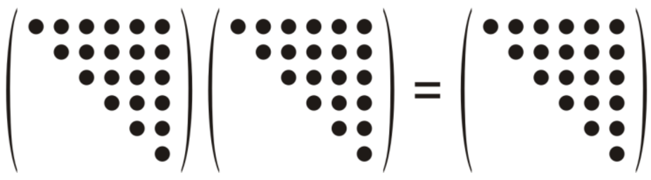
A.2 [2] What differentiates a hierarchical ordering from a partial ordering?

A.3 [3] The following loop is for a matrix-matrix multiplication of two $n \times n$ matrices. The run-time is $\Theta(n^3)$.

```
for ( int i = 0; i < n; ++i ) {
    for ( int k = 0; k < n; ++k ) {
        C[i][k] = 0.0;

        for ( int j = 0; j < n; ++j ) {
            C[i][k] += A[i][j] * B[j][k];
        }
    }
}
```

Assume that the matrices are known to be upper-triangular ($a_{ij} = b_{ij} = 0$ if $i > j$). Optimize the above algorithm to minimize the number of unnecessary multiplications and argue whether or not the new algorithm is $o(n^3)$. An image is provided for your convenience, and you may use functions in the C++ standard template library (you need not write the namespace `std::`).



B. Linear abstract data types, arrays and linked list

B.1 [4] Assume that the entries in a double-sentinel list are known to be linearly ordered (smallest first). Write an insert function that will place the object in the correct location in your sorted double sentinel list. If the argument already appears in the list, return `false`; otherwise, create a new double node at the appropriate location and return `true`. Use only the member variables `list_head` (`lh`) `list_tail` (`lt`), `list_size` (`ls`), `next_node` (`nn`), `previous_node` (`pn`), and `element` (`e`). Do not call any member functions of either class. You may use the symbol in parenthesis to refer to the listed member variables (e.g., `++ls` instead of `++list_size`). You may use `Dn` in place of the class name `Double_node`.

```
template<typename Type>
bool Double_sentinel_list<Type>::insert( Type const &obj ) {













}
```

B.2 [4] In class we saw that if you double the size of an array each time it is filled, then the amortized number of copies per insertion is Θ(1), as after inserting $n = 2^m$ objects, the number of copies is

$$\frac{\sum_{k=0}^{m-1} 2^k}{2^m} = \Theta(1) \ .$$

The number of calls to `new[]` is Θ(ln(n)).

If we increased the size of the array by a fixed amount, we saw that inserting n objects into the array results in an amortized Θ(n) copies per insertion with Θ(n) calls to `new[]`.

Suppose, instead, each time the array is full, we increase the size of the array to the next perfect cubes, so the sizes of the array are 1, 8, 27, 64, 125, 216, 343, etc. Answer the following two questions:

1. Given n insertions, what is the amortized number of copies per insertion?
2. Asymptotically speaking, what is the number of calls to `new[]` given n insertions?

C. Hierarchies and trees

C.1 [**3**] Prove by induction that a traversal of a general tree with n nodes requires $\Theta(n)$ time. Assume that function calls and accessing the i^{th} child are both $\Theta(1)$ operations.

D. Ordered trees (perfect, complete and balanced trees)

D.1 [1] The relationship between the number of nodes in a binary tree n and the number of leaf nodes ℓ, the number of single-child nodes s, and the number of full nodes f is

$$n = \ell + s + f.$$

Simplify this so the right-hand side has only two variables instead of three by writing one in terms of one of the other two.

E. Sorted abstract data types and search trees

E.1 [3] A colleague suggests relaxing the need to perform operations in an AVL tree by allowing the difference in height to be at most two, as opposed to being at most one. Draw worst-case relaxed AVL trees of heights 5 and 6. Demonstrate that your trees satisfy the recurrence relation $F(h) = 1 + F(h - 1) + F(h - 3)$ with appropriate values of $F(0)$, $F(1)$ and $F(2)$.

E.2 [2] The same colleague later suggests that, to reduce the work required to perform an erase from an AVL tree, instead of removing the node, tag the erased nodes with a Boolean-valued flag, and then if a new value is inserted into the tree that fits into that tagged node (that is, the new value is greater than anything in the left sub-tree and smaller than anything the right sub-tree), place the value in that node and mark the node as no longer being erased. You now have two additional choices:

1. call front and back on the left-and right sub-trees any time you reach an erased node, or
2. store the maximum element in the left sub-tree (if any) and the minimum element in the right sub-tree (if any) in the current node.

What is the impact on the run-time impact on `insert`(…) and the memory-requirement impact on the entire tree for each of these options.

E.3 [4] Insert the following 10 numbers, in the order given, into an initially empty B+-tree where *l* = *m* = 3.

45, 87, 70, 48, 23, 95, 32, 12, 19, 18, 1

F. Priority queues and binary heaps

F.1 [3] The same colleague as before came to you with the suggestion that you could merge two binary min-heaps stored as arrays by using the same merging algorithm used by merge sort (that is, by simply comparing the next entries in the list). Find a counter-example to show your colleague is wrong.

F.2 [3] Define the null-path length of a node in a tree and describe the properties of a leftist heap.

G. Sorting algorithms

G.1 [2] In order for a sorting method to remove O(n^2) inversions in o(n^2) time, it is necessary for a Θ(1) operation to remove ω(1) inversions. Explain when such operations take place in merge sort and justify your answer.

G.2 [4] Quick sort is generally faster than either merge sort or heap sort, and usually it requires only Θ(ln(n)) memory; however, on occasion, it may require ω(ln(n)) memory and have a ω(n ln(n)) run time. Suppose you implement the following algorithm:

1. If the array size is $n < 256$, use insertion sort;
2. otherwise, if the call stack already has 8 calls to quick sort on it, call heap sort;
3. otherwise, call quick sort recursively following partition.

Questions:

1. Why does this algorithm run in Θ(n ln(n)) time even in the worst-case scenario for quick sort?
2. Suppose quick-sort divides the list into a ratio of 1:2 at each step. How large would n have to be for heap sort to be called even once?

H. Hash tables

H.1 [4] Using quadratic probing (as shown in class) with the least-significant digit as the initial hash function, insert the following eight numbers into the empty hash table shown in Table 1.

34, 15, 65, 53, 73, 42, 40, 80

Table 1. A hash table using quadratic probing.

0	1	2	3	4	5	6	7
40	73	42	53	34	15	65	80

Erase the entries 34, 42, 65 and 40 from the hash table in Table 1. Following this, insert the values 54, 37 and 16 and put your solution in Table 2.

Table 2. Your solution.

0	1	2	3	4	5	6	7	8	9

I. Graph algorithms

I.1 [2] Prove that a directed acyclic graph has at least one vertex of in-degree zero.

I.2 [4] Apply Kruskal's algorithm to the graph in Figure 1. Indicate the steps you are taking.

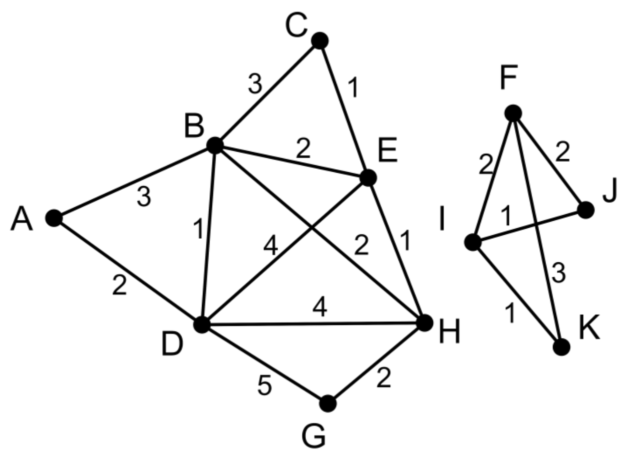


Figure 1. A weighted graph.

I.3 [2] Which approach is better for Kruskal's algorithm?

1. Sort the array of edges by the weights, and then step through the array until a minimum spanning tree is found.
2. Convert the array of edges into a min-heap and then pop the smallest edge until a minimum spanning tree is found.

You must use asymptotic analysis in your answer. You may assume that the number of edges is $\Theta(|V|^2)$ and you may assume an array of edges is already available.

J. Algorithm design

J.1 [4] Consider the following recursive traversal on the vertices of a directed graph (the vertices are numbered `0` to `num_vertices - 1`) which tries to find if a vertex is connected to another vertex.

```
bool is_connected( int starting_vertex, int final_vertex ) {
    int count = 0;

    for ( int i = 0; i < num_vertices; ++i ) {
        // 'starting_vertex' is connected to 'final_vertex' if
        //   1. 'starting_vertex' is adjacent to 'i' and
        //   2. 'i' is connected to 'final_vertex'

        if ( i != starting_vertex && adjacent( starting_vertex, i ) ) {
            if ( is_connected( i, final_vertex ) ) {
                return true;
            }
        }
    }

    return false;
}
```

Answer the following questions:

1. What are the issues if this algorithm is used on a directed acyclic graph?
2. What are the issues if this algorithm is used in a general directed graph?
3. How would you correct these issues, and what algorithm design technique are you using?

J.2 (4) You have just come up with an integer multiplication algorithm that satisfies

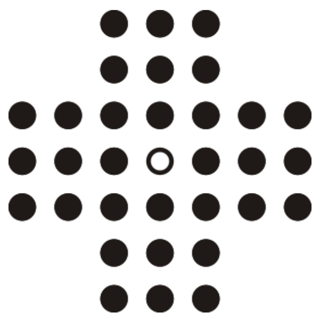
$$T(n) = \begin{cases} \Theta(1) & n = 1 \\ 4T\left(\frac{n}{3}\right) + \Theta(n) & n > 1 \end{cases}$$

Show how you can take the expression

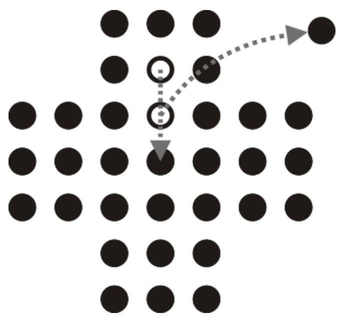
$$T(n) = T(b^m) = a^m \sum_{\ell=0}^{m} \left(\frac{b^k}{a}\right)^{\ell}$$

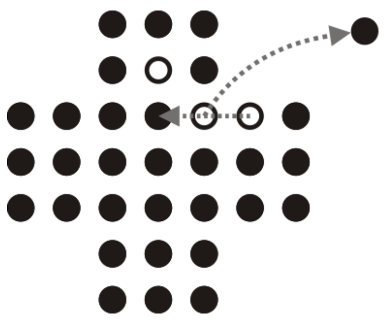
to deduce the run time of your algorithm. Is your algorithm faster or slower than the Karatsuba algorithm we saw in class, which runs in $\Theta(n^{\lg(3)})$?

J.5 [4] The game of Peg Solitaire is played as follows. A board is set up with 32 pegs arranged in 33 holes, as follows, where the center hole is left blank.

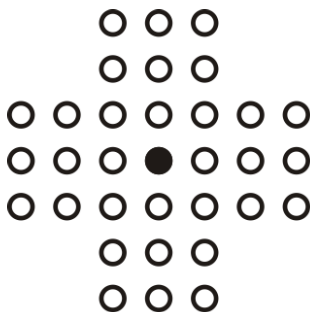


The only legal moves are for a peg to jump either horizontally or vertically over another peg into a blank hole, and the peg that is jumped is removed from the board leaving a new blank. This continues until no jumps are possible. For example, the first two moves of the game could be as follows:





The goal is to finish with a board where only one peg is left in the center hole.



Describe a backtracking algorithm to solve this problem. Your description of the algorithm must be in sufficient detail so that a 1st-year student who has completed ECE 150 could implement it. That student, however, is not aware of any of the new material taught in ECE 250. Be sure to include in your description the data structure that will be passed between recursive calls and the changes that must be made to it.

K. Theory of computation

K.1 [2] A Turing machine consists of a tape, a head that can read from and write to the tape, a state, and a transition table that maps symbols on the tape and the state to a new symbol on the tape, a new state and movement of the tape under the head. How do these components relate to the components of a modern computer?

L Bonus Questions (If you attempt both, cross out the one we should not mark, otherwise, we will mark L.1.)

L.1*a* [3] Describe an implementation of a disjoint set by first describing the underlying data structure (member variables) and how they are initialized. Second, what are the two primary member functions, and how do they manipulate or use the underlying data structure.

L.2*a* [1] Is a splay tree always going to be balanced, in the sense that the height is always $\Theta(\ln(n))$? (Yes or No)

L.2*b* [2] Demonstrate how the splay tree in Figure 2 would change if a search is made for the number 13.

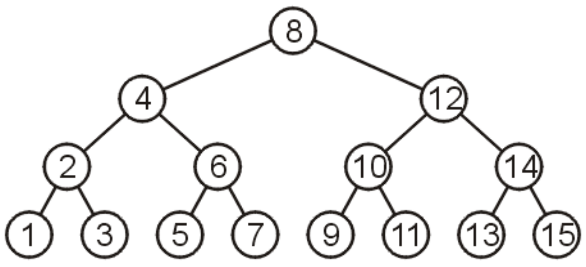


Figure 2. A splay tree.