MA/CSSE 473 Summer 2014 Final exam specification (Last updated August 11, 2014)

Resources allowed:

1. Required: Working Windows or Mac laptop with webcam and microphone:
2. Calculator (not a smart phone, iPod, etc. that happens to have a calculator.
    a. The only necessary operations are +, *, -, /, exponentiation.
3. Two pieces of blank paper for use as scratch paper.
4. (optional one 8.5" x 11" sheet of paper with handwritten notes on one side.

ProctorU: http://www.rose-hulman.edu/class/csse/csse473/201440/Resources/ProctorU-and-summer-2014-classes.pdf . This service is required, and it is free if you sign up for an exam time at least 72 hours before that time, and if you don't miss your appointment. You must begin between midnight and 9 PM Indiana time on Wednesday, August 13.

Material Covered

- **Background material from 230:** Algorithms and analysis for implementing stacks, queues, linked lists, sequential lists, binary trees, binary search trees, binary heaps, dictionaries. Recursion and mathematical induction. Writing and solving simple recurrence relations, analysis of nested loops as in Weiss chapter 5 and its exercises. Fibonacci numbers. Sequential and binary search. Well-known sorting methods: Insertion, selection, merge, quick, heap. Binary tree traversals: preorder, inorder, postorder, level order. Formal definitions of O(N), Θ(N), etc.
- **HW 0 - HW 16** (including the "not to turn in" problems)
- **Assigned readings** from Chapters 1-11 from the textbook plus Sections 12.1, 12.2, as well as documents posted on Moodle that are listed on the Schedule page, and material from PowerPoint slides for all sessions. (Chapters 1-5 from the textbook as well as documents posted on Moodle that are listed on the Schedule page for Sessions 1-16), and material from PowerPoint slides for sessions 1-18.
  More details below on specific algorithms you should know.

Question types: In order to minimize the amount of typing you must do, the exam will have several multiple-choice and T-F-IDK questions. For questions where you have to write something, You do not have to have perfectly formatted mathematics. English and "mathematical pseudocode" is fine.
For example, Theta(3N^2 + (N^2 + N)/sqrt(N-4) or (1/N)*Sum(k-1, k=2 .. N).

T-F-IDK. Below you will find several statements. A statement is true (T) if it is always true. It is false (F) if there is at least one counterexample (sometimes false). You may also choose IDK to indicate that you do not know the answer. *There is some value (to me and you) in knowing what you don't know.*

Point values: Correct answer: 4, incorrect answer: -1, IDK: 1, blank: 0. So if you get seven questions correct, choose IDK for three, get two wrong, and leave one blank, your score is 7*4 +3*2 – 2*1 + 1*0 = 32 out of 52. What if you are not sure of an answer? If you are totally guessing an answer, you are probably better off pointwise if you choose IDK. If you have a feeling that one "real" answer is correct, you are probably better off guessing that answer.

Can you come up with a reasonably efficient algorithm to … ? Many of the homework problems (especially the puzzle problems) have been this type of problem.

Does this proposed algorithm for problem _____ work?

What is the worst-case running time for the following algorithm?

Which of these algorithms is an example of <design technique>? (design_technique may be something like.Greedy, Divide-and-conquer, Dynamic Programming, Decrease by a constant factor. The problem will ask you to choose among a list of several algorithms. The problem could also go the other way; giving you the name of an algorithm that we studied and asking you to choose among a list of algorithm design techniques.

Do you know the details of this algorithm? Some questions will be checking to see whether you know and understand the details of specific algorithms. Here are examples of algorithms that you should be able to explain and discuss the results of what happens when they are implemented with specific input data.

Examples (there are more examples in the background material section above)**: (the first 6 bullets are form Dasgupta)**

- Addition, multiplication, exponentiation of integers (bit by bit, digit by digit), modular arithmetic.
- Euclid's algorithm for finding GCD
- Extended Euclid for finding modular inverse, and the multiples that lead to GCD.
- Fermat's little theorem. Uses and limitations when doing primality testing.
- Carmichael numbers, probabilistic primality testing.
- RSA public-key cryptography
- Sieve of Eratosthenes (L 1.1)
- Towers of Hanoi (L 2.4)
- Selection Sort, Bubble Sort, and their analysis (L 3.1)
- Sequential Search (L 3.2)
- Brute force String Match (L 3.2)
- Brute Force Convex Hull (L 3.3)
- Brute Force Closest Pair (L 3.3)
- DFS and BFS in a graph (L 3.5)
- Fast exponentiation (L 4.1)
- Insertion Sort (L 4.1)
- Topological Sort (L 4.2)
- Johnson-Trotter algorithm for generating permutations (L 4.3)
- Lexicographic order permutation generation (L 4.3), from permutation to number and back (my slides)
- Subset generation and binary reflected Gray code (L 4.3)
- Binary Search (L 4.4)
- Fake Coin Problem (L 4.4)
- Russian Peasant Multiplication (L 4.4)
- Josephus Problem (L 4.4)
- QuickSelect (L 4.5)
- Interpolation Search (L 4.5)
- Binary tree search and insertion (L 4.5)
- Nim (binary digital sum)
- Merge sort (L 5.1)
- Quick sort (L 5.2) Know the basic approach to solving average case recurrence)
- Big integer multiplication (L 5.4)
- Fast Matrix Multiplication (L 5.4)
- Recursive closest pair (L 5.4)
- Recursive convex hull (L 5.5)
- Gaussian Elimination (L 6.2)
- LU Decomposition (L 6.2)
- Matrix Inverse and Determinant computation (L 6.2)
- Insertion an balancing: AVL trees and 2-3 trees (L 6.3)

- Heap insertion and maximum element deletion (L 6.4)
- Heapsort (L 6.4)
- Horner's Rule for left-to-right polynomial evaluation (L 6.5)
- Left-to-right binary exponentiation (L 6.5)
- Sorting by Counting (L 7.1)
- Horspool and Boyer-More algorithms (L 7.2)
- B-tree insertion and size calculation (L 7.4)
- Optimal static Binary search tree, including "gaps" (as in the HW problem, the slides, and the Reingold/Hanson excerpt that is posted on Moodle) (also L 8.3)
- Warshall's Algorithm for finding the transitive closure of a graph (L 8.4)
- Floyd's Algorithm for All-pairs shortest Path problem (L 8.4)
- Prim's Algorithm for finding a MST (L 9.1)
- Kruskal Algorithm for finding a MST (L 9.2)
- Disjoint sets and union-find problem (L 9.2)
- Dijkstra's Algorithm for single-source shortest Path problem (L 9.3)
- Huffman trees and codes (L 9.4)
- For-Fulkerson algorithm for maximum flow (L 10.2)
- Stable Marriage problem (L 10.4)
- Backtracking as applied to Queens, Hamiltonian Circuit problem, Subset-sum problem

Concepts, definitions, etc.

- Graph representations (adj matrix, adj lists) (L 1.4)
- Graph definitions: edge, directed, head tail, digraph, loop, complete, dense, sparse, path, simple path, cycle, connected component (L 1.4)
- Efficiency: best case, worst, average, amortized (L 2.1)
- Asymptotic notation (O, Ω, Θ), including formal definition of big-O, limits and asymptotics (L 2.2)
- Nested loops and summation (L 2.3, W Chapter 5)
- Traveling Salesmen problem, Knapsack Problem, Assignment problem (L 3.4)
- Master Theorem for divide-and-conquer algorithms. No need to memorize; I will give you the formulas (L 5.1)
- Binary tree traversals and properties (L 5.3)
- Transform and conquer – Presorting (L 6.1)
- Reducing the solution of a problem in one domain to solving a problem in (possibly) another domain (L 6.6)
- Hashing: the basic ideas, separate chaining vs Open Addressing, Approaches to collision resolution (this is mostly review from CSSE 230) (L 7.3 and there is additional information in the PowerPoint slides)
- Basics of the Dynamic Programming approach (L 8.1)
- What is a greedy algorithm (L 9.intro)
- (We did not do anything with sections 10.1 or 10.3, so you can ignore them)
- Lower bound arguments for problem solution efficiency (L 11.1)
- Decision trees (review of 230) (L 11.2)
- Know the definitions of P, NP, and NPC (L 11.3)
- (We did not do anything with section 11.4, so you can ignore it)
- Backtracking (L 12.1)
- Branch and Bound (L 12.2)
- (We did not do anything with sections 12.3 or 12.4, so you can ignore them)