Final Exam Study Guide

Weight per section will be even -- the post test 2 material represents about 2.5-3 weeks of material when factoring in the stuff from Test 2, and considering 14 week semester but the earlier stuff has already been tested once so, doubling, the post-test 2 material is about ⅓.

You may have a half page, **double-sided**, of notes, or one page, one side.

Clarification, though I thought it was pretty clear in the first message: You can substitute up to one section for a test (not two). I will set up the spreadsheet such that the best choice will be the one I use when calculating the grade. This cannot hurt your grade.

“One of” here means I may ask either question type but not both

Test 1

One of

* Use a limit, perhaps L'Hopital's, to compare the order of growth
* Be able to order functions in increasing order of growth

There will be one question asking you to use the definition of big O/Theta/Omega

For sums and recurrences I will focus on forms that you have seen a lot of (ones that come up when doing algorithms), ones similar to Ch2 slide 56, master method… for recurrences

One recurrence- will say to do it with **backward substitution** (but if you can do it another way, do that as well to check your answer)

One sum

Be able to define a Brute Force algorithm to solve a problem

Test 2

DFS topological sort

* Show all marks
* Label all edges
* If topological sort isn't possible, then terminate early
* Make sure you actually give the topological sort, should it exist.

Gray code:

* Generate or
* Be able to tell if a given example possibly valid and explain why (just make sure valid minimal change subset algorithm.)

Be able to do Merge sort. I might ask a secondary question related to :

* Stability in merge sort
* Worst case performance of merge sort

One of:

* AVL tree
* 2-3 tree

Be able to:

* Max-heapify (make a heap) an array in a bottom up manner or
* Delete some small number of nodes from a heap (delete the max repeatedly) and show the final array

NO Horspool’s

Hash tables:

One of:

* Be able to insert into either an open hash table or a closed one with linear probing (know what “open”, “closed”, and “linear probing” mean when talking about hash tables)
* Be able to answer a general question about hash tables e.g.:
  + Performance as load factor increases
  + What to do if allocating additional memory, such as in the linked lists for open hash tables is undesirable

Post Test 2:

Be able to do Master theorem examples. A reasonable 10 point question might be to do two of the following three recurrences “any way you like”, two will be possible with the master method, with minor manipulation, possibly, e.g.:

T(n) = 3 T(n/3) + n2 T(n) = 2T(n-1) + 1 T(n) = T(n/2) + T(n/2) + n

* It has to fit the Master Theorem pattern, which is common for many divide and conquer algorithms (and decrease by a constant *facto*r) algs. i.e.:

T(n) = a T(n/b) + nd

* Just get the values of a,b, and d, compare a to bd and see what the answer is (plug and chug)
* For the above example:
  + The left is obviously valid for the master method/theorem
  + The middle is not (subtraction instead of division)
  + The right one is -- just sum the two recursive parts to get into the proper form, i.e.

T(n/2) + T(n/2) + n = 2T(n/2) + n

* *For our purposes, we will just say that if f(n) ( n2 , 1, and n in each of the above) is not a polynomial of n, then it is not acceptable for the master method ( remember n0 = 1)*

*Boyer-Moore (will do both)*

* Be able to build good suffix table
* Given the tables, perform a string match

One of:

* Warshall’s (transitive closure)
* Floyd's (all pairs shortest paths)

(For both of these, know what you're​ doing - I might just ask you to give the transitive closure or all pairs shortest paths. One truck might be of I give you the regular adjacency matrix and ask you to do the transitive closure)

One of:

* Prim’s -for MST
* Dijkstra’s- for **single** source shortest path

(Again, know what you're doing-I might not just tell you what algorithm by name)

Ch. 11/Ch.12 Material:

* “Big ideas”:
  + Lower Bound
  + Tight Lower Bound: if we have an algorithm of the same order as a lower bound that lower bound is “tight”.
  + <You don’t need to build or interpret a decision tree>
  + Optimization problem: construct a solution that maximizes or minimizes some objective function
  + Decision problem: answer yes/no to a question
  + P -- the class of decision problems solvable in polynomial time
  + NP -- the class of decision problems solvable in **N**ondeterministic **P**olynomial-time -- i.e. with a hypothetical nondeterministic algorithm -- the solutions can be *verified* in polynomial time
  + Nondeterministic Algorithms have 2 stages:
    - Guess a random solution
    - Verify with a **deterministic** algorithm whether the solution is valid. The algorithm is in P (meaing it is polynomial).
  + NPC (NP complete): This is the set of problems that are in NP *and* that every problem in NP can be reduced to in polynomial time (meaning we can convert any problem in NP to a version that our NPC algorithm could solve, given some way to )
  + <I will do a few examples -- they’re kind of neat -- but actually doing this is well beyond the scope of the course>
  + Backtracking:
    - Builds a state-space tree
    - Can solve combinatorial (and other..) problems but doesn’t guarantee polynomial time
    - Valid for Optimization and Decision problems
  + Branch and Bound:
    - Only for optimization
    - Uses knowledge gained from:
      * Best possible lower/upper bound (for minimization/maximization problems)
      * The best solution seen so far
    - A good thing to try for difficult optimization algorithms with no alternative other than backtracking
    - No guarantee of polynomial time (special case of backtracking).