Final Examination

- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- When the quiz begins, write your name on every page of this quiz booklet.
- You have 180 minutes to earn 200 points. Do not spend too much time on any one problem. Read them all through first, and attack them in the order that allows you to make the most progress.
- This quiz booklet contains 15 pages, including this one. Two extra sheets of scratch paper are attached. Please detach them before turning in your quiz at the end of the exam period.
- This quiz is closed book. You may use three $8\frac{1}{2}'' \times 11''$ or A4 crib sheets (both sides). No calculators or programmable devices are permitted. No cell phones or other communications devices are permitted.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Do not put part of the answer to one problem on the back of the sheet for another problem, since the pages may be separated for grading.
- Do not waste time and paper rederiving facts that we have studied. It is sufficient to cite known results.
- Show your work, as partial credit will be given. You will be graded not only on the correctness of your answer, but also on the clarity with which you express it. Be neat.
- Good luck!

Problem	Parts	Points	Grade	Grader		Problem	Parts	Points	Grade	Grader
1	6	18				6	1	25		
2	6	18				7	2	30		
3	4	24				8	3	30		
4	1	15				9	1	20		
5	1	20								
						Total		200		

Name: _____

Problem 1. Miscellaneous True/False [18 points] (6 parts)

For each of the following questions, circle either T (True) or F (False). **Explain your choice.** (No credit if no explanation given.)

(a) T F If the load factor of a hash table is less than 1, then there are no collisions.
Explain:

(b) T F If $SAT \leq_P A$, then A is NP-hard.
Explain:

(c) T F The longest common subsequence problem can be solved using an algorithm for finding the longest path in a weighted DAG.
Explain:

(d) T F Applying a Givens rotation to a matrix changes at most one row of the matrix.
Explain:

(e) T F The problem of finding the shortest path from s to t in a directed, weighted graph exhibits optimal substructure.
Explain:

(f) T F A single rotation is sufficient to restore the AVL invariant after an insertion into an AVL tree.
Explain:

Problem 2. More True/False [18 points] (6 parts)

For each of the following questions, circle either T (True) or F (False). **Explain your choice.** (No credit if no explanation given.)

(a) T F Using hashing, we can create a sorting algorithm similar to COUNTING-SORT that sorts a set of n (unrestricted) integers in linear time. The algorithm works the same as COUNTING-SORT, except it uses the hash of each integer as the index into the counting sort table.
Explain:

(b) T F There exists a comparison-based algorithm to construct a BST from an unordered list of n elements in $O(n)$ time.
Explain:

(c) T F It is possible for a DFS on a directed graph with a positive number of edges to produce no tree edges.
Explain:

(d) T F A max-heap can support both the INCREASE-KEY and DECREASE-KEY operations in $\Theta(\lg n)$ time.
Explain:

(e) T F In a top-down approach to dynamic programming, the larger subproblems are solved before the smaller ones.
Explain:

(f) T F Running a DFS on an undirected graph $G = (V, E)$ always produces the same number of cross edges, no matter what order the vertex list V is in and no matter what order the adjacency lists for each vertex are in.
Explain:

Problem 3. Miscellaneous Short Answer [24 points] (4 parts)

You should be able to answer each question in no more than a few sentences.

(a) Two binary search trees t_1 and t_2 are *equivalent* if they contain exactly the same elements. That is, for all $x \in t_1$, $x \in t_2$, and for all $y \in t_2$, $y \in t_1$. Devise an efficient algorithm to determine if BSTs t_1 and t_2 are equivalent. What is the running time of your algorithm, assuming $|t_1| = |t_2| = n$?

(b) You are given a very long n-letter string, S, containing transcripts of phone calls obtained from a wiretap of a prominent politician. Describe, at a high level, an efficient algorithm for finding the 4-letter substring that appears most frequently within S.

(c) A *word ladder* is a sequence of words such that each word is an English word, and each pair of consecutive words differs by replacing exactly one letter with another. Here is a simple example:

TREE, FREE, FRET, FRAT, FEAT, HEAT, HEAP

Assume that you are supplied with a function GET-WORDS(w) that returns, in $\Theta(1)$ time, a list of all English words that differ from w by exactly one letter (perhaps using a preprocessed lookup table). Describe an efficient algorithm that finds a word ladder, beginning at w_1 and ending at w_2, of minimum length.

(d) What procedure would you use to find a longest path from a given vertex s to a given vertex t in a weighted directed acyclic graph, when negative edge weights are present?

Problem 4. Changing Colors [15 points]

Consider a directed graph G where each edge $(u, v) \in E$ has both a weight $w(u, v)$ (not necessarily positive) as well as a color $color(u, v) \in \{\text{red}, \text{blue}\}$.

The weight of a path $p = v_1 \to v_2 \to \ldots \to v_k$ is equal to the sum of the weights of the edges, *plus* 5 for each pair of adjacent edges that are not the same color. That is, when traversing a path, it costs an *additional* 5 units to switch from one edge to another of a different color.

Give an efficient algorithm to find a lowest-cost path between two vertices s and t, and analyze its running time. (You may assume that there exists such a path.) For full credit, your algorithm should have a running time of $O(VE)$, but partial credit will be awarded for slower solutions.

Problem 5. DAG Paths [20 points]

Consider two vertices, s and t, in some directed acyclic graph $G = (V, E)$. Give an efficient algorithm to determine whether the number of paths in G from s to t is odd or even. Analyze its running time in terms of $|V|$ and $|E|$.

Problem 6. Square Depth [25 points]

Imagine starting with the given number n, and repeatedly chopping off a digit from one end or the other (your choice), until only one digit is left. The *square-depth* $S(n)$ of n is defined to be the maximum number of square numbers you can arrange to see along the way. For example, $S(32492) = 3$ via the sequence

$$32492 \rightarrow \mathbf{3249} \rightarrow \mathbf{324} \rightarrow 24 \rightarrow \mathbf{4}$$

since 3249, 324, and 4 are squares, and there is no better sequence of chops giving a larger score (although you can do as well other ways, since 49 and 9 are both squares).

Describe an efficient algorithm to compute the *square-depth*, $S(n)$, of a given number n, written as a d-digit decimal number $a_1 a_2 \ldots a_d$. Analyze your algorithm's running time.

Your algorithm should run in time polynomial in d. You may assume a function IS-SQUARE(x) that returns, in constant time, 1 if x is square, and 0 if not.

Problem 7. Least-Squares Minimization [30 points] (2 parts)

(a) Some least-squares minimizations can be solved by substitution even though the coefficient matrix is not upper triangular. For example, the following problem can be solved by substitution:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -7 & -2 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 8 \\ 4 \end{pmatrix} .$$

One way to solve such problems is to find an ordering of the rows and an ordering of the columns: find x_1 from the third constraint, then substitute x_1 and find x_2 from the second constraint, and so on.

Describe an efficient algorithm to find suitable orderings for the unknowns and the constraints. For simplicity, the algorithm is allowed to assume that there is such a reordering.

The input matrix is given to the algorithm as a Python list of m rows, where each row is a list of tuples; each tuple contains a column index (0 to $n - 1$) and a nonzero value. Zero matrix entries are not represented at all. The output should be a list of n (unknown index, constraint index) pairs, giving the substitution ordering. The algorithm should run in time $\Theta(n + m + k)$, where k the number of nonzeros in the entire matrix (which is also the number of tuples in the input).

(b) For many problems, substitution alone is not sufficient; Givens rotations are necessary.

One interesting challenge in performing a sequence of rotations efficiently is to compute unions of pairs of integer sets.

Describe an algorithm that repeatedly takes two sets of integers represented as lists of integers and and generates a list of the union of the two sets. Also describe any data structures that your algorithm uses.

The integers are all between 0 and $n-1$, there are no duplicates in the input lists, and there should be no duplicates in the output list. The pairs of sets are unrelated.

The algorithm can use $O(n)$ preprocessing time before computing the first union. After the preprocessing phase, computing each union should take $O(k+k')$ *worst-case* time, where k and k' are the sizes of the two input lists.

Problem 8. Sorting on Disks [30 points] (3 parts)

You need to sort a large array of n elements that is stored on disk (in a file). Files support only two operations:

1. READ(f) returns the next number stored in file f.
2. WRITE(f, p) writes the number p to the end of file f.

These operations are slow, so we wish to sort the given array into another file while incurring as few READs and WRITEs as possible. Your algorithm may create new files if you desire.

If we had unlimited RAM, we could solve this problem by simply reading the entire array into memory, sorting it, then writing it to disk. Unfortunately, the computer can only store a limited number of numbers in memory, so this simple approach is infeasible.

(a) If our computer can store at most $\frac{n}{2}$ numbers in memory, how would you sort the file? Assume the computer has enough working memory to store any auxiliary information your algorithm needs. How many READs and WRITEs does your algorithm incur?

(b) How would you sort the file if the computer can store at most $\frac{n}{4}$ numbers in memory? Again assume adequate working memory. How many READs and WRITEs does your algorithm incur?

(c) Now imagine the computer uses some technology (e.g., flash memory) where the READ operation is fast, but the WRITE operation is expensive. Modify your algorithm from part (b) to minimize the number of WRITEs. The computer can still store at most $\frac{n}{4}$ numbers in memory.

Problem 9. Star Power! [20 points]

Consider a directed, weighted graph G where all edge weights are positive. You have one Star, which (along with making you temporarily invincible) lets you traverse the edge of your choice for free. In other words, you may change the weight of any one edge to zero.

Give an efficient algorithm to find a lowest-cost path between two vertices s and t, given that you may set one edge weight to zero. Analyze your algorithm's running time. For full credit, your algorithm should have a running time of $O(E + V \lg V)$, but partial credit will be awarded for slower solutions.

SCRATCH PAPER

SCRATCH PAPER