Final Exam

- 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 12 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. 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	8	24				5	1	24		
2	4	32				6	1	24		
3	3	24				7	1	24		
4	3	24				8	1	24		
						Total		200		

Name: _____

Problem 1. True or False [24 points] (8 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 There exists an algorithm to build a binary search tree from an unsorted list in $O(n)$ time.
Explain:

(b) T F There exists an algorithm to build a binary heap from an unsorted list in $O(n)$ time.
Explain:

(c) T F To solve the SSSP problem for a graph with no negative-weight edges, it is necessary that some edge be relaxed at least twice.
Explain:

(d) T F On a connected, directed graph with only positive edge weights, Bellman-Ford runs asymptotically as fast as Dijkstra.
Explain:

(e) T F A Givens rotation requires $O(1)$ time.
Explain:

(f) T F In the worst case, merge sort runs in $O(n^2)$ time.
Explain:

(g) T F There exists a stable implementation of merge sort.
Explain:

(h) T F An AVL tree T contains n integers, all distinct. For a given integer k, there exists a $\Theta(\lg n)$ algorithm to find the element x in T such that $|k - x|$ is minimized.
Explain:

Problem 2. Short Answer [32 points] (4 parts)

(a) The *eccentricity* $\epsilon(u)$ of a vertex u in a connected, undirected, *unweighted* graph G is the maximum distance from u to any other vertex in the graph. That is, if $\delta(u, v)$ is the shortest path from u to v, then $\epsilon(u) = \max_{v \in V} \delta(u, v)$.

Give an efficient algorithm to find the eccentricity of a given vertex s. Analyze its running time.

(b) What is the asymptotic cost of solving a linear system of equations with $n-1$ equations of the form

$$a_{i,i}x_i + a_{i,i+1}x_{i+1} = b_i \quad i = 1, \ldots, n-1$$

and one equation of the form

$$a_{n,1}x_1 + a_{n,2}x_2 + \cdots + a_{n,n}x_n = b_n$$

(none of the a's in this equation are zero). The a's and b's are given numbers and the x's are unknowns. Assume that we use Givens rotations to reduce the coefficient matrix to a triangular form. Justify your answer.

(c) Suppose a hash function h maps arbitrary keys to values between 0 and $m-1$, inclusive. We hash n keys, $k_1, k_2, \ldots, k_n$. If $h(k_i) = h(k_j)$, we say that k_i and k_j collide. Assuming simple uniform hashing, how should we choose m (in terms of n) such that the expected number of collisions is $O(1)$? Justify your answer.

(d) Suppose you have a directed acyclic graph with n vertices and $O(n)$ edges, all having nonnegative weights. Propose an efficient method of finding the shortest path to each vertex from a single source, and give its running time in terms of n.

Problem 3. Judge Jill [24 points] (3 parts)

Judge Jill has created a web site that allows people to file complaints about one another. Each complaint contains exactly two names: that of the person who filed it and that of the person he/she is complaining about.

Jill had hoped to resolve each complaint personally, but the site has received so many complaints that she has realized she wants an automated approach.

She decides to try to label each person as either good or evil. She only needs the labeling to be consistent, not necessarily correct. A labeling is consistent if every complaint labels one person as good and the other person as evil, and no person gets labeled both as good and evil in different complaints.

(a) [8 points] Propose a way to model the consistent labeling problem as a graph problem.

(b) [10 points] Propose an efficient algorithm to consistently label all the names as good or evil, or to decide that no such classification exists. Use the graph model you proposed in the previous part of the problem. Analyze the running time of the algorithm.

(c) [6 points] Later, Judge Jill wants to be more thorough. She will interview some people to figure out who is good and who is evil. She can always determine whether a person is good or evil by interviewing him or her. Assuming that one person in every complaint is good and the other is evil, what is the minimum number of people she needs to interview to correctly classify all the people named in the complaints?

Problem 4. Bitdiddle Bins [24 points] (3 parts)

Ben Bitdiddle has devised a new data structure called a Bitdiddle Bin. Much like an array or a set, you can INSERT values into it, and you can LOOKUP values to see if they are contained in the structure. (He'll figure out DELETE later.)

A Bitdiddle Bin is implemented as a pair of lists (arrays), designated the *neat list* and the *messy list*, with these properties:

- The *neat list* is always in sorted order. (The messy list may or may not be sorted.)
- The *messy list* has a size of at most $\sqrt{n}$, where n is the total number of values in the entire Bitdiddle Bin.

The LOOKUP algorithm for a Bitdiddle Bin is as follows:

1. Use binary search to look for the value in the neat list.
2. If it wasn't in the neat list, iterate over the entire messy list to look for the value.

The INSERT algorithm is as follows:

1. Append the value to the messy list.
2. If the messy list is now too big, CLEANUP.

This CLEANUP subroutine is run whenever the messy list grows beyond $\sqrt{n}$ items:

1. Sort the messy list.
2. Merge the messy list with the neat list.
3. The merge result is the new neat list. The new messy list is empty.

(a) [4 points] What is the worst-case asymptotic runtime of LOOKUP on a Bitdiddle Bin?

(b) [8 points] What is the worst-case asymptotic runtime of INSERT on a Bitdiddle Bin? Explain.

(c) [12 points] What is the *amortized* asymptotic runtime of each INSERT operation, when inserting n values into an empty Bitdiddle Bin? Explain.

Problem 5. Local Minimum [24 points]

Consider an array A containing n distinct integers. We define a *local minimum* of A to be an x such that $x = A[i]$, for some $0 \leq i < n$, with $A[i-1] > A[i]$ and $A[i] < A[i+1]$. In other words, a local minimum x is less than its neighbors in A (for boundary elements, there is only one neighbor). Note that A might have multiple local minima.

As an example, suppose $A = [10, 6, 4, 3, 12, 19, 18]$. Then A has two local minima: 3 and 18.

Of course, the absolute minimum of A is always a local minimum, but it requires $\Omega(n)$ time to compute.

Propose an efficient algorithm to find some local minimum of A, and analyze the running time of your algorithm.

Problem 6. Straits [24 points]

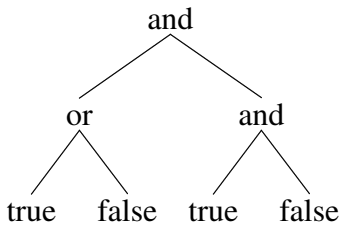
Let G be a connected, weighted, undirected graph. All the edge weights are positive. An edge e is a *strait* between vertices u and v if it has weight ℓ, is on a path from u to v whose other edges weigh ℓ or more, and every path between u and v contains an edge of weight ℓ or less. In other words, to go from u to v you must cross an edge of weight ℓ, but you do not need to cross edges lighter than ℓ.

Describe an efficient algorithm for finding the weight of the strait between every pair of vertices in G. Analyze the running time of the algorithm.

Hint: Use a $|V|$-by-$|V|$ table to keep track of the weights of all the straits. Initialize it so that it is correct for a graph with no edges. Then add the edges one by one.

Problem 7. The Cake Is a Lie! [24 points]

At Aperture Bakeries, every cake comes with a binary boolean-valued tree indicating whether or not it is available. Each leaf in the tree has either a *true* or a *false* value. Each of the remaining nodes has exactly two children and is labeled either *and* or *or*; the value is the result of recursively applying the operator to the values of the children. One example is the following tree:



If the root of a tree evaluates to *false*, like the one above, the cake is a lie and you cannot have it. Any *true* cake is free for the taking. You may modify a tree to make it *true*; the only thing you can do to change a tree is to turn a *false* leaf into a *true* leaf, or vice versa. This costs \$1 for each leaf you change. You can't alter the operators or the structure of the tree.

Cake is good. Cheap cake is even better. Describe an efficient algorithm to determine the minimum cost of a cake whose tree has n nodes, and analyze its running time.

Problem 8. I Am Locutus of Borg, You Will Respond To My Questions [24 points]

Upon arrival at the planet Vertex T, you and Ensign Treaps are captured by the Borg. They promptly throw the ensign out of the airlock. You had better solve their problem, lest you share his fate.

The Vertex T parking lot is an $n \times n$ matrix A. There are already a number of spaceships parked in the lot. For $0 \leq i, j < n$, let $A[i][j] = 0$ if there is a ship occupying position (i, j), and 1 otherwise.

The Borg want to find the ***largest square parking space*** in which to park the Borg Cube. That is, find the largest k such that there exists a $k \times k$ square in A containing all ones and no zeros. In the example figure, the solution is 3, as illustrated by the 3×3 highlighted box.

	0	1	2	3	4
0	0	1	1	1	0
1	1	1	1	1	1
2	1	1	1	1	1
3	1	1	0	0	0
4	1	1	1	0	1

Describe an efficient algorithm that finds the size of the largest square parking space in A. Analyze the running time of your algorithm.

Hint: Call $A[0][0]$ be the *top-left* of the parking lot, and call $A[n-1][n-1]$ the *bottom-right*. Use dynamic programming, with the subproblem $S[i, j]$ being the side length of the largest square parking space whose bottom-right corner is at (i, j).

SCRATCH PAPER

SCRATCH PAPER