1 Format
You will have 83 minutes to complete the exam. The exam will have true/false questions, multiple choice, example/counterexample problems, run-this-algorithm problems, and problem set style present-and-prove problems.

Whenever you are asked to give an algorithm for a problem, you are expected to analyze its correctness and running time. More efficient correct solutions receive more credit. Less efficient but correct solutions would receive partial credit. We understand and expect that proofs of correctness will be more concise on exams than they would be on homework.

The exam is long, so be sure to work efficiently. Do not be discouraged, however, if you do not finish.

2 Topics Covered
This is a pretty comprehensive list of topics we have gone over this first half of the semester, but anything said in lecture through 2/13 inclusive is fair game for the exam.

2.1 Math/Fundamentals
- **Induction.** If $P(n)$ is a statement ("$\exists n$ is true"), $P(1)$ is true, and $\forall n, P(n) \rightarrow P(n+1)$, then $\forall n, P(n)$ is true.
- **Big-O Notation.** $o, O, \omega, \Omega, \Theta$ and identifying which of these relations hold for two functions.
- **Recurrence Relations.** Solve simple ones by finding a pattern and proving them with induction. More complicated recurrences can be solved using the Master Theorem (must memorize).
- **Integer Multiplication.** Perform 3 multiplications on $n/2$ digit numbers, and then do some additions.
- **Fast Powering.** Use repeated squaring to find an $n$th power in $O(\log n)$ time.
- **Merge Sort.** Sort a list of $n$ numbers in $O(n \log n)$ time. Implement recursively or iteratively.

2.2 Graph Search
- **Representation.** Adjacency list versus adjacency matrix
- **Depth First Search (DFS).** Uses a stack to process nodes, Pre and post order numbers, labels for the edges (tree, forward, back, cross).
  **Important Applications.** Detecting cycles, topological sorting, finding strongly connected components (relate to 2SAT).
- **Breadth First Search (BFS).** Uses a queue to process nodes, can be used to find the shortest path when all edge weights are 1.
• **Dijkstra’s Algorithm.** Single source shortest path for non-negative edge weights. Uses a heap or priority queue to process nodes. Does not work when there are negative edge weights (why?).

• **Heaps.** Binary heap implementation, operations: `DELETEMN`, `INSERT`, `DECREASEKEY`, how they are used in Dijkstra’s algorithm.

• **Bellman-Ford Algorithm.** Single source shortest path for general edge weights, edge relaxation procedure (referred to as `update` in the lecture notes), detecting negative cycles.

• **Floyd-Warshall Algorithm.** All pairs shortest paths via dynamic programming, detecting negative cycles.

• **Shortest Path in DAG.** Can be done in linear time via dynamic programming regardless of edge weights.

### 2.3 Minimum Spanning Trees

• **Basic Properties.** Connected, acyclic, $|E| = |V| - 1$.

• **Cut Property:** The basis for why our MST algorithms are correct, allows us to greedily add edges to a sub-MST.

• **Prim’s Algorithm.** Implemented in a similar fashion as Dijkstra’s, builds MST from a single vertex.

### 3 Practice Problems

**Problem 1.**
Answer True or False for the following:

(a) $T \ F \ 4\sqrt{n} = o(2^n)$

(b) $T \ F \ \log_5(n) = \Omega(\log_3(n))$

(c) $T \ F \ 2^{\log n} = \omega(n)$

(d) $T \ F \ \log(n)^{\log(n)} = o(n)$

(e) $T \ F \ \text{Suppose } T(n) = 2T(n/b) + n. \text{ Then, as } b \to \infty, \text{ we eventually get } T(n) = \Theta(1)$.

(f) $T \ F \ \text{If } T(n) = T(\sqrt{n}) + 1, \text{ then } T(n) = o(\log \log n)$.

**Problem 2.**
Given a directed acyclic graph $G = (V, E)$ and two nodes $u, v \in V$, calculate the number of distinct paths from $u$ to $v$. (Two paths are considered distinct if they have at least 1 vertex not in common)
Problem 3.
A directed graph \( G = (V, E) \) is called semiconnected if for each pair of distinct vertices \( u, v \in V \), there is either a path from \( u \) to \( v \) or a path from \( v \) to \( u \). Find an algorithm to determine whether a directed graph is semiconnected. *Hint:* Look at the SCC graph.

Problem 4.
Suppose you are given an adjacency-list representation of an \( n \)-vertex graph undirected \( G \) with non-negative edge weights in which every vertex has at most 5 incident edges. Give an algorithm that will find the \( K \) closest vertices to some vertex \( v \) in \( O(K \log K) \) time. *Warning:* Your solution’s run-time must not involve \( N \) in any way!

Problem 5.
We are given a directed graph \( G = (V, E) \) on which each edge \( (u, v) \in E \) has an associated value \( r(u, v) \), which is a real number in the range \( 0 \leq r(u, v) \leq 1 \) that represents the reliability of a communication channel from vertex \( u \) to vertex \( v \). We interpret \( r(u, v) \) as the probability that the channel from \( u \) to \( v \) will not fail, and we assume that these probabilities are independent. Give an efficient algorithm to find the most reliable path between two given vertices.

Problem 6.
True or False
(a) \( T \quad F \) The heaviest edge in a graph is never part of a MST.
(b) \( T \quad F \) If \( G \) has a cycle with a unique heaviest edge \( e \), then \( e \) cannot be part of any MST
(c) \( T \quad F \) The shortest-path tree computed by Dijkstra’s algorithm is necessarily an MST.
(d) \( T \quad F \) Prim’s algorithm works with negative weighted edges.
(e) \( T \quad F \) If \( G \) has a cycle with a unique lightest edge \( e \), then \( e \) must be part of every MST.

Problem 7.
Given a weighted graph with \( n \) vertices and \( m \leq n + 10 \) edges, show how to compute a minimum spanning tree in \( O(n) \) time.

Problem 8.
(Source: MIT 6.046 PS 4) Consider an undirected graph \( G = (V, E) \) with a weight function \( w \) providing non-negative real-valued weights, such that the weights of all the edges are different.

(a) Prove that, under the given uniqueness assumption, \( G \) has a unique Minimum Spanning Tree.
(b) Determine if each of these MST algorithms is correct or not. Explain or give a counterexample.

i. **Divide-and-Conquer MST:** Divide the set \( V \) of vertices arbitrarily into disjoint sets \( V_1 \) and \( V_2 \), each of size roughly \( V/2 \). Define graph \( G_1 = (V_1, E_1) \), where \( E_1 \) is the subset of \( E \) for which both endpoints are in \( V_1 \). Define \( G_2 = (V_2, E_2) \) analogously. Recursively find (unique) MSTs for both \( G_1 \) and \( G_2 \); call them \( T_1 \) and \( T_2 \). Then find the (unique) lightest edge that crosses the cut between the two sets of vertices \( V_1 \) and \( V_2 \), and add that to form the final spanning tree \( T \).

ii. **Cycle-Breaking MST:** The algorithm operates in phases. In each phase, the algorithm first finds some nonempty subset of the simple cycles in the graph. Then it identifies the heaviest edge on each cycle, and removes all these heavy edges. Phases continue until we have a spanning tree.