CS161 Final Examination

This is an open-book, open-note exam. Your exam should have 13 pages, and there are five questions totaling 120 points. You may use any notes, class handouts, and the main course textbook.

Your answers should be concise, and when possible should be a list of important points rather than prose. Wordy and/or irrelevant answers will reduce your score for that problem. Your answers should be the summary of work done on scratch paper that you do not hand in.

Remote SITN students should take the exam in one three-hour sitting, sending back both a faxed version (immediately, to **650-723-6092**), and the hardcopy to

Jerry Cain
Gates Building, Room 189
Stanford University
Stanford, CA 94305

Name: _____

SUID: _____

I accept the letter and spirit of the honor code. I have neither given nor received help on this exam. I pledge to write more neatly than I ever have before in my entire life.

(signed)_____

		Score	Grader
1. Reviewing the Basics	(20)	_____	_____
2. True/False	(24)	_____	_____
3. Binary Search Trees	(26)	_____	_____
4. Dynamic Programming	(20)	_____	_____
5. Graph Theory	(30)	_____	_____
Total	**(120)**	_____	

Problem 1: Reviewing the Basics (20 points)

a) [4 points] Prove that $(n+1)^2 = O(n^2)$ by giving constants n_0 and c in the definition of O-notation.

b) [6 points] Consider a priority queue that supports the operations **`Insert`** and **`Extract-Min`**. Argue that in the worst case, if we perform a mixture of n **`Insert`** and **`Extract-Min`** operations on the priority queue, there is at least one operation that takes $\Omega(\lg n)$ time. Assume that the order of elements in the priority queue is determined by comparisons only. (Hint: Recall that for any set of numbers, at least one number must equal or exceed the average.)

c) [10 points] When performing arithmetic on large numbers, it is often necessary to do the calculations in smaller parts. In this problem, you're going to multiply two n-bit numbers, X and Y. Assume that $X = A\ 2^{n/2} + B$ and $Y = C\ 2^{n/2} + D$, where A, B, C, and D are each $(n/2)$-bit numbers.

- [4 points] We can calculate XY using the fact that $XY = AC\ 2^{n} + (AD + BC)2^{n/2} + BD$. Assume that adding and subtracting two k-bit numbers can be done in $O(k)$ time, and also assume that multiplying any number by 2^{k}, requiring exactly k left shifts, can also be done in $O(k)$ time. Give a recurrence for $T(n)$, the time required to multiply two n-bit numbers, and then solve the recurrence.

- [4 points] We can also compute $XY = AC\ 2^{n} + ((A - B)(D - C) + AC + BD)2^{n/2} + BD$, **reusing** the calculations for AC and BD. Give a recurrence for $T(n)$, the time needed to multiply two n-bit numbers using this method, and solve the recurrence.

- [2 points] Which method is better, asymptotically?

Problem 2: True or False (24 points)

For each of the following problems, answer **`True`** or **`False`**, unless you don't know the answer. Correct answers will receive 2 points, incorrect answers will receive -2 points, and no responses will be awarded 0 points. If you are guessing, it is to your advantage to mark **`No Response`**.

◯ **True**
◯ **False**
◯ **No Response**

For any asymptotically nonnegative function $f(n)$, we have that $f(n) + o(f(n)\lg n) = \Theta(f(n))$. (That's a little o, not a big O).

◯ **True**
◯ **False**
◯ **No Response**

The recurrence relation $T(n) = 3T(n/3) + \lg n$ is not covered by any of the three cases of the Master Theorem.

◯ **True**
◯ **False**
◯ **No Response**

All sorting algorithms require $\Omega(n\lg n)$ in the worst case.

◯ **True**
◯ **False**
◯ **No Response**

The worst-case running time of **`Randomized-Quicksort`** on an array of size n is $\Omega(n^2)$.

◯ **True**
◯ **False**
◯ **No Response**

The $(\lg n)^{th}$ smallest number of n unsorted numbers can be determined in $O(n)$ worst-case time.

◯ **True**
◯ **False**
◯ **No Response**

A legal binary search tree on n integer keys in the range from 1 to n^2 can be synthesized in $O(n)$ worst-case time.

◯ **True**
◯ **False**
◯ **No Response**

The number of legal binary search tree structures on the first n counting numbers is $O(2^n)$.

◯ **True**
◯ **False**
◯ **No Response**

◯ True
◯ False
◯ No Response

The problem of determining the optimal order for multiplying a chain of matrices can be solved by a greedy algorithm, since it displays the optimal substructure and overlapping subproblems properties.

◯ True
◯ False
◯ No Response

We can always improve the asymptotic running time by choosing dynamic programming over top-down recursion.

◯ True
◯ False
◯ No Response

The topological sort of an arbitrary directed graph $G = (V,E)$ can be computed in $O(\max(V,E))$ time.

◯ True
◯ False
◯ No Response

In $O(E + V \lg V)$ time, Dijkstra's algorithm with Fibonacci heaps can be used to compute shortest paths from a source vertex to all other vertices in a graph $G = (V,E)$ with nonnegative edge weights.

◯ True
◯ False
◯ No Response

In a directed acyclic graph $G = (V,E)$ with nonnegative edge weights, the presence of an edge (u,v) E implies that $d[u] \leq d[v]$ is true at all times during Bellman-Ford.

Problem 3: Red-Black Trees (26 points)

a) [4 points] Consider the ordered insertion of the keys 1, 2, 3, 4, 5, …, n-1, and finally n, into an initially empty red-black tree. Which insertion is the first to force a rotation? Draw the state of the tree just prior to the rotation, including all nil nodes and all colors.

b) [5 points] Draw the legal red-black tree storing 7 keys of maximum height. (Hint: what must the black-height of such a red-black tree be? What does that tell you about the tree height?) Make sure your tree is a legal red-black tree.

c) [7 points] Consider a red-black tree on n keys that maintains, in each node of the tree, the height of the node. (Recall that the height of a node is the maximum number of edges on any simple path from the node to one of its descendant leaves.) Describe how to update this height information efficiently when a rotation occurs. Analyze the running time of your update algorithm.

d) [10 points] You've already shown how to maintain a dynamic set of integers that supports **`Insert`**, **`Delete`**, **`Search`**, and **`Min-Gap`**. Leveraging off of either your answer to this problem set question, or off of my solution, explain how you would augment the dynamic set even further to support the set operation **`Nearest-Neighbors`**, which returns the two entries which are closer together than any other pair of numbers (thereby defining the **`Min-Gap`** value). The **`Nearest-Neighbors`** operation should run in constant time.

Problem 4: Maximizing profit (20 points)

Suppose you have one machine and a set of n jobs $a_1, a_2, a_3, \ldots, a_n$ to process on that machine. Each job a_k has a processing time t_k, a profit p_k, and deadline d_k. The machine can only process one job at a time, and job a_k must run uninterruptedly for t_k consecutive time units. If a_k is completed by its deadline, you receive a profit p_k, but if it is completed after its deadline, then you receive a profit of 0. Present an algorithm to find the maximum profit attainable by any schedule. Imagine that the first job is launched at time 0, and that the deadlines come as expiration times relative to the start of the first job. All jobs require between 1 and m time units, inclusive. You needn't actually construct the schedule.

Hints:

- Assume that the jobs are sorted by deadline, so that $d_1 \le d_2 \le d_3 \le \ldots \le d_{n-1} \le d_n$.
- Let $D[j,k]$ represent the maximum profit attainable from jobs $a_j, a_{j+1}, a_{j+2}, \ldots, a_n$ where jobs are constrained to start at time k or later. The desired result would then come via $D[1,0]$.
- Your algorithm should run in $O(mn^2)$ time.

(Additional Space for Problem 4)

Problem 5: Graph Theory (30 points)

Your responses to parts a and b should be concise—two sentences at most. Overly wordy responses will not receive full credit.

a. [6 points] Explain why Dijkstra's algorithm sometimes fails to report the correct set of minimal path distances when some of the edge weights are negative.

b. [6 points] Explain why the Bellman-Ford algorithm requires <u>only one</u> additional iteration (beyond the first $|V|-1$) to detect the presence of negative weight cycles? More precisely, why must at least one $d[v]$ value relax on that iteration if there are negative-weight cycles?

· [8 points] Argue that if all of the weights of a graph are positive, then any subset of edges that connects all of the vertices and has a minimum total weight must also be a tree. Give an example to show that the same conclusion does not follow if we allow some weights to be nonpositive.

. [10 points] A directed graph $G = (V,E)$ is **unipathic** if for any two vertices $u,v \in V$, there is at most one simple path for u to v. Suppose that a unipathic graph G has both positive and negative weight edges. Design an efficient algorithm to determine the shortest-path weights from a source s to all vertices $v \in V$ for such a unipathic graph. If some shortest-path weights to vertices reachable from s do not exist, then your algorithm should report that a negative-weight cycle exists in the graph. What is the running time of your algorithm?