Final Exam

Instructions:
- Do not open this quiz booklet until directed to do so. Read all the instructions on this page.
- Write your name below and circle your recitation at the bottom of this page.
- Write your solutions in the space provided. If you need more space, write on the back of the sheet containing the problem. Pages will be separated for grading.
- You are allowed three one-sided, letter-sized sheets with your own notes. No calculators or programmable devices are permitted. No cell phones or other communication devices are permitted.

Advice:
- You have 180 minutes to earn a maximum of 180 points. Do not spend too much time on any single problem. Read them all first, and attack them in the order that allows you to make the most progress.
- When writing an algorithm, a clear description in English will suffice. Using pseudo-code is not required.
- Do not waste time rederiving facts that we have studied. Simply state and cite them.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Parts</th>
<th>Points</th>
<th>Grade</th>
<th>Grader</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>180</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Name: ________________________________

Circle your recitation:
- R01 Brando Miranda 10AM
- R02 Brando Miranda 11AM
- R03 Parker Zhao 12PM
- R04 Alex Jaffe 12PM
- R05/R07 Danil Tyulmankov 1PM 2PM
- R06 Peinan Chen 1PM
- R08 Kevin Tian 2PM
- R09 Allen Park 3PM
- R10 Daniel Manesh 4PM
Problem 0. What is Your Name? [2 points] (2 parts)

(a) [1 point] Flip back to the cover page. Write your name and circle your recitation section.

(b) [1 point] Write your name on top of each page.
Problem 1. True or False (Asymptotics) [8 points] (4 parts)
Which of the following asymptotic relations are correct? No justification is needed here.

(a) T F If \( f(n) = 4n^2 + 5n + 4 \),
then \( f(n) = O(n^3) \).

(b) T F If \( f_1(n) = 5n + \log n \) and \( f_2(n) = 2n + 9\sqrt{n} \),
then \( f_1(n) \cdot f_2(n) = \Theta(n^2) \).

(c) T F If \( f_1(n) = \Theta(n) \) and \( f_2(n) = \Theta(n) \),
then \( \sqrt{f_1(n) \cdot f_2(n)} = \Theta(n) \).

(d) T F If \( f(n) = n + 1 \),
then \( n f(n) = O(n^n) \).
Problem 2. True or False (Recurrences) [8 points] (4 parts)
Which of the following are correct solutions to the given recurrences? In all cases, you may ignore roundoffs. No justification is needed here.

(a) T F \( T(1) = O(1), T(n) = 3T(n/3) + O(n^2). \)
   Proposed solution: \( T(n) = O(n^2) \)

(b) T F \( T(1) = O(1), T(n) = 8T(n/2) + O(n^3). \)
   Proposed solution: \( T(n) = \Theta(n^3) \)

(c) T F \( T(1) = O(1), T(n) = 9T(n/3) + O(n) \) for \( n > 1. \)
   Proposed solution: \( T(n) = \Theta(n^3) \)

(d) T F \( T(1) = O(1), T(n) = 2T(n/3) + 3T(n/9) + O(n^3). \)
   Proposed solution: \( T(n) = \Theta(n^3 \log n) \)
Problem 3. True or False (Max Heaps) [8 points] (4 parts)
Which of the following are true statements about every max heap $H$, when viewed as a tree? No justification is needed here.

(a) T F The tree satisfies the AVL balance property, that is, for every node $u$ in $H$, the height of the left subtree of $u$ and the height of the right subtree of $u$ differ by at most one.

(b) T F If we add a new key into the tree with the INSERT operation, this key is initially added as a leaf of the tree before being moved to its appropriate location.

(c) T F To build a new heap of $n$ elements requires $\Theta(n \log n)$ time.

(d) T F If we increase the value of the key at a single node $u$ of $H$, then max-heapify restores the heap property in time $O(\log n)$ for a heap of $n$ elements.
Problem 4. True or False (Gradient Descent and Newton’s Method) [12 points] (4 parts)
Which of the following facts about gradient descent are true? No justification is needed here.

(a) T F  In order to find the value of \(e^4\), we can run Newton’s method on the function
\[ f(x) = \ln x - 4 \]
with an appropriate choice of starting point \(x^{(0)}\).

(b) T F  Newton’s method is guaranteed to converge, from any starting point \(x^{(0)}\).

(c) T F  The gradient descent algorithm applied to the function
\[ f(x) = (x + 1)^2(x - 1)(x - 2) = x^4 - x^3 - 3x^2 + x + 2 \]
is guaranteed to converge to a global minimum of that function.

(d) T F  Every function that has a single global minimum is convex.
Problem 5. Sorting Prefixes [22 points] (3 parts)

Suppose that we are given a sequence $A[1\ldots n]$ of $n$ distinct elements that are $k$-prefix unsorted, for some integer $0 \leq k \leq \frac{n}{2}$. That is, the length-$s$ suffix of $A$, where $s = n - k$, is already sorted in non-decreasing order. (The elements of the length-$k$ prefix can be arbitrary and in arbitrary order.)

For example, $A = [3, 1, 5, 2, 4, 6]$ is a 3-prefix unsorted sequence of length 6.

(a) [6 points] Provide an asymptotic estimate of the worst-case bound (in terms of $n$ and $k$) on the number of comparisons that Insertion Sort makes on such $k$-prefix unsorted sequences.
(b) [8 points] Describe a comparison-based algorithm for sorting any $k$-prefix unsorted sequence that is efficient with respect to the number of comparisons it makes. (In other words, we care only about the number of comparisons and not the time efficiency here.) Provide an asymptotic worst-case estimate of the number of comparisons made by your algorithm. (You can assume that you know the value of $k$.)
(c) [8 points] Give an asymptotic lower bound (in terms of $n$ and $k$) on the number of comparisons needed by any comparison-based sorting algorithm to sort any $k$-prefix unsorted sequence $A$. Your lower bound here in part (c) and the one you obtain in part (b) should match. Briefly justify your result.

Hint: Recall that if $k \leq \frac{n}{2}$ then $\binom{n}{k} = \Theta(n^k)$. 
Problem 6. Currency Market Disequilibrium [16 points] (1 part)

Due to market inefficiencies, sometimes there is a short-term possibility for large financial gain in currency exchange markets. For example, imagine the following exchange rates:

<table>
<thead>
<tr>
<th>Currency Conversion</th>
<th>Exchange Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100 yen = 1 dollar</td>
<td>$R_{1,2} = 100 \text{ yen/dollar}$</td>
</tr>
<tr>
<td>1 euro = 90 yen</td>
<td>$R_{2,3} = 0.01111111 \text{ euros/yen}$</td>
</tr>
<tr>
<td>9 dollars = 8 euros</td>
<td>$R_{3,1} = 1.125 \text{ dollars/euro}$</td>
</tr>
</tbody>
</table>

Starting with 100 dollars, one could purchase 10,000 yen, use all of those yen to purchase euros, and all of the euros in turn to re-purchase dollars. The overall cash at the end of this purchase cycle would be $100 \cdot R_{1,2} \cdot R_{2,3} \cdot R_{3,1} = \$125$ (that is, a net profit of 25%; and the cycle can be followed repeatedly for increased profits).

Let $R$ be an $n \times n$ matrix representing the conversion rates between all $n$ currencies, with the element $R_{i_1, i_2}$ being the currency conversion rate from currency $i_1$ to currency $i_2$.

Alyssa P. Hacker would like to write an algorithm to monitor the exchange rates represented by the matrix $R$ to detect such profit cycles whenever they occur.

To help her, describe an $O(n^3)$ algorithm to determine whether or not there is a sequence of currency trades that produces a net profit (that is, detects whether there exists a situation in which $R_{i_1, i_2} \cdot R_{i_2, i_3} \cdot \ldots \cdot R_{i_{m-1}, i_m} \cdot R_{i_m, i_1} > 1$, for some currencies $i_1, \ldots, i_m$). You don’t need to identify the currencies used by this sequence.

Don’t forget to provide a running time analysis of your algorithm.

*Hint:* $\log(a \cdot b) = \log a + \log b$. 
**Problem 7. Ben Hashes Again** [24 points] (3 parts)

Ben Bitdiddle became a hashing aficionado. His favorite collision resolution scheme is open addressing and he is quite fond of the linear probing technique, that is, using a hash function $h_L$ in which the $i$-th probe in hashing a key $k$ is given by

$$h_L(k, i) = h(k) + i \pmod{m},$$

where $h(k)$ is a “classic” hash function and $m$ is the size of the hash table.

Still, being the 6.006 expert he is, he came up with a new open addressing hash function $h_E$ that he called “exponential hashing”, in which the $i$-th probe is given by

$$h_E(k, i) = h(k) + (2^i - 1) \pmod{m}.$$

He wants now to demonstrate the superiority of this new hash function over the linear probing hash function.

**(a) [8 points]** Consider a situation in which the function $h$ is defined as

$$h(k) = k \pmod{m},$$

and two ordered sequences

- $0, 1, \ldots, n/2 - 1$; and
- $m, m + 1, \ldots, m + n/2 - 1$,

were inserted into the initially empty hash table one after another. Here, $n = 2^t$, for some $t \geq 0$, and $n \leq m/4$.

Provide an asymptotic estimate (i.e., a $\Theta$-estimate) of the number of probes $P_L$ needed to perform all of these insertions using the hash function $h_L$. Show the work that led to this estimate.
(b) [8 points] Provide an asymptotic estimate (i.e., a $\Theta$-estimate) of the number of probes $P_E$ needs to perform all the insertions described in part (a) using the hash function $h_E$. Show the work that led to this estimate.
(c) [8 points] Assume that the function $h$ satisfies the simple uniform hashing assumption (SUHA).

Devise an initial configuration of the occupied slots of the hash table such that the expected number $P'_L$ of probes needed to insert a new key $k$ into that hash table when using the hash function $h_L$ is asymptotically smaller than the expected number $P'_E$ of probes needed to insert the key $k$ into the hash table using the hash function $h_E$.

Remember to provide both a precise description of the initial configuration and asymptotic estimates of $P'_L$ and $P'_E$. Show the work that led to these estimates.
Problem 8.  **Stock Gain Problem with Dynamic Programming**  [20 points]  (2 parts)

It turns out that the stock gain problem from Lecture 1 can be solved with dynamic programming. In fact, one can use this technique to solve the following, more general, variant of that problem in which we have an ability to execute multiple trades of the same stock, but are constrained to at most one trade of one share of stock per day.

Specifically, let $A[1..n]$ be the daily prices of the stock. Our goal is to make as much gain as possible by executing a sequence of buy-and-sell transactions, with each transaction happening on a different day. In other words, we want to find a sequence $(b_1, s_1), \ldots, (b_t, s_t)$ of $t$ buy-and-sell transactions, for some $0 \leq t \leq n/2$, with $b_i < s_i$ being the buy and sell days for the $i$-th trade and all $b_i$s and $s_i$s being distinct, that maximizes our corresponding gain defined as

$$
\sum_{i=1}^{t} (A[s_i] - A[b_i]).
$$

For example, if the daily prices were $A = [1, 10, 5, 2, 8, 4]$ then the maximum gain achievable would be 15 and it would correspond to executing trades $b_1 = 1$, $s_1 = 2$, $b_2 = 4$, and $s_2 = 5$.

On the other hand, if the daily prices were $A = [8, 4, 5, 7, 9]$ then the maximum gain achievable would be 7 and it would correspond to executing trades $b_1 = 2$, $s_1 = 5$, $b_2 = 3$, and $s_2 = 4$. (Note that in this example the trades are nested.)

(a)  [10 points]  Let $V[i, j]$, for $1 \leq i < j \leq n$, be the maximum gain achievable by executing multiple complete buy/sell trades between day $i$ and day $j$ (inclusive).

State a valid recurrence for $V[i, j]$ that can be used to compute $V[i, j]$ with a dynamic programming approach. Analyze the running time of the resulting algorithm.

*Hint: Note that without loss of generality we can assume that in the optimal solution for different buy-and-sell transactions do not “cross”. That is, we can choose labeling so it can never be the case that $b_i < b_{i'} < s_i < s_{i'}$, for any $i \neq i'$.

(In the second example above, an equivalent solution with “crossed” transaction is $b_1 = 2$, $s_1 = 4$, $b_2 = 3$, and $s_2 = 5$. We will always choose the equivalent “uncrossed” solution for convenience.)
(b) [10 points] Now consider a modification of the problem in which you are allowed to own no more than $k$ copies of the stock at any time. Describe a modification of the above dynamic programing approach to tackle this new version of the problem. Don’t forget to analyze the running time of the resulting algorithm.
Problem 9. **Cookie Quality Control** [20 points] (2 parts)

To support the increasing need for high-volume, high-quality, fast-response-time cookie deliveries, Prof. Madry has established the 6.006 bakery in the Stata Center. Excellence, integrity, and above all asymptotic efficiency are the hallmarks of the new facility.

To ensure cookie perfection, a special, highly sensitive testing apparatus has been installed that analyzes cookies by weight two-at-a-time and determines which is heavier. (It is so sensitive that no two cookies weigh exactly the same. We assume here that each use of the apparatus takes constant time.)

Prof. Madry finds that he spends more and more time at the bakery, and that it and the testing apparatus are an inspiration for many advances in algorithm design. For instance, he was able to develop an $O(n)$ (linear time) algorithm that finds from among a set of $n$ cookies, one that is both heavier than at least a quarter of the cookies and lighter than at least (another) quarter. (Note that this algorithm returns only the cookie and *not* these two quarters.)

(a) [10 points] Using the “quartering” algorithm from above, design an algorithm that identifies the $k$-th heaviest cookie in a set of $n$ cookies. Give and solve the recurrence that describes the running time of your algorithm.
(b) [10 points] If one did not have the quartering algorithm, one could solve the problem of finding the $k$-th heaviest cookie by using a comparison-based sorting method, e.g., Merge sort, which would run in $\Theta(n \log n)$ time. This seems to be doing too much work, as it solves the problem more thoroughly than needed. (It sorts all the cookies, and at worst we only need the top $k$.)

Design an alternative approach to finding the $k$-th heaviest cookie that does not use the quartering algorithm and outperforms the $\Theta(n \log n)$ sorting bound, for sufficiently small $k$. Your algorithm should run in $O(n + k \log n)$ time.
Problem 10. Winning a Game of Strategy [20 points] (3 parts)

Consider a one-player game in which there are \( r \) red chips and \( b \) blue chips on a table, with \( r = R \) and \( b = B \) initially. In each move, you must remove either one red chip or one blue chip. At the end of your move, you add to the running sum (which is initially zero) a number \( f(r, b) \), where \( r \) and \( b \) are the number of red and blue chips, respectively, remaining on the table at the end of that move, and \( f \) is a certain function known to you in advance. The game continues until there are no chips remaining on the table. Your goal is to maximize the value of the running sum at the end of the game.

(a) [8 points] Describe an algorithm that computes, for a given function \( f \) and the initial number \( R \) and \( B \) of red and blue chips, the maximum value of the sum achievable. What is the running time of your algorithm?
(b) [4 points] A new rule has been added to the game, and now you lose automatically if the product of the number of red chips and blue chips remaining at the end of one of your moves is 6006. How would you modify your algorithm to correctly adapt to this new rule?
(c) [8 points] Let us discard the new rule from part (b) now and consider a two-player version of the game, in which you alternate turns with an adversary. On his or her turn, the adversary also removes either a red or a blue chip, and we add to the running sum the value of \( f(r, b) \) corresponding to the number of red and blue chips remaining on the table after the move. The adversary, however, chooses his or her moves so as to make the final sum as small as possible.

Describe an algorithm that computes, for a given function \( f \) and the initial number \( R \) and \( B \) of red and blue chips, the largest final value that you can guarantee that you will at least be able to achieve in the two-player game against an adversary that plays optimally. What is the running time of this algorithm? Assume that you make the first move and that \( R + B \) is even.
Problem 11. Retrieving the 6.006 Crown [20 points]  (3 parts)

You embark on a quest to retrieve a legendary treasure, the 6.006 Crown, from a maze. The good news is that you already have a map of the maze, with the location of the treasure marked on it. The not-so-good news is that there is a guardian, a particularly nasty ogre, that patrols the maze—and you definitely don’t want to encounter him before recovering the crown. (Once you have the crown in hand, its algorithmic powers will let you deal with the ogre handily.) You need to find a route through the maze that reaches the crown while making certain to avoid the ogre (if such a route exists).

The maze can be modeled as an undirected graph \( G = (V, E) \), with vertices representing the entrance \( s \), the location of the treasure \( t \), and the starting location of the ogre \( g \) (which is distinct from \( s \) and from \( t \)), marked on it. You start at the entrance \( s \) and, in each time step, you can traverse one of the edges incident on the vertex you are currently at. Similarly, the ogre starts at the vertex \( g \) and, in each time step, can either traverse one of the incident edges or stay put.

(a) [8 points] Design an \( O(V + E) \) time algorithm that given \( G \) either:

- returns an \( s \to t \) path \( P \) in \( G \) such that if you follow it you are guaranteed to avoid being intercepted by the ogre. That is, no matter what sequence of moves the ogre makes from \( g \), he will not reach any vertex of the path (including the last vertex \( t \)) before or at the same time that you do;
- or (correctly) concludes that no such path \( P \) exists in \( G \).
(b) [6 points] Imagine now that the ogre, due to his size, cannot pass through certain narrow corridors. That is, the graph $G$ has some subset $\hat{E}$ of edges marked as narrow and the ogre cannot traverse them (but you still can). Provide an $O(V + E)$ time algorithm that solves the task defined in part (a) while taking the existence of narrow edges into account.
(c) [6 points] Suppose next that, thanks to your mastery of the 6.006 material, you possess the power to teleport yourself to an adjacent room, and thus traverse the corresponding edge instantaneously.

Clearly, as long as there exists a path from $s$ to $t$ that avoids the ogre’s starting room $g$, by using this teleportation power sufficiently many times, you can always reach the 6.006 crown while avoiding interception by the ogre. (We assume here that the ogre has good enough reflexes that it is impossible to “teleport through him”.) However, your 6.006 instructors taught you that such power should be used only as often as is absolutely necessary.

Design an $O(V + E)$ algorithm that computes the minimal number $\ell^*$ of uses of that power needed to evade the ogre. (You should ignore here the existence of narrow corridors introduced in part (b).)