Instructions. This is a 150-minute test, with a total of 104 points available. There are four pages and six problems. Note that you do NOT need to give the pseudo code of the algorithms listed in the lecture notes once referenced; instead, you may simply give the name of the algorithm. **Pace Yourself and Good Luck!**

**Problem 1. (48 pts total)** Please give a brief answer for each of the following questions.

**Q1.** Order the following five functions from the slowest to the fastest growth:

\[ n^{4/5}, \quad (\sqrt{n})^{\lg n}, \quad \sqrt{n} \cdot (\lg n)^4, \quad (\lg n)^{\sqrt{n}}, \quad (\lg n)! \]

**Q2.** Let \( f(n) \) and \( g(n) \) be asymptotically positive, prove that \( \max\{f(n), g(n)\} = \Theta(f(n) + g(n)) \) by showing both the \( O \) and \( \Omega \) relations.

**Q3.** Consider the following algorithm for computing the square root of a number:

```
Square-Root(x)
1 for i = 1, ..., x/2
2 if i^2 = x
3 output i
```

Does this algorithm run in polynomial time? Justify your answer.

**Q4.** Show the major operations of Quicksort on the **letter** input, **NTURULES**, to make the eight characters into alphabetical order, based on the partition procedure discussed in class. Please mark the two **U**’s as **U**₁ and **U**₂, according to their orders in the input, and show their positions during the processing.

**Q5.** Can we have a priority queue in the comparison sorting model with both the properties being satisfied simultaneously: (1) EXTRACT-MIN runs in \( O(\lg \lg n) \) time, and (2) BUILD-MAX-HEAP runs in \( O(n) \) time? Why?
Q6. Suppose that an array contains \( n \) numbers, each of which is 0, 1, 2, \ldots, or 100. Is it possible that we sort this array in the worst-case, little-oh \( o(n \lg n) \) time? How to do it if it can and what is the best time complexity you can achieve? Or why it is not possible?

Q7. Strassen’s algorithm relies on the fact that two \( 2 \times 2 \) matrices can be multiplied using only 7 multiplications and 18 additions/subtractions. Suppose you found a way to multiply two \( 2 \times 2 \) matrices using 3 multiplications and 10 additions/subtractions. Based on this, you can devise an algorithm similar to Strassen’s algorithm for multiplying two \( n \times n \) matrices. What would the running time of this algorithm be?

Q8. Given three matrices \( A_1, A_2, A_3 \) of dimensions \( 5 \times 3, 3 \times 1, 1 \times 7 \), respectively. Fill the missing fields \( a, b, c, \) and \( d \) in the \( m \) and \( s \) tables below, where \( m[i, j] \) gives the minimum number of scalar multiplications needed to compute \( A_i A_{i+1} \ldots A_j \) and \( s[i, j] \) records the value of \( k \) such that the optimal parenthesization of \( A_i A_{i+1} \ldots A_j \) splits the product between \( A_k \) and \( A_{k+1} \). Find an optimal parenthesization of the matrix-chain product \( A_1 A_2 A_3 \).

\[
\begin{array}{ccc}
 & a & \\
\hline
j & b & c \\
0 & & 0 \\
\end{array}
\quad
\begin{array}{ccc}
 & d & \\
\hline
j & & i \\
1 & 2 \\
\end{array}
\]

Problem 2. (2 pts total) Please give your opinion(s)/comment(s) on this course (e.g., strengths and weaknesses)? Any specific comments that may improve the quality of this course are very much welcome. Thank you.

Problem 3. (10 pts total) For each of the following recurrence relations, give the asymptotic growth rate of the solution using the \( \Theta \) notation. Assume in each case that \( T(n) \) is \( \Theta(1) \) for \( n \leq 2 \).

(a) (5 pts) \( T(n) = 3T(n/2) + n^2/\lg n \). Show your steps!

(b) (5 pts) \( T(n) = T(pn) + T(qn) + n \), where \( p + q < 1 \). Show your steps!
Problem 4. (12 pts total) Let $A[1..n]$ be an array of $n$ distinct numbers. If $i < j$ and $A[i] > A[j]$, then the pair $(i, j)$ is called an inversion of $A$. Inversions are often used for collaborative filtering to match your preferences (for books, movies, music, etc.) with other people on the internet. Once the web site has identified people with similar tastes to yours—based on your rating or search of things and the inversion computations—it can recommend you new things that these other people have liked.

(a) (2 pts) Give the four inversion pairs of the array $< 2, 4, 3, 1 >$.

(b) (6 pts) Give an efficient, subquadratic (little-oh $o(n^2)$) algorithm that determines the number of inversions in any permutation on $n$ elements by modifying the Merge procedure in MergeSort discussed in class. Instead of giving a detailed pseudo code, it suffices to describe your procedure. See below for the MergeSort pseudo code from class. (Hint: Think about how you count the number of inversions while merging.)

(c) (4 pts) First give the recurrence of the running time for your overall algorithm that counts the number of inversions and then derive the $\Theta$ notation for its time complexity.

<table>
<thead>
<tr>
<th>MergeSort($A, p, r$) // $A$: array of integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 if $p &lt; r$</td>
</tr>
<tr>
<td>2 $q = \lfloor (p + r) / 2 \rfloor$</td>
</tr>
<tr>
<td>3 MergeSort($A, p, q$)</td>
</tr>
<tr>
<td>4 MergeSort($A, q + 1, r$)</td>
</tr>
<tr>
<td>5 Merge($A, p, q, r$)</td>
</tr>
</tbody>
</table>

Problem 5. (15 pts total) Figure 1 shows a binary tree with seven nodes.

(a) (3 pts) Label the tree in Figure 1 with numbers from the set $\{1, 2, 4, 5, 6, 7, 8\}$ so that it is a legal binary search tree.

(b) (3 pts) Label each node in the tree with $R$ or $B$ denoting the respective colors RED and BLACK so that the tree is a legal red-black tree.
(c) (3 pts) Make the left child of the root be the root by performing a single rotation. Draw the binary search tree that results, and label your tree with the keys from Part (a). Is it possible to label the nodes with colors so that the tree is a red-black tree? Justify your answer.

(d) (3 pts) Give the red-black tree that results from inserting the key 3 into the tree in Part (b).

(e) (3 pts) Give the red-black tree that results from deleting the key 6 from the tree in Part (b).

Problem 6. (17 pts total) Let \( X = x_1x_2\ldots x_m \) and \( Y = y_1y_2\ldots y_n \) be two character strings. This problem asks you to find the maximum common substring length for \( X \) and \( Y \). Notice that substrings are required to be contiguous in the original strings. For example, \textit{photograph} and \textit{tomography} have common substrings \textit{ph}, \textit{to}, \textit{ograph}, etc. The maximum common substring length is 6.

(a) (3 pts) Figure 2 gives the computation of the maximum common substring length on the two strings, \( ABAB \) and \( BAB \), similar to the table used for computing the length of LCS in class. Only partial results are given. Please complete all the entries in the table.

(b) (6 pts) Dynamic programming can be used to find the longest common substring efficiently. The idea is to find length of the longest common \textbf{suffix} for all substrings of both strings. Suppose \( \gamma = \alpha\beta \) is the concatenation of two strings; we say that \( \beta \) is a suffix. Find the optimal substructure for the longest common suffix problem (a recurrence relation for \( LCSuff(X, Y, m, n) \)).

(c) (4 pts) Define the longest common substring length \( LCSubStr(X, Y, m, n) \) in term of \( LCSuff(X, Y, i, j) \).

(d) (4 pts) Give a dynamic programming algorithm for solving this problem. What are the time and space complexity of your algorithm?

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 2: Table for computing the maximum common substring length.