1. (15 points) Sorting and Order Statistics (I). Mark all correct answers. Justification is optional.

a. To sort a general array, which of the following algorithms (in their textbook forms, i.e., no modifications) satisfy the following two criteria: (1) is stable; (2) is in-place?

   (a) Randomized quick sort
   (b) Merge sort
   (c) Insertion sort
   (d) Heap sort
   (e) Counting sort
   (f) None of the above

b. To sort a general array, which of the following algorithms (in their textbook forms, i.e., no modifications) satisfy the following two criteria: (1) runs in $\Theta(n \log n)$ time on average and (2) is in-place?

   (a) Randomized quick sort
   (b) Merge sort
   (c) Insertion sort
   (d) Heap sort
   (e) Counting sort
   (f) None of the above

c. Suppose that we have an array of $n \geq 1000$ data records to sort, and that the key of each record is a 4-bit binary number (e.g., 0000, 0011, etc.). Which of the following algorithms satisfy the following two criteria: (1) runs in $\Theta(n)$ time; (2) is stable.

   (a) Counting sort
   (b) Merge sort
   (c) Heap sort
   (d) Partition
   (e) None of the above

d. Suppose that we have an unsorted array of $n \geq 1,000,000$ real numbers (no duplicates in the array), and we want to find and output the smallest 1000 numbers in the array. The order of the numbers in the output is not important. Which of the following strategy will correctly output these numbers in $\Theta(n)$ time even in the worst case?

   (a) Sort the array with merge sort and then output the first 1000 numbers.
   (b) Find the 1000-th smallest element using RANDOM_SELECT and output elements less than or equal to the 1000-th smallest element.
   (c) Build a min-heap on the array, and then do 1000 EXTRACT_MIN from the heap.
   (d) None of the above
e. Which of the following statements is CORRECT?

(a) Quick sort and Randomized quick sort have the same average-case asymptotic running time.

(b) Finding the $i$-th smallest element from an unsorted array can be done in $\Theta(\log n)$ time.

(c) In the worst case, SELECT has the same running time as quick sort.

(d) If an array $A$ is sorted in non-decreasing order, reversing $A$ will result in a max heap.

(e) The worst-case running time of Randomized SELECT is $\Theta(n \log n)$.

(f) None of the above.

f. To sort an input array of $n$ random integers in the range of 1 to $n/2$, which of the following algorithm is the most time efficient?

(a) Randomized quick sort

(b) Merge sort

(c) Heap sort

(d) Counting sort

g. Suppose that you have a quick sort implementation that always takes the first element as pivot for partition. For an input array of $n$ real numbers that is already sorted, what will be the algorithm's running time?

(a) $\Theta(n)$

(b) $\Theta(n \log n)$

(c) $\Theta(n^2)$

(d) $\Theta(\log n)$

h. Which of the following sorting algorithm satisfies the following three criteria: (1) runs in $O(n \log n)$ time even in the worst case, (2) is in-place, and (3) is stable?

(a) Standard merge sort

(b) Randomized quick sort

(c) Heap sort using the original order of the elements as a secondary key

(d) Counting sort
2. (10 points) Sorting and Order Statistics (II).

Suppose that you have an array of \( n \geq 1,000 \) objects, each of which has two keys: the first key is a short string of two upper-case letters (e.g., 'AA', 'CF', etc.), and the second key is a 5-bit binary number (e.g., 01011, 00101, etc). You want to order the objects using the first key primarily, and use the second key to break ties. You have a library that implements several sorting algorithms and you'd like to choose a combination of them to accomplish the sorting task. For each of the following strategies, first indicate whether it can correctly sort the objects. If yes, give the expected running time of the strategy as a function of \( n \), and indicate whether the sorting is in place or stable.

- Strategy 1: Run quick sort on the first key, and then counting sort on the second key
- Strategy 2: Run counting sort on the first key, and then counting sort on the second key
- Strategy 3: Run counting sort on the second key, and then quick sort on the first key
- Strategy 4: Run counting sort on the second key, and then counting sort on the first key
- Strategy 5: Run counting sort on the second key, and then insertion sort on the first key
- Strategy 6: Run quick sort on the second key, and then merge sort on the second key
- Strategy 7: Run merge sort on the second key, and then merge sort on the second key

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Sort correctly?</th>
<th>If sort correctly, answer following question:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Expected running time</td>
</tr>
<tr>
<td>Strategy 1</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Strategy 2</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Strategy 3</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Strategy 4</td>
<td>Yes</td>
<td>( O(n) )</td>
</tr>
<tr>
<td>Strategy 5</td>
<td>Yes</td>
<td>( O(n^2) )</td>
</tr>
<tr>
<td>Strategy 6</td>
<td>Yes</td>
<td>( O(n \log n) )</td>
</tr>
<tr>
<td>Strategy 7</td>
<td>Yes</td>
<td>( O(n \log n) )</td>
</tr>
</tbody>
</table>

3. (15 points) Heaps

a. Does the array \([15 7 11 8 4 9 5 3 2 6]\) represent a max heap? Why or why not?

b. Consider the following procedure for building a max heap. The algorithm takes an unsorted array \( A \) as an input and make \( A \) a heap.

\[
\text{BuildHeap}(A) \{
    \text{heap.size}(A) = \text{length}(A);
    \text{for} \ (i = \text{floor}(\text{length}(A)/2) \ \text{down to} \ 1) \n    \ \ \ \text{Heapify}(A, i);
\}
\]
(a) Illustrate how buildheap works using the following example. Each step is the result of one iteration of the for loop in the above pseudocode.

(b) What is the time complexity to build a min heap with \( n \) elements?

\( \mathcal{O}(n) \)
(c) Starting from the heaps on the left, show the contents of the new heaps after each heap operation.

Insert 14

ExtractMax

Decrease the 2nd key (12) to 5
4. (12 points) Use dynamic programming to find LCS between two strings BAABAB and BBAAAB. In your trace-back, show only one path (if there are multiple), and the LCS corresponding to that path.

\[
\begin{array}{cccccc}
B & A & A & B & A & B \\
0 & 0 & 0 & 0 & 0 & 0 \\
B & 0 & 1 & 1 & 1 & 1 & 1 \\
B & 0 & 1 & 2 & 2 & 2 & 2 \\
A & 0 & 1 & 2 & 3 & 3 & 3 \\
A & 0 & 1 & 2 & 3 & 3 & 4 \\
B & 0 & 1 & 2 & 3 & 4 & 4 & 5 \\
\end{array}
\]

LCS: BAAAB

5. (8 points) Dynamic programming for the shortest path problem on a special graph.

Use dynamic programming to find the shortest path from node S to node G on the following directed graph. Note that one can only move following the directions of the arrows. For full credit, show both the shortest path and its length.

![Directed Graph]

6. (20 points) Checkboard problem. You are given a checkboard which has 3 rows and \( n \) columns, and has an integer written in each square. You are allowed to select a number from each column so as to maximize the sum of the selected numbers. There is one constraint: you cannot select two squares that share a vertical edge. For example, given the checkboard below, you can select 6, 7, 3, 8 or 5, 7, 9, 7, among other possibilities. However 6, 7, 9, 10 is not a legal selection, since the last two numbers were selected from two adjacent squares. The optimal selection is 6, 9, 5, 10 (or 6, 7, 9, 8), which sums to 30.

\[
\begin{array}{cccc}
6 & 5 & 3 & 7 \\
4 & 7 & 5 & 8 \\
5 & 9 & 9 & 10 \\
\end{array}
\]

a. (3 points) A natural greedy algorithm is to select the largest number in the first column, and then for the remaining columns, always select the square with the largest number that is not adjacent to the square selected in the preceding column. Show that this algorithm does
not correctly solve this problem, by giving an example of a $3 \times 3$ checkboard on which the algorithm does not return the correct answer.

\[
\begin{array}{ccc}
3 & 100 & 2 \\
2 & 2 & 100 \\
1 & 1 & 1 \\
\end{array}
\]

**Optimal:** $2 + 100 + 100 = 202$

**Greedy:** $3 + 2 + 2 = 7$

b. (3 points) Let $c(i, j)$ represents the number in the $i$-th row and $j$-column of the checkboard, $F(i, j)$ represents the sum of the optimal selection for the first $j$ columns if you must select the $i$-th row in the $j$-th column. Write down a recurrence for computing $F(i, j)$. Which $F(i, j)$ would give you the optimal solution for the complete checkboard?

\[
F(i, j) = c(i, j) + \max_{k < i} F(k, j - 1)
\]

c. (14 points) Show how to use the recurrence above and dynamic programming to solve the following checkboard. Its running time should be $\Theta(n)$.

\[
\begin{array}{cccccccc}
c(i, j) & & & & & & & \\
0 & 4 & 1 & 9 & 4 & 6 & 8 & 9 & 0 & 7 \\
7 & 4 & 1 & 3 & 9 & 7 & 5 & 2 & 3 & 2 \\
5 & 6 & 4 & 3 & 4 & 5 & 6 & 9 & 3 & 6 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
F(i, j) & & & & & & & \\
0 & 11 & 14 & 24 & 22 & 39 & 46 & 54 & 55 & 65 \\
7 & 9 & 14 & 18 & 33 & 35 & 44 & 48 & 58 & 59 \\
5 & 13 & 15 & 17 & 28 & 38 & 45 & 55 & 57 & 64 \\
\end{array}
\]
7. (10 points) Greedy algorithm.
Use greedy algorithm to solve the fractional knapsack problem. The knapsack has a weight limit of 10LB.

<table>
<thead>
<tr>
<th>Item ID</th>
<th>Weight (LB)</th>
<th>Value ($)</th>
<th>Value / Weight</th>
<th>Weight taken</th>
<th>Value taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>25</td>
<td>5</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>15</td>
<td>3</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>12</td>
<td>4</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>5</td>
<td>2.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Total</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>10</td>
<td>46</td>
</tr>
</tbody>
</table>

8a. (Extra Credit - 10 points) Longest common subsequence among three sequences.
The problem is to find a longest common subsequence among three strings, X, Y, and Z.

a. One simple idea is to use dynamic programming to first find LCS between two sequences, say, X, and Y, and then find the LCS between LCS(X, Y) and Z. Argue (or give an example to show) why this algorithm may not always find the optimal solution.

\[
X: \quad ABB \\
Y: \quad BBA \\
Z: \quad A
\]

\[\text{LCS}(X, Y, Z) = A\]

\[\text{LCS}(\text{LCS}(X, Y), Z) = \text{null}\]

b. Describe an efficient dynamic programming algorithm that can solve the problem, and analyze its running time. (Assume that all three sequences have the same length n.)

\[
\text{LCS}(X[1..i], Y[1..j], Z[1..k]) = \max \left\{ \begin{array}{l}
\text{LCS}(X[1..i], Y[1..j], Z[1..k]) \\
\text{LCS}(X[1..i] Y[1..j], Z[1..k]) \\
\text{LCS}(X[1..i], Y[1..j], Z[1..k]) \\
\text{LCS}(X[1..i], Y[1..j], Z[1..k]) \\
\end{array} \right. \\
\text{O}(n^3)
\]

\[
\Omega(X_i, Y_j, Z_k) = \begin{cases} 
1 & \text{if } X_i = Y_j = Z_k \\
0 & \text{otherwise}
\end{cases}
\]