1. (20 points) Sorting and Selection

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Expected time complexity</th>
<th>Worst-case time complexity</th>
<th>Extra memory complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Randomized Quicksort</td>
<td>Θ(n log n)</td>
<td>Θ(n²)</td>
<td>Θ(1)</td>
</tr>
<tr>
<td>Merge Sort</td>
<td>Θ(n log n)</td>
<td>Θ(n log n)</td>
<td>Θ(n)</td>
</tr>
<tr>
<td>Counting Sort</td>
<td>Θ(n + k)</td>
<td>Θ(n + k)</td>
<td>Θ(n + k)</td>
</tr>
<tr>
<td>Randomized Selection</td>
<td>Θ(n)</td>
<td>Θ(n²)</td>
<td>Θ(1)</td>
</tr>
<tr>
<td>Linear-time Selection</td>
<td>Θ(n)</td>
<td>Θ(n)</td>
<td>Θ(n)</td>
</tr>
</tbody>
</table>

b. When \( k \in \Theta(n) \), counting sort can be used to give a linear-time selection algorithm. If memory is a concern, then the randomized selection algorithm can be used, whose expected running time is linear in practice, even though its worst-case running time is quadratic.

c. When \( k \) is much larger than \( n \), counting sort loses its main advantage. The randomized selection algorithm should be preferred in practice, although the worst-case linear time selection algorithm has a theoretical advantage.

2. (5 points) Quicksort.

Theoretically, you can use the worst-case linear time selection algorithm to choose the median as the pivot, which guarantees that the two subarrays will have exactly the same size. Since finding medians only takes linear time, and partition also takes linear time, the running time of quick sort is \( T(n) = 2T(n/2) + \Theta(n) \). The solution to this recurrence is \( T(n) = \Theta(n \log n) \).

3. (23 points) Heaps

a. This is not a heap, since it is not a complete binary tree. The third node does not have a right child.

b. See figures on next page.

c. See figures on next page.

d. The time complexity to build a min heap is the same as to build a max heap, which is \( \Theta(n) \).

e. The worst-case time complexity of BuildHeap using HeapInsert is \( \Theta(n \log n) \). In the worst case, each insertion takes time \( h \), which is the current height of the heap. A heap with \( i \) elements have height \( \lfloor \log_2 i \rfloor \). Therefore, the overall complexity is \( \sum_{i=2}^{n} \lfloor \log_2 i \rfloor \), which is in \( \Theta(n \log n) \), as shown below.

\[
\sum_{i=2}^{n} \lfloor \log_2 i \rfloor \leq \sum_{i=2}^{n} \log_2 i \leq \log_2(n!) = \Theta(n \log n).
\]

\[
\sum_{i=2}^{n} \lfloor \log_2 i \rfloor \geq \sum_{i=2}^{n} (\log_2 i - 1) \geq \log_2(n!) - n = \Theta(n \log n - n) = \Theta(n \log n).
\]
4. (7 points) Hash tables.

0: 14
1: 8
2: 9→2
3: {} 
4: 18
5: 12→19
6: {} 

5. (15 points) Longest common subsequence (LCS).

a. See figure below.

\[ j \quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \]

\[ i \quad x_{ij} \quad y_{ij} \quad A \quad C \quad B \quad A \quad D \quad C \quad A \quad B \]

\[ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \]
\[ 0 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \]
\[ 0 \quad 1 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \quad 2 \]
\[ 0 \quad 1 \quad 2 \quad 2 \quad 3 \quad 3 \quad 3 \quad 3 \]
\[ 0 \quad 1 \quad 2 \quad 3 \quad 3 \quad 3 \quad 3 \quad 4 \]
\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 4 \quad 4 \quad 5 \quad 5 \]
\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 5 \quad 5 \quad 5 \]
\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6 \quad 6 \]
\[ 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 6 \quad 6 \]

b. There are two LCSs, ACBADA and ACBADC.

6. (15 points) Restaurant location problem.

The optimal solution $420k can be achieved by selecting location 1, 4, 7, and 10, as shown below.

Distance between locations (mi)

\[ 15 \quad 17 \quad 4 \quad 11 \quad 9 \quad 2 \quad 8 \quad 5 \quad 6 \quad 8 \]

Estimated Profit ($10k)

Total Profit

Select i

Do not select i

Best

15 17 17 26 26 28 34 31 40 42
0 15 17 17 26 26 28 34 34 40
15 17 17 26 26 28 34 34 40 42
7. (15 points) Shortest path problem.

![Shortest path problem diagram]

The shortest path is shown above in red. Its length is 19.

8. (20 points) Extra credit: Algorithm design.

a. The following pseudocode shows the algorithm.

```plaintext
FindMedian(X[1..n], Y[1..n])
if n == 1 then
    return (X[1] + Y[1])/2; /* median for two numbers. */
end if
/* If n is an odd number, f = c = (n+1)/2; otherwise f = n/2 and c = n/2+1. */
f = ⌊(n + 1)/2⌋;
c = ⌈(n + 1)/2⌉;
mX = (X[f] + X[c])/2; /* median of X */
mY = (Y[f] + Y[c])/2; /* median of Y */
if mX == mY then
    return mX.
else if mX < mY then
    return FindMedian(X[c..n], Y[1..f]);
else
    return FindMedian(X[1..f], Y[c..n]);
end if
```

The running time of the algorithm is \( T(n) = T(n/2) + \Theta(1) \). Using Master Theorem case 2, we have \( T(n) = \Theta(\log n) \).

b. Let S be a longest palindromic subsequence (LPS) of X. Then, we note that (1) S is a subsequence of X, and (2) S is also a subsequence of the reverse of X. That is, S must be a common subsequence of X and reverse(X). Therefore, to find the LPS of X, we can compute LCS(X, reverse(X)). However, if X and reverse(X) have several LCSs, then not all LCSs are palindromic. For example, if X is TACTA, then LCS(TACTA, ATCAT) can be any of the four strings: ACA, TCT, TCA, ACT, only two of which are palindromic. The actual LPS can be found easily by taking half of a LCS and reverse it to get the other half.

It is also easy to develop a DP algorithm for this problem directly. Let LPS(i, j) be the length of the longest palindromic subsequence for the substring X[i..j]. The recursion to compute LPS(i, j) is as follows:

\[
LPS(i, j) = \begin{cases} 
1 & \text{if } i = j; \\
LPS(i + 1, j - 1) + 1 & \text{if } x[i] = x[j] \text{ and } j > i; \\
\max\{LPS(i + 1, j), LPS(i, j - 1)\} & \text{if } x[i] \neq x[j] \text{ and } j > i 
\end{cases}
\]

When computing the dynamic programming table, first initialize all diagonal entries to 1. Then work on the upper triangle towards the upper-right corner. The length of the LPS is stored in the entry LPS(1, n).