1. **(20 points) Sorting and Selection**

   Indicate whether the following statement is true or false. Justification is NOT required but preferred.

   a. Quick sort and randomized quick sort have the same worst-case asymptotic running time.

       Answer: True. Both are $\Theta(n^2)$ in the worst case.

   b. In worst-case analysis, quick sort is more efficient than merge sort.

       Answer: False. Quicksort is $\Theta(n^2)$ and merge sort is $n \log n$.

   c. Merge sort runs in linear time in the best case.

       Answer: False. $\Theta(n \log n)$.

   d. Counting sort is usually more efficient than other sorting algorithms because its running time is $\Theta(n)$ while the others are at least $\Theta(n \log n)$.

       Answer: False. Counting sort is linear only if the input is a set of small integers or can be converted to small integers.

   e. Counting sort is a stable and in-place sorting algorithm.

       Answer: False. Not in place.

   f. Finding median from a sorted array takes the same amount of time as from an unsorted array.

       Answer: False. Constant time for the former.

   g. Heap sort has the same asymptotic running time as merge sort, but uses less space.

       Answer: True.
h. On average, randomized quick sort is expected to be more efficient than (standard) quick sort.

Answer: False. Asymptotically the same.

i. To find the largest element in an array, the most efficient approach is to build a max heap and do ExtractMax.

Answer: False. To find the largest element you can simply scan the array.

j. The worst-case running time of Randomized SELECT is \( \Theta(n) \) while the standard (non-randomized) SELECT is \( \Theta(n \log n) \).

Answer: false. \( \Theta(n^2) \) for both.

2. (15 points) Heaps

a. Does the array \([10 8 9 4 5 6 7 1 2 3]\) represent a max heap? Why or why not?

Yes. Child nodes are no larger than parent node.

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.

\[
\text{BuildHeap(}A\text{)} \{ \\
\text{heap_size(}A\text{)} = \text{length}(A); \\
\text{for (}i = \text{floor(length}[A]/2) \text{ downto 1)} \\
\quad \text{Heapify}(A, i); \\
\}
\]

(a) Illustrate how the procedure BuildHeap works using array \( A = [14 15 10 5 6 16 11 3 12 20] \). Show necessary intermediate steps for full credit (e.g., show the content of the tree after each call to Heapify). Make sure your final tree is indeed a heap.
(b) What is the asymptotically tight time complexity of BuildHeap?
   Answer: $\Theta(n)$.

(c) Suppose that we change the third line of BuildHeap to “for ($i = 1$ to $\text{length}[A]/2$)”, would the algorithm still work? If no, state the reason. If yes, what is the time complexity of the modified algorithm?
   Answer: No. Cannot call heapify because subtrees are not necessarily heaps.

(d) Now suppose that we change the third line to “for ($i = \text{length}[A]$ downto 1)”. Answer again the questions asked in (c).
   Answer: yes. same time complexity.

3. (17 points) Longest common subsequence (LCS).

a. (12 points) Use dynamic programming to find LCS between two strings BABABBAB and ABBAABBA. If there are multiple longest common subsequences, report all of them.

b. (5 points) Given two strings that contain only ‘A’s and ‘B’s, revise the LCS algorithm to find a “weighted” LCS, where for each ‘A’ in the common subsequence we receive 2 points and for each ‘B’ in the common subsequence we receive 3 points. Hint: Let WLCS($i, j$) represents the maximum value for the weighted LCS between $X[1..i]$ and $Y[1..j]$. To solve this problem it is sufficient to just define WLCS($i, j$) recursively. However you can fill in the table blow to validate your formula (Optional).

$$W LCS(i, j) = \max \begin{cases} 
W LCS(i - 1, j - 1) + \sigma(X[i], Y[j]) \\
W LCS(i, j - 1) \\
W LCS(i - 1, j)
\end{cases}$$

where

$$\sigma(a, b) = \begin{cases} 
2 & (\text{if } a=b= 'A') \\
3 & (\text{if } a=b= 'B') \\
0 & (\text{otherwise})
\end{cases}$$

4. (23 points) Dynamic programming II (Gas Station Location Problem).

a. (15 points) You are given a list of $n$ locations to consider building gas stations. Each location has its own estimated profit value $p_i$. Your task is to find an optimal selection that will result in the highest estimated total profit, subject to the policy that any two selected locations may NOT be adjacent to each other (i.e., you cannot selected both location $i$ and $i + 1$.)
(a) Let $S(i)$ be the estimated total profit of the optimal plan that considers only locations 1 to $i$. Write down the recurrence function for computing $S(i)$.

$$S(i) = \max \begin{cases} S(i - 2) + p_i \\ S(i - 1) \end{cases}$$

(b) Show how to use the recurrence function and dynamic programming to find the optimal plan given the estimated profits below.

<table>
<thead>
<tr>
<th>Location $i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit $p_i$</td>
<td>9</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>select = no</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>select = yes</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S(i)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Optimal value:

Locations (indices) selected in an optimal solution:

b. (8 points) Now consider a modification to the policy about the distance between selected locations. The new policy says that you cannot have three selected locations next to each other, i.e., for any three selected locations $i < j < k$, $k - i > 2$. (For example, you can select 1, 2, 4 or 1, 3, 4, but not 1, 2, 3.) Provide a dynamic programming algorithm to solve the problem. Hint: at each location, consider three possible cases: (1) the location is not selected, (2) the location is selected and the previous location is not selected, and (3) the location is selected and the previous location is also selected.

(a) Let $P(k, i)$ be the optimal total profit for the first $i$ locations, where location $i$ falls into case $k$ ($1 \leq k \leq 3$). Give the recurrence to calculate $P(k, i)$. 
\[ P(1, i) = \max_{1 \leq k \leq 3} P(k, i - 1) \]
\[ P(2, i) = P(1, i - 1) + p_i \]
\[ P(3, i) = P(2, i - 1) + p_i. \]

(b) Use the recurrence above and dynamic programming to find an optimal selection for the following problem.

<table>
<thead>
<tr>
<th>Location ( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profit ( p_i )</td>
<td>9</td>
<td>3</td>
<td>6</td>
<td>8</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>6</td>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

**Optimal value:**

**Locations (indices) selected in an optimal solution:**

5. (12 points) Greedy algorithm.

a. (8 points) Use greedy algorithm to solve the **fractional** knapsack problem. The knapsack has a weight limit of 10 LBs.

<table>
<thead>
<tr>
<th>Item ID</th>
<th>Weight (LB)</th>
<th>Value ($)</th>
<th>Value / Weight ($/LB)</th>
<th>Weight (LB) taken</th>
<th>Value ($) taken</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>15</td>
<td>5</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td>30</td>
<td>6</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>C</td>
<td>5</td>
<td>10</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>3</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>7</td>
<td>3.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Total</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>10</td>
<td>57</td>
</tr>
</tbody>
</table>

b. (2 points) If the value of each item in the list above is now increased by a certain fixed constant (e.g., $1), will you need to re-run the algorithm to get the optimal selection?

Answer: yes. As the change may affect the value/weight ratio differently for different items, their order may change and therefore you need to re-sort the items.
c. **(2 points)** If the value of each item in the list above is now increased by a certain fixed percentage (e.g., 20%), will you need to re-run the algorithm to get the optimal selection?

Answer: no. This does not change the relative order of the items by their value/weight ratio so if you actually rerun the algorithm you would get the same (fractional) items.

6. **(13 points) Minimum spanning tree.**

a. Find a minimum spanning tree for the following graph. It is your choice to use either the Prim’s or the Kruskal’s algorithm. For full credit, you need to:

   (a) Clearly state which algorithm you used.
   
   (b) Highlight the tree edges.
   
   (c) Label the tree edges by the order in which they were selected according to the algorithm you used.

It is not necessary to show all intermediate steps, such as the contents of the heap or array, but having this type of information available would help me in giving you partial credit if you did not get the correct tree.

![Graph Image]

b. Indicate whether the following statement is true or false. Justification is NOT required but preferred.

   (a) For a sparse graph with real-valued edge weights, Prim’s algorithm with priority queue is more efficient than Kruskal’s algorithm.

Answer: False. Both have running time $m \log n$. 


(b) For a sparse graph whose edge weights are all small integers between 1 and $k$, where $k$ is much smaller than the number of edges ($m$), Kruskal’s algorithm with counting sort is more efficient than Prim’s algorithm.

Answer: True if $m$ is at least $n \log n$. Kruskal’s is $(m + n \log n)$ while Prim’s is $m \log n$. False is $m$ is $\Theta(n)$ - both would be $n \log n$.

(c) For a completely connected, weighted graph, Prim’s algorithm with distance array is more efficient than Prim’s algorithm with priority queue. (Note: a completely connected graph is one where every pair of vertices is connected by an edge.)

Answer: True. Prim’s is $\Theta(n^2)$ and Kruskal’s become $\Theta(n^2 \log n)$. 