Protocol 1 Build-MaxHeap(A)

1: procedure Build-MaxHeap(A)
2: heap-size = length[A];
3: for i=floor(heap-size/2) downto 1 do
4:   do Max-Heapify(A,i)
5: end for
6: end procedure

a) (10pts) Give the formal pre-conditions for the algorithm Build-MaxHeap(A).
   \textbf{Answer:} \forall A[j]: j = \lfloor heap-size/2 \rfloor + 1, \ldots heap-size, A[j] is the root of a max-heap.

b) (10pts) Develop a loop invariant for this loop leading to the correct post condition that A is a max-heap.
   \textbf{Answer:} The same the answer above:
   \forall A[j]: j = \lfloor heap-size/2 \rfloor + 1, \ldots heap-size, A[j] is the root of a max-heap.

2) Your job is to design an algorithm to dispense change using the fewest possible coins.
   a) (15pts) Given the US coin system(Q=25, D=10, N=5, P=1) write an algorithm to make change in coins for any amount less than one dollar (one dollar = 100).
Protocol 2 Dispense–coins(value)
1: procedure Dispense–coins(value)
2: change = value
3: while value ≥ 25 do
4:  value = value − 25
5:  Dispense(Quarter)
6: end while
7: while value ≥ 10 do
8:  value = value − 10
9:  Dispense(Dime)
10: end while
11: while value ≥ 5 do
12:  value = value − 5
13:  Dispense(Nickle)
14: end while
15: while value ≥ 1 do
16:  value = value − 1
17:  Dispense(Penny)
18: end while
19: end procedure

b) (5pts) Is it possible to modify your algorithm for any coinage system? Explain or give a counter example.
Answer: No. A counter–example would be dispensing 14 using coins: 12, 10, 7, 5, 1
3) You have the following algorithm

Protocol 3 LCS-Length(A)

1: procedure LCS-LENGTH(X,Y)
2:   m = X.length;
3:   n = Y.length;
4:   let c[0...m, 0...n] be a new table
5:   for i = 0 to m do
6:     c[i, 0] = 0
7:   end for
8:   for j = 0 to n do
9:     c[0, j] = 0
10: end for
11: for i = 1 to m do
12:   for j = 1 to n do
13:     if x_i == y_j then
14:       c[i, j] = c[i - 1, j - 1] + 1
15:     else if c[i - 1, j] ≥ c[i, j - 1] then
16:       c[i, j] = c[i - 1, j]
17:     else
18:       c[i, j] = c[i, j - 1]
19:     end if
20:   end for
21: end for
22: return c[m, n]
23: end procedure

a) (15pts) Given LCS-Length(X,Y) show the correctly how the procedure finds the LCS for 
X=”CCAGTAC” and Y=”CATG’.
Answer:

<table>
<thead>
<tr>
<th></th>
<th>C</th>
<th>C</th>
<th>A</th>
<th>G</th>
<th>T</th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>→1</td>
<td>→1</td>
<td>→1</td>
<td>→1</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>↓1</td>
<td>↓1</td>
<td>→2</td>
<td>→2</td>
<td>→2</td>
<td>→2</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>↓1</td>
<td>↓1</td>
<td>↓1</td>
<td>↓1</td>
<td>↓1</td>
<td>↓1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>↓1</td>
<td>↓1</td>
<td>↓3</td>
<td>↓3</td>
<td>↓3</td>
<td>↓3</td>
</tr>
</tbody>
</table>

b) (5pts) What does this algorithm return as the LCS?
Answer: 3

4) The next two questions relate to the 0/1 Knapsack Problem with a capacity of 30 and the 
following weights and values: w_i[40, 15, 15] and p_i[30, 25, 25]
a) Find the Optimal Solution using dynamic programming

My Answer: Notice that all weights and the capacity can be evenly divided by 5 to 
cut the number of columns down to 8. If you used all 40 columns, you didn’t lose any
points but you did a lot of extra work.

\( c = 6, n = 3 \)

\[
\begin{array}{c|c|c|c}
 p_i & 30 & 25 & 25 \\
 w_i & 8 & 3 & 3 \\
\end{array}
\]

The optimal value is 50 and the items taken are: \( x_i = \langle 0, 1, 1 \rangle \)

b) Find a solution using a greedy algorithm
   **Answer:** Total value is 50 from \( x_i = [0, 1, 1] \)

c) Which is the best solution given these inputs?
   **Answer:** Due to a typo in the problem, both give the same answer.

5) Briefly define the Principle of Optimality
   **Answer:** Optimal solutions to a problem incorporate optimal solutions to related subproblems, which we may solve independently.