**Week 4: Dynamic Programming**

1. **longest\_common\_subsequence**:

* **Guide**:
  + To find the LCS between two strings, consider how comparing the characters of the strings might help you build the solution progressively. Utilize dynamic programming to store solutions to overlapping subproblems in a 2D table.
* **Pseudocode**:
* FUNCTION longest\_common\_subsequence(s1, s2):  
   INITIALIZE a table of size len(s1)+1 by len(s2)+1 to 0s  
   FOR each character in s1:  
   FOR each character in s2:  
   UPDATE table based on character matches or mismatches  
   RECONSTRUCT the LCS using the table  
   RETURN the LCS

1. **knapsack**:

* **Guide**:
  + Approach the 0/1 knapsack problem by considering if including or excluding an item is beneficial. Use a 2D table where entry [i][w] will be storing the maximum value which can be achieved with i items and w weight.
* **Pseudocode**:
* FUNCTION knapsack(weights, values, W):  
   INITIALIZE a table of size len(values)+1 by W+1 to 0s  
   FOR i from 1 to len(values)+1:  
   FOR w from 1 to W+1:  
   UPDATE table[i][w] based on the inclusion or exclusion of the item  
   RETURN table[len(values)][W]

1. **coin\_change**:

* **Guide**:
  + For this problem, think about building up the solution for each amount up to the given amount. If we can achieve a smaller amount using the least number of coins, how does that influence achieving a larger amount?
* **Pseudocode**:
* FUNCTION coin\_change(coins, amount):  
   INITIALIZE a list to represent the minimum coins for every amount from 0 to given amount  
   FOR each coin in coins:  
   FOR amt from coin to amount:  
   UPDATE the minimum coins needed for amt  
   CHECK if the amount is achievable with given coins  
   RETURN the minimum coins needed for given amount