Seonhye Yang (sy3420) <u>Homework 4</u>

Problem 1

$A = [1 \ldots\ldots n]$ → at most, can make $A[i]$ move / jump.
↑ frog

- It can jump $i, \ldots, i - A[i]$. Each element represents the maximum number of jumps length.

$$\text{CanReach}(n) = \text{Max}(n - \text{CanReach}(A(n)), \text{CanReach}(n-1))$$

What we are saying here is that we maximize jump because we can at most jump $1 - A[i]$.

We want max because the maximum jump length it can make is $A[i]$.

b. Base Case would be a case of a single or zero element array where no jump is required. We would return 0.
CanReach (0)

c. <u>Bottom-up DP</u>

```
• for i=n to 1:
      compute l(i)
  memo[0 ··· n] = []
  memo[0] = 0      base case
  for i=1 to n
      memo[i] = max(i - memo[l(i)], memo[i-1])
  return memo[n]
```

We used a nested for-loops for the pseudocode. Thus, the time complexity is $O(n^2)$. If we want a better algorithm, we could use greed method.

Problem 2

MakeChange (i) : # of ways to make change for i cents with use of dimes, nickels, and pennies.

a. MakeChange (8) = 1+1+1+1+1+1+1+1 (8 pennies) ①
1+1+1+5 (3p + 1n) ②
1+1+5+1 (2p + 1n + 1p) ③
1+5+1+1 (1p + 1n + 2p) ④
5+1+1+1 (1n + 3p) ⑤

5 - ways to make change

MakeChange (9) : 1+1+1+1+1+1+1+1+1 (9 pennies) ①
1+1+1+1+5 (4p + 1n) ②
1+1+1+5+1 (3p + 1n + 1p) ③
1+1+5+1+1 (2p + 1n + 2p) ④
1+5+1+1+1 (1p + 1n + 3p) ⑤
5+1+1+1+1 (1n + 4p) ⑥

6 - ways to make change.

b. Let's say there are n cents coins and we have our constraints / rules:
1 cent = pennies
5 cents = nickel
10 cents = dime

for $n \geq 10$, find MakeChange (n)

MakeChange (n) = MakeChange (n-1) + MakeChange (n-5) + MakeChange (n-10)

c.
```
A[n+1] = 0
A[0]   = 1
for i in 1 to n+1 :
  if i-1 ≥ 0 :
    A[i] = A[i-1]
  if i-5 ≥ 0 :
    A[i] = A[i-5]
  if i-10 ≥ 0 :
    A[i] = A[i-10]
```

Time Complexity = $O(3 \cdot n)$. Loop runs n times and calculates for i-1, i-5, i-10.

Problem 3

a. Given 2 strings $S[1 \dots n]$ and $T[1 \dots m]$
- Let $LCS(n, m)$ denote of longenst common substring of $S[1..n]$ and $T[1..m]$

① $LCS(n,m) = 1 + LCS(n-1, m-1)$ for $S[n-1] = T[m-1]$

This is if the last characters match, then reduce both length by 1.

② $LCS(n,m) = 0$ for $S[n-1] \neq T[m-1]$

The max length longest common Substring is the longest common substring.

- $LCS(n,m) = LCS(n-1, m-1) + 1$ for $S[n-1] = T[n-1]$ this is if the characters match, then we reduce length by 1.
- $LCS(n,m) = 0$ if $S[n-1] \neq T[m-1]$ This will be 0 since last charaters don't match.
- $LCS(n,m) = \max(LCS(i,j)) \Rightarrow$ for $1 \leq i \leq n$ and $1 \leq j \leq m$
  The maximum length is the longest common substring.

$$LCS(i,j) = \begin{cases} 1 + LCS(i+1, j+1) & \text{for } 0 \leq i[S], 0 \leq j[T], S[i] = T[j] \\ 0 & \text{otherwise} \end{cases}$$

b. The base case would be when the size = 0 of the string. In the case of this, we would have only 0 length as LCS. $LCS(n,m) = 0$

c.
```
for i to (n+1):
    for j to (m+1):
        if i==0 or j==0:
            LCS[i][j] = 0
        elif S[i-1] == T[j-1]:
            LCS[i][j] = LCS[i-1][j-1]+1
            final = max (final, LCS[i][j]
        else:
            LCS[i][j] = 0
        return final
```

Time Complexity $= O(n \cdot m)$. We are using 2 for loops for both strings, thus the TC of LCS is $O(n \cdot m)$.

Problem 4

row of n coins = $V_1, V_2 \dots V_n$
- selects first or the last coin from row.

Lets define ;

$M_{i,j}$ = The max values of coins taken by Alice for coins numbered i to j.

Optimal value for Alice = $M_{1,n}$

we have 1^{st} coin as V_1 and 2^{nd} coin as V_2 .... n^{th} coin as V_n. Given the row of coins from coin i to coin j, Alice can take either coin i or coin j. Thus, Alice can choose and gain value $V[i]$ or $V[j]$.

Assume Bob plays optimally.

Alice should pick coin on the following condition:

1. If $j = i+1$, then Alice must select the larger of $V[i]$ or $V[j]$. Then Alice will win.
2. Otherwise, if Alice picks coin i, the she would get total value:

$$\min\{M_{i+1,j-1}, M_{i+2,j}\} + V[i]$$

3. Otherwise, if Alice picks coin j, then the total value:

$$\min\{M_{1,j-2}, M_{i+1,j-1}\} + V[j]$$

- For $i = 1, 2, 3 \dots n-1$:

$$M_{i,i+1} = \max\{V[i], V[i+1]\}$$

- $M_{i,j} = \max\{\min\{M_{i+1,j-1}, M_{i+2,j}\} + V[i], \min\{M_{i,j-2}, M_{i+1,j-1}\} + V[j]\}$

Time Complexity:
- We can compute $M_{i,j}$ using memoization by starting with $M_{i,i+1} = \max\{V[i], V[i+1]\}$ and then computing all $M_{i,j}$ where $j-i+1$ .... and so on.

There are $O(n)$ iterations and each iteration runs in $O(n)$ time. Thus, total time for this algorithm is $O(n^2)$