Problem 1:

I don’t know

Problem 2:

1. T[i] = MinBills[i]

2. T[i] = min{T[i-1], T[i-6], T[i-27], T[i-38], T[i-50]} + 1

3. T[0] = 0

4. return T[N]

5. Pseudocode:

Initialize table T of length N + 1

T[0] 🡨 0

For i = 1 to N

If i < 6

T[i] = T[i-1] + 1

S[i] = k 🡨 {1} such that T[i] = 1 + T[i – k]

Else If i < 27

T[i] = min{T[i-1], T[i-6]} + 1

S[i] = k 🡨 {1, 6,} such that T[i] = 1 + T[i – k]

Else If i < 38

T[i] = min{T[i-1], T[i-6], T[i-27]} + 1

S[i] = k 🡨 {1, 6, 27} such that T[i] = 1 + T[i – k]

Else If i < 50

T[i] = min{T[i-1], T[i-6], T[i-27], T[i-38]}+ 1

S[i] = k 🡨 {1, 6, 27, 38} such that T[i] = 1 + T[i – k]

Else

T[i] = min{T[i-1], T[i-6], T[i-27], T[i-38], T[i-50]} + 1

S[i] = k 🡨 {1, 6, 27, 38, 50} such that T[i] = 1 + T[i – k]

End Else If

Return T[N] //Output number of bills

current – index = N //Output bill sequence

While current – index not equal 0

Print: “bill” S[current – index]

current – index 🡨 current – index – S[current – index]

Problem 3:

Part 1:

1. T[n] is the total number of possible sequences of moves (steps taken) per value of k to equal n
2. DP relation: T(n) = T(n-1) + T(n-2) + … T(n-k)
3. T[0] = 1
4. Sum of T[n]
5. Pseudocode:

Create array T where size is n + 1

numCombos(n, k)

Set T[0] = T[1] = 1

If n < 0

Return 0

If n == 0

Return 1

If T[n] == 0

For i = 1 until i <= k where i++ every loop

T[n] += numCombos(n - i, k)

End For

End If

Return T[n]

End function

Problem 3:

Part 2:

1. T[n] is the total number of possible sequences of moves ( steps taken ) per value of k to equal n and <= max steps ( 100 )
2. DP relation: T(n) = steps <= max steps 100 for each (T(n-1) + T(n-2) + … T(n-k))
3. T[0] = 0
4. Sum of T[n]
5. Pseudocode:

Set count = 0

NumCombosH(n, k)

If k=0

If length of the result is <= 100

count++

End If

End If

For i = 1 until i <= n where i++ every loop

If i <= k

NumCombosH(n, k - i)

End If

End If

Return count

End Function

Problem 4:

isSubsetSum(int arr[], int n, int sum)

Create boolean 2D array with 2 rows and sum + 1 columns

For i = 0 until i <= n while i++ every loop

For j = 0 until j <= sum while j++ every loop

If i = 0

Set T[i % 2][j] = false

Else If j = 0

Set T[i % 2][j] = true

Else if arr[i - 1] <= j

//set current index based on the true or false values

Set T[i % 2][j] to T[(i + 1) % 2][j - arr[i - 1]] OR T[(i + 1) % 2][j]

Else

T[i % 2][j] = T[(i + 1) % 2][j]

End Else If

End For

End For

Return T[n % 2][sum]

For loop through T starting at T[n % 2][sum] working backwards

Go up one row

If value is true

Go up one row

Else

Go up one row and go back sum amount of columns

End Else If

End For

End Function

Problem 5:

1. T[n][W] is the best value for a combined weight of up to W
2. DP relation: T[i][w] = max(val[i-1] + T[i-1][w-W[i-1]], T[i-1][w]) or T[i][w] = T[i-1][w]
3. T[0][0] = 0
4. Return T[n][W]
5. Pseuodocode:

suitecasePacking(W, weight[], val[], n)

Set i = 0 and w = 0

Initialize T[n+1][w+1]

For i <= n

For w <= W

If I = 0 or w = 0

T[i][w] = 0

Else If W[i-1] <= w

T[i][w] = max(val[i-1] + T[i-1][w-W[i-1]], T[i-1][w])

Else

T[i][w] = T[i-1][w]

End Else If

End For

End For

Return T[n][W]

End Function

Problem 6:

1. T[I, j] = LCS of X1..Xi and Y1…Yj
2. If X[i] = Y[j] set T[i,j] = 1 + T[i-1][j-1] else T[i,j] = 0
3. Initialize T[0,j] = 0 for all of j and T[i,0] = 0 for all of i
4. T[LCS(x), LCS(y)]
5. Pseudocode:

LongestConsecutiveSequence(X, Y)

Create array T where rows are len(X) + 1 and columns are len(Y) + 1

Set max\_val = 0, pos\_i = 0, pos\_j = 0

For i = 0 to len(X)

T[i, 0] = 0

For i = 0 to len(Y)

T[0, i] = 0

For i = 1 to len(X)

For j = 1 to len(Y)

If X[i] = Y[j]

Set T[i, j] = 1 + T[i - 1][j - 1]

//check to see if we found a larger consecutive substring than previous

If T[i, j] > max value

max value = T[i, j]

pos\_i = i

pos\_j = j

End If

Else

T[i, j] = 0

End Else If

// return the largest consecutive substring found

For i = max\_val – 1 to 0

Ret\_val[i] = T[pos\_i, pos\_j]

Subtract pos\_i by 1

Subtract pos\_j by 1

End For

Return ret\_val

End Function