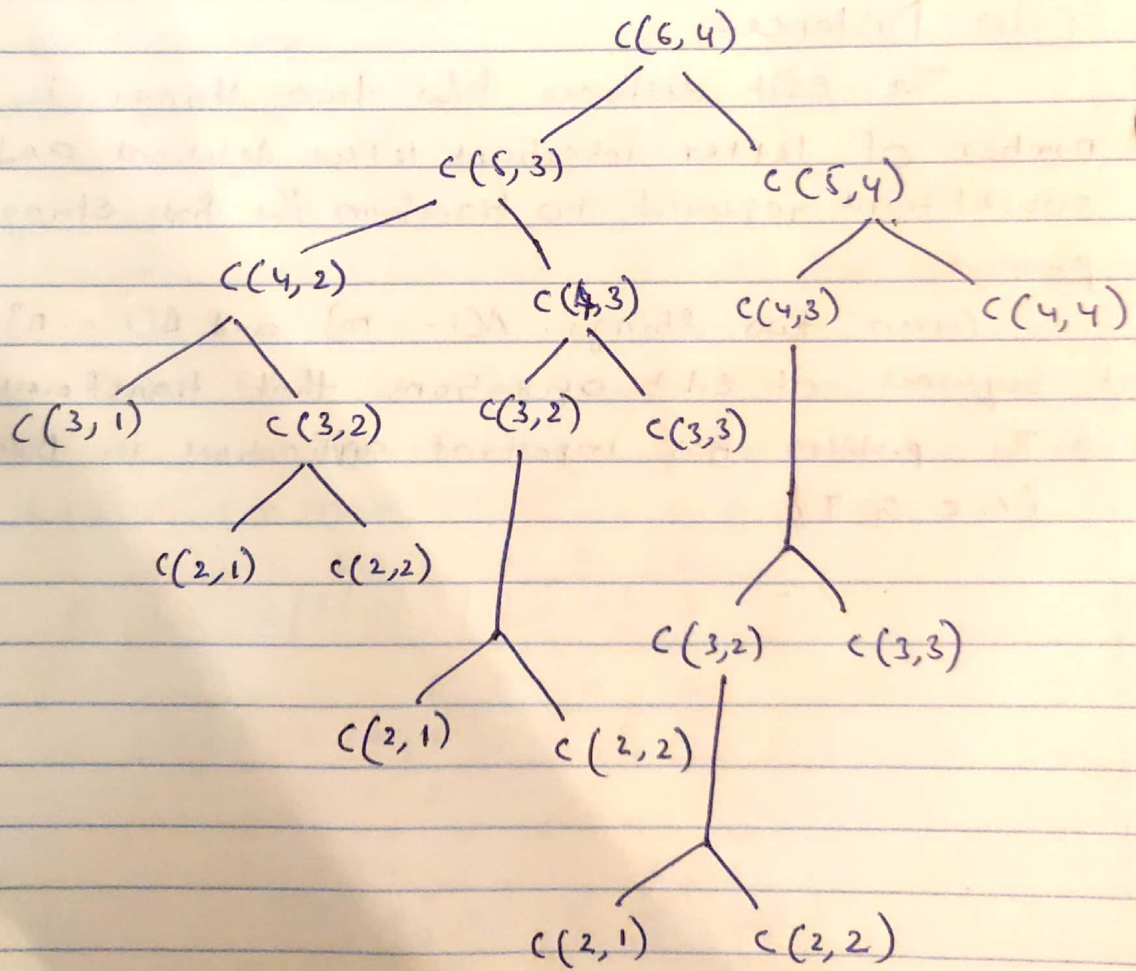
Assignment 3

1. a) Recursive Relation:

$$C[i,j] = \begin{cases} 1 & \text{if } j = i \text{ or } j = 1 \\ C[i-1,j] + C[i-1,j-1] & \text{otherwise.} \end{cases}$$

b) Compute pascal (i, j)
1. if ((i = j) or (j = 1)) return 1
2. return (Compute pascal (i-1, j) + Compute pascal (i-1, j-1))



we can see from the diagram that the algorithm is performing overlapping computations.

c) The pascal triangle can also be computed in a bottom up fashion using dynamic programming.

Consider a table $C and fill it out starting at the first element using the recursive definition.

```
Compute pascal (C)

for i = 1 to n do
    for j = 1 to i do
        if (j = i) or (j = 1) then
            C[i, j] = 1
        else C[i, j] = C[i-1, j] + C[i-1, j-1]
```

The running time of the algorithm is $O(n^2)$

2) a) let's consider the target bill as 8, the given denominations are {1, 4, 6}

According to the greedy algorithm, first we can use a 6 bill followed by 1 bill and another 1 bill which is {6, 1, 1}. But the target can be achieved by using {4, 4} which will consume only 2 bills which is minimum. Hence, the greedy algorithm will not work for our example.

b) Algorithm.

first we can consider the base case as $n = 0$ we can return 0 and if it is not zero then we can recursively loop the algorithm to substract from the given ~~Algorit~~ Denominations until we get to the base case. and store the minimum in the memory

c) We can use the same recursive algorithm in the dynamic programming bottom-up approach. We use a one dimensional array S to store the solutions to all the sub problems

Minimum Bills (n)

1. Initialize a table $T[0 \cdots n]$ and let $S[0] = 0$
2. for $i = 1$ to n do
   $T[i] = 1 + \min\{T[i-d_1], \cdots T[n-d_k]\}$;
3. return $T[n]$

The running time is $O(kn)$.

3) Algorithm

- Initialize an auxilary array of size n ~~and~~
  Assign the first element of array to A[0]
  for i ← 1 to n-1
    current_element ⇐ Sum[i-1]
    if Sum[i] < 0
      Sum[i] ← A[i]
    else
      Sum[i] ← A[i+1] + sum[i-1]
  return Max(Sum).

4) a) LCS Algorithm

Given two sequences $A[0 \cdots m-1]$ & $B[0 \cdots n-1]$ of length m & n respectively.

Consider $L(A_{m-1}, B_{n-1})$ be the length of LCS of A&B. The recursive definition for the algorithm is

$$L(A_{m-1}, B_{n-1}) = \begin{cases} 1 + L(A_{m-2}, B_{m-2}) & \text{if } A[m-1] == B[n-1] \\ \max\left(L(A_{m-2}, B_{n-1}), L(A_{m-1}, B_{n-2})\right) & \text{if } A[m-1] \neq B[n-1] \end{cases}$$

we can use a bottom up approach to define the Dynamic Programming Approach.

LCS Algorithm (A, B)

Initialize a table $L[0 \ldots m][0 \ldots n]$

```
For i ← 0 to m
    for j ← 0 to n
        if (i == 0 or j == 0)
            L[i][j] ← 0
        Else if (A[i-1] == B[i-1])
            L[i][j] ← 1 + L[i-1][j-1]
        else
            L[i][j] = Max (L[i-1][j], L[i][j-1])
return L[m][n]
```

The Time Complexity is $O(mn)$

b)

Consider $A[1 \ldots n]$ be the given array and $B[1 \ldots n]$ be the reverse of A

we can calculate the length of the common subsequence of A and B. aa let it be X. Therefore

$$X \leftarrow LCS(A, B)$$

Now we can use the same algorithm which we have used above for both the arrays and the time taken will be $O(n^2)$

So, the algorithm will also work in the time of $O(n^2)$