a) Longest Common Subsequence

LONGEST COMMON SUBSEQUENCE PROBLEM

- Given two sequences, the problem is finding a common subsequence of all subsequences that is of maximal length.
- The longest common subsequence found should be in the same order as in the given two sequences
- Subsequences are not required to occupy consecutive positions within the original sequence.

Algorithm

```
function LCSLength( X[1..m], Y[1..n])
  C = array(0..m, 0..n)
  for i := 0..m
    C[i,0] = 0
  for j := 0..n
    C[0,j] = 0
  for i := 1..m
    for j := 1..n
      if X[i] = Y[j] //i-1 and j-1 if reading X & Y from zero
        C[i,j] := C[i-1,j-1] + 1
      else
        C[i,j] := max(C[i,j-1], C[i-1,j])
  return C[m,n]
```

For Longest Common Subsequence, X: BDCCBA and Y: ABCBDCB

SOLUTION:

X/Y		Θ	A	B	C	B	D	C	B
		0	1	2	3	4	5	6	7
Θ	0	0	0	0	0	0	0	0	0
B	1	0	0	1	1	1	1	1	1
D	2	0	0	1	1	1	2	2	2
C	3	0	0	1	2	2	2	3	3
C	4	0	0	1	2	2	2	3	3
B	5	0	0	1	2	3	3	3	4
A	6	0	1	1	2	3	3	3	4

The longest common subsequence is B D C B

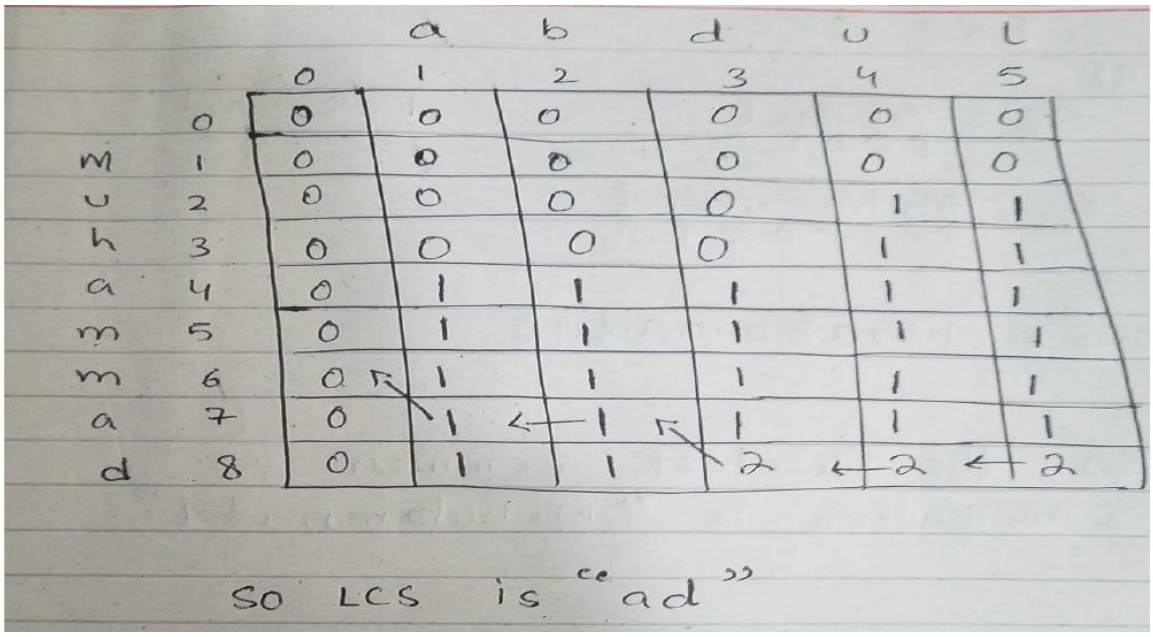
b) Shortest common sub sequence

Example Problem

S1 = “abdul”

S2 = “muhammad”

Now we first need to find the LCS of both strings by dynamic programming approach.



Now for the shortest common supersequence we need to find the following;

1) Length of shortest common supersequnce

Length of SCS =(length of S1 + length of S2) - Length of LCS

=(5+8) - 2

= 11

So in our example the length of SCS will be **11**

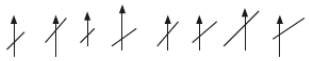
Find SCS :

Note:

1. Add LCS characters only once .
2. Assume first common character belongs to LCS character.
3. Add non LCS characters in order of either S1 then S2 or vice versa.

S1= a b d u l

S2 = m u h a m m a d



LCS = a d

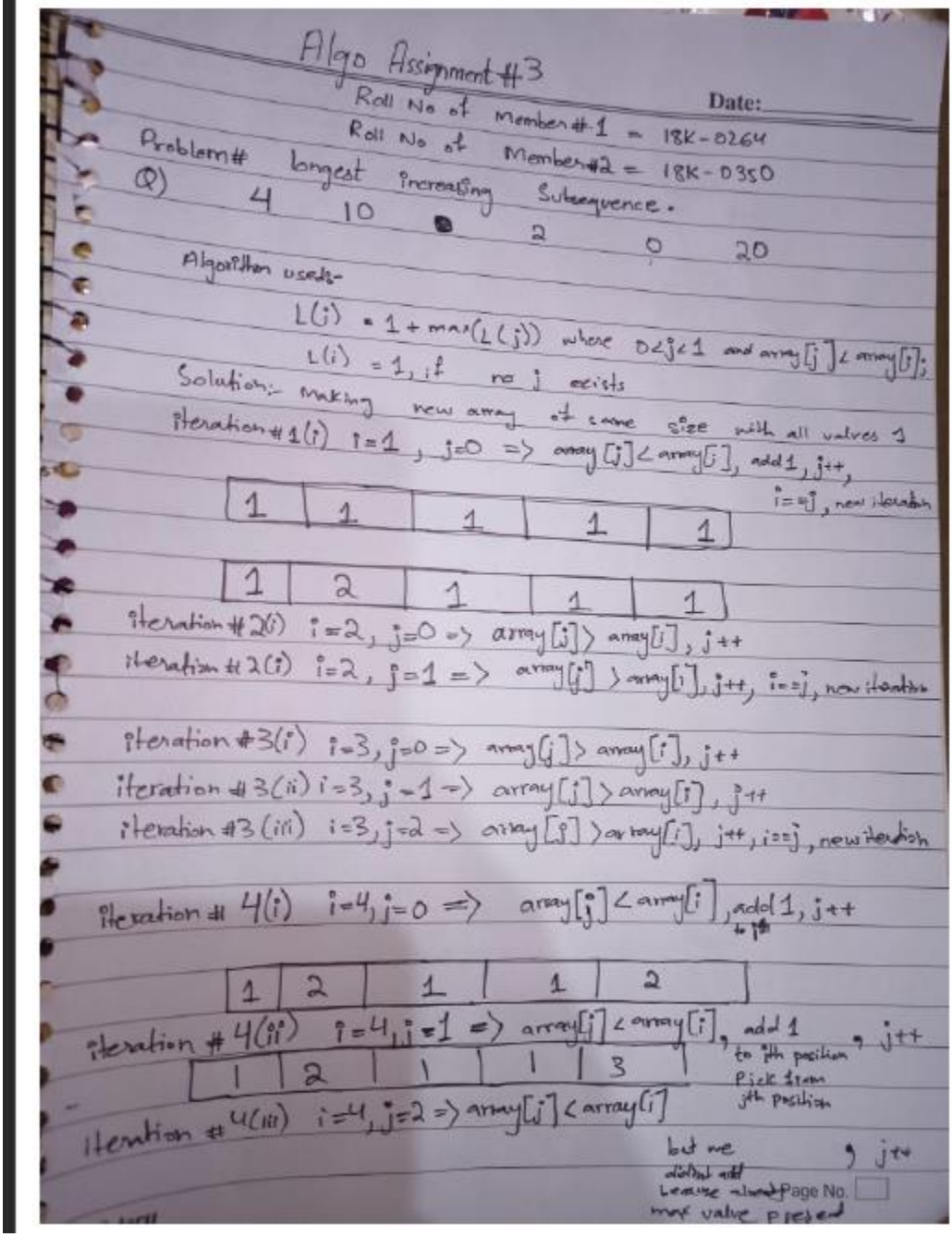
SCS = muhabmmadul

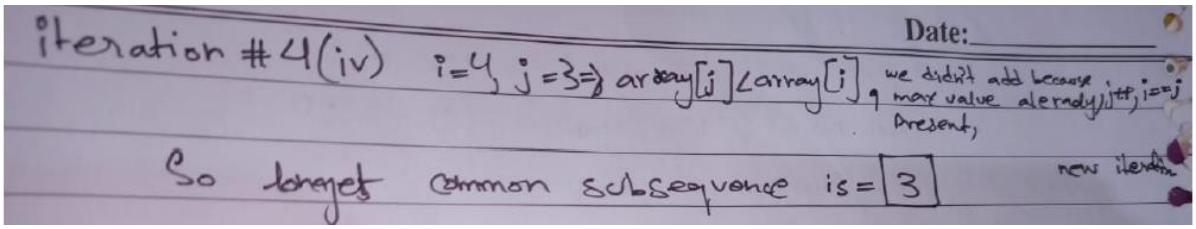
So “muhabmmadul” is the SCS in our example of length **11**

Algorithm

```
int p1=0,p2=0;
       for(char c: lcs)
           while(str1[p1]!=c)  //Add all non-LCS chars from str1
               ans += str1[p1++];
           while(str2[p2]!=c)  //Add all non-LCS chars from str2
               ans += str2[p2++];
           ans += c;   //Add LCS-char and increment both ptrs
           ++p1
           ++p2
Return ans
```

c) Longest Increasing subsequence





d) Levenshtein-Distance

The minimum edit distance between two strings is the minimum number of editing operations:

- Insertion
- Deletion
- Substitution

Needed to transform one into the other.

Examples include Spell correction, computational biology etc.

Normally, each operation has cost of 1. However, in Levenshtein, the substitution cost can be taken as 2.

For two strings X of length n Y of length m. We define D(i,j) the edit distance between X[1..i] and Y[1..j] i.e., the first i characters of X and the first j characters of Y. The edit distance between X and Y is thus D(n,m).

Algorithm

//Initialization
D(i,0) = i
D(0,j) = j
//Recurrence Relation:
For each i = 1…n
    For each j = 1…m
        D(i,j)= min(
            D(i-1, j) + 1,
            D(i, j-1) + 1,
            D(i-1, j-1) + 2, if X(i) ≠ Y(j)
            D(i-1, j-1) + 0   if X(i) = Y(j))
//Termination:
D(n,m) is distance

N	9	8	9	10	11	12	11	10	9	**8**
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
Ø	0	1	2	3	4	5	6	7	8	9
	Ø	E	X	E	C	U	T	I	O	N

The Levenshtein distance (edit distance) between the strings “INTENTION” and “EXECUTION” is 8, while considering the substation cost as 2. Otherwise, the distance is 5.

E) Matrix Chain Multiplication

Input: p[] = {40, 20, 30, 10, 30}
Output: 26000
There are 4 matrices of dimensions 40x20, 20x30, 30x10 and 10x30.
Let the input 4 matrices be A, B, C and D. The minimum number of
multiplications are obtained by putting parenthesis in following way
(A(BC))D --> 20*30*10 + 40*20*10 + 40*10*30

Input: p[] = {10, 20, 30, 40, 30}
Output: 30000
There are 4 matrices of dimensions 10x20, 20x30, 30x40 and 40x30.
Let the input 4 matrices be A, B, C and D. The minimum number of
multiplications are obtained by putting parenthesis in following way
((AB)C)D --> 10*20*30 + 10*30*40 + 10*40*30

Input: p[] = {10, 20, 30}
Output: 6000
There are only two matrices of dimensions 10x20 and 20x30. So there
is only one way to multiply the matrices, cost of which is 10*20*30

```
#include <bits/stdc++.h>
using namespace std;

// Matrix Ai has dimension p[i-1] x p[i]
// for i = 1..n
int MatrixChainOrder(int p[], int i, int j)
{
    if (i == j)
        return 0;
    int k;
    int min = INT_MAX;
    int count;

    // place parenthesis at different places
    // between first and last matrix, recursively
    // calculate count of multiplications for
    // each parenthesis placement and return the
    // minimum count
    for (k = i; k < j; k++)
    {
        count = MatrixChainOrder(p, i, k)
                + MatrixChainOrder(p, k + 1, j)
                + p[i - 1] * p[k] * p[j];

        if (count < min)
            min = count;
    }
```

```
    // Return minimum count
    return min;
}

// Driver Code
int main()
{
    int arr[] = { 1, 2, 3, 4, 3 };
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Minimum number of multiplications is "
         << MatrixChainOrder(arr, 1, n - 1);
}
```

f) Knapsack Problem

Given a set 'S' of 'n' items,

such that each item i has a positive value vi and positive weight w_i

The goal is to find maximum-benefit subset that does not exceed the given weight W.

Algorithm

```
Knapsack(j,w)
  for i ← 0 to n
          M[i,0] ← 0
 for w ← 0 to W
          M[0,w] ← 0

 for j ← 1 to n
          for w ← 0 to W
                    if wj > w
                         M[j,w] = M[j – 1, w))
                    else M[j,w] ← MAX(vj + M[j – 1, w – wj],
M[j – 1, w])
return M[n,W]
```

Problem Question:

Value = [2,3,1,4]

Weight = [3,4,6,5]

W = 8

Solution:

			Weight								
vi	wi	Index	0	1	2	3	4	5	6	7	8
1	6	3	0	0	0	2	3	4	4	5	6
4	5	4	0	0	0	2	3	4	4	5	6
3	4	2	0	0	0	2	3	3	3	5	5
2	3	1	0	0	0	2	2	2	2	3	2
		0	0	0	0	0	0	0	0	0	0

The maximum benefit/value = 6

The selected items' indexes are: s_i = [1,0,0,1]

g) Partition Problem

Given a set of positive integers, check if it can be divided into two subsets with equal sum.

First. The sum of all elements in the set is calculated. If sum is odd, we can't divide the array into two sets. If sum is even, check if a subset with sum/2 exists or not.

For given example in the question:

Case 1:

S = {1, 8, 13, 23, 1, 17}.

The sum is 63, we can't divide the array into two sets.

Case 2: (last number of Set is modified)

Let S = {1, 8, 13, 23, 1, 18}, the sum is 64, now we can check if a subset with sum/2 exists or not. The result showed that set can be partitioned into S1 = {1, 8, 23}, and S2 = {13, 1, 18}, with each subset having a sum of 32.

Case 3: (3^{rd} number of Set is modified)

Let S = {1, 8, 13, 23, 1, 16}, the sum is 62, now we can check if a subset with sum/2 exists or not. The result showed that set can be partitioned into S1 = {8, 23}, and S2 = {1,13, 1, 16}, with each subset having a sum of 31. However, note that the size of sets are not equal.

Case 3: (3^{rd} number of Set is modified)

Let S = {1, 8, 13, 32, 1, 17}, the sum is 82, now we can check if a subset with sum/2 exists or not. The result showed that set cannot be partitioned.

For code program algorithm, refer to the website referred in question.

h) Rod Cutting Problem

Rod cutting problem is a type of allocation problem. Allocation problem involves the distribution of resources among the competing alternatives in order to minimize the total costs or maximize total return (profit).

In the rod cutting problem we have given a rod of length n, and an array that contains the prices of all the pieces smaller than n, determine the maximum profit you could obtain from cutting up the rod and selling its pieces.

Length[] = {1,2,3,4,5,6,7,8}

price[] = {1,5,8,9,10,17,17,20}

Rod Length: 6

Price (i)	Length (i)	0	1	2	3	4	5	6
1	1	0	1	2	3	4	5	6
5	2	0	1	5	6	10	11	15
8	3	0	1	5	8	10	13	16
6	4	0	1	5	8	10	13	16
10	5	0	1	5	8	10	13	16
17	6	0	1	5	8	10	13	17
17	7	0	1	5	8	10	13	17
20	8	0	1	5	8	10	13	17

Maximum Profit = 17

Selected Pieces = One piece of Rod Length 6, to get maximum profit for rod length 6.

i) Coin Change Making Problem

Coin Change Making Problem

Overview:

The **change-making problem** addresses the question of finding the minimum number of coins (of certain denominations) that add up to a given amount of money. It is special case of the integer knapsack problem.

Problem Statement:

Given an array of integers that represent coins called coins and an integer amount return the minimum number of coins it requires to make complete change for amount.

Here, S = (1,3,5,7), Desired Change is 38.

Dynamic Programming Approach:

In Dynamic programming we store previously calculated values for quick retrievals and using those smaller values to calculate large values. So, we can use an array in which we can store the minimum of change require for change from 0 to 38 in this case.

Initially we can fill array with 39 in this problem assuming all coins are 1 or greater than 1 or else we can fill it with ∞.

Now we will look at each coin separately or in our case 1. How many 1-coins are needed to make amount x. Here for 0 we will take 0 coins as there are no negative coins.

For 1

1: 1 coin

For 2

1+1 = 2: 2 coins

For 3

1+1+1 = 3: 3 coins

Note: At every step we are comparing the value we calculated and value previously present in array like min(1+1+1,∞).

coin	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Continue...

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38

Now we will use both 1 and 3

We will use previous values till 2 as we can't make 1 or 2 from a 3-coin now:

For 3

3: 1 coin

For 4

So here we have multiple options as we can use combination of both coins either we can use

1+1+1+1= 4: 4coins

1+3 = 4: 2coins

Min(4,1+1) so we will chose 2

For 6

1+1+1+1+1+1 = 6: 6coins

As we are using Dynamic programming so we know that we can make six by adding another 3-coin over 3

So Min(6,1+1) therefore we will chose 2

For 7

1+1+1+1+1+1+1= 7: 7coins

Similarly here we can add 1-coin over 6 to make 7

So Min(7,1+2) therefore will use 3

By now we can identify a pattern which is:

Min(array[current_amount],1+array[current_amount-current_coin])

By using this method, the final array will be:

coin	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
7	0	1	2	1	2	1	2	1	2	3	2	3	2	3	2	3	4	3	4

continue

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
3	4	3	4	5	4	5	4	5	4	5	6	5	6	5	6	5	6	7	6

Therefore, we need total 6 coins for the **amount 38.**

Time Complexity of this method is O(n*m)

Space complexity is O(m)

n is no of coins

m is amount.

Now we have figured out the number of coins we need but we are unable to find which coins do we need. For that we can also try a tabular approach (2d array). All the operations are same except when we are looking at current amount we will be looking at one column above

and when we are [current_amount-current_coin] we can look further [current_row_index-current_coin] to put it in simple terms:

If(coin[i]>j) { table[i][j] = table[i-1][j] }

else{ table[i][j] = min(table[i-1][j],1+table[i][j-coin[i]]) }

Solution:

rows: I column: j

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
3	0	1	2	1	2	3	2	3	4	3	4	5	4	5	6	5	6	7	6
5	0	1	2	1	2	1	2	3	2	3	2	3	4	3	4	3	4	5	4
7	0	1	2	1	2	1	2	1	2	3	2	3	2	3	2	3	4	3	4

Continue...

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
7	8	7	8	9	8	9	10	9	10	11	10	11	12	11	12	13	12	13	14
5	4	5	6	5	6	5	6	7	6	7	6	7	8	7	8	7	8	9	8
3	4	3	4	5	4	5	4	5	4	5	6	5	6	5	6	5	6	7	6

Here the minimum coins used are 6.

Coins {7,7,7,7,5,5}

Here the time complexity is same but space complexity is O(m*n)

Note: Kindly refer to excel file for better understanding of tabular solution.

j) Word Break

PROBLEM:
For Word Break Problem, S = { i, like, sam, sung, samsung, mobile, ice,
cream, icecream, man, go, mango},

INTRODUCTION:
Given an input string and a dictionary of words, find out if the input string can be segmented into a space-separated sequence of dictionary words.
We will try to search from the left of the string to find a valid word when a valid word is found, we will search for words in the next part of that string

INPUT:
ilikemobile

OUTPUT:
i like mobile
F T

SOLUTION:

	0	1	2	3	4	5	6	7	8	9	10
0	T0	F	F	F	T0	F	F	F	F	F	T0
1		F	F	F	T4	F	F	F	F	F	T4
2			T2	F	F	F	F	F	F	F	F
3				F	F	F	F	F	F	F	F
4					F	F	F	F	F	F	F
5						F	F	F	F	F	T10
6							F	F	F	F	F
7								F	F	F	F
8									T8	F	F
9										F	F
10											F

T0 => I , T4=> like , T10=>mobile