(1) first part

Dynamic Programming

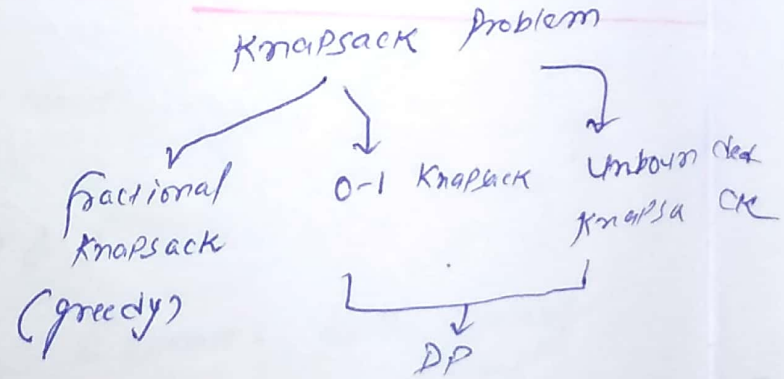
- i) Choice [Recursion call thro terms]
- ii) Optimal (max^m, min^m, ...)

Recursive
- Memoziation
- Top-down

1) 0-1 Knapsack (6) → Only one no. of item.
2) Unbounded Knapsack (5) → item number not limited
3) Fibonanci (3)
4) LCS (15)
5) LIS (10)
6) Kadane's Algorithm (6)
7) matrix chain multiplication (7)
8) DP on trees (4)
9) DP on grid (14)
10) Others (5)

0-1 Knapsack Problem

- i) Subset Sum
- ii) Equal sum partition
- iii) Coun of Subset sum
- iv) Minm Subset Sum diff
- v) Target Sum
- vi) No Of Subset Sum given difference



input wt[] :-

1	3	4	5

val[] :-

1	4	5	7

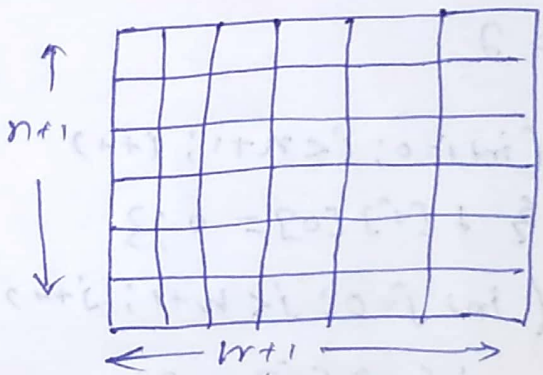
O/P → max^m profit

W = 7

```
int knapsack ( int wt[], int val[], int W, int n)
{   if (n==0 || W==0)
        return 0;
    if ( wt[n-1] <= W)
        { return max (val[n-1] + knapsack (wt[], val[],
                                  W-wt[n-1], n-1)
                      knapsack (wt[], val[], W, n-1); }
    Else
        return knapsack (wt[], val[], W, n-1);
}
```

Memoization :-

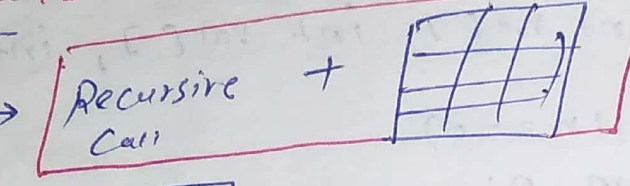
```
int t[n+1][W+1];
memset (t, -1, sizeof(t));
```

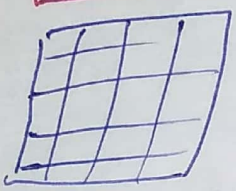



```
int knapsack (int wt[], int val[], int W, int n)
{   if (n==0 || W==0)
        return 0;
    if (t[n][W] != -1)
        return t[n][W];
    if (wt[n-1] <= W)
        return t[n][W] = max (val[n-1] + knapsack (wt, val,
                                  W-wt[n-1], n-1),
                              knapsack (wt, val, W, n-1));
    else return t[n][W] = knapsack (wt, val, W, n-1);
}
```

memoization demerits :- full stack size by recursive call.

Top-down approach :-

Memoized (dp) → Recursive Call +  → stack overflow

top-down →  only matrix.

↓

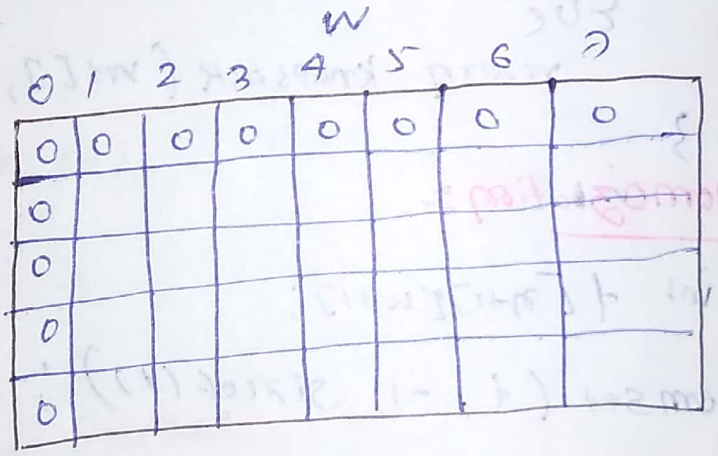
Best step-1 initialization

step-2 iterative function

wt[] = [• 1 3 4 5] → n = 4

val[] = [1 4 5 7]

W = 7




```
for(int i=0; i<n+1; i++)
  { t[i][0] = 0; }
for(int j=0; j<W+1; j++)
    t[0][j] = 0;
  for(int i=1; i<n+1; i++)
    for(int j=1; j<W+1; j++)
    { if(wt[i-1] <= j)
        t[i][j] = max(val[i-1] + t[i-1][j-wt[i-1]]
                      , t[i-1][j]);
      else
      t[i][j] = t[i-1][j];
    }
```

return t[n][W];

Subset Sum Problem

arr[] : 2 3 7 8 10] Present or Not

Sum = 11

Does any subset exist in given array so that its Sum is 11.

t[n+1][W+1] = t[6][12]

	0	1	2	3	4	5	6	7	8	9	10	11
0	T	F	F	F	F	F	F	F	F	F	F	F
1	T											
2	T											
3	T											
4	T											
5	T											

arr size ↓ (rows) — Sum → (columns) — Empty subset (column 0)

```
for (int i=0; i<n+1; i++)
    t[i][0] = True;
for (int j=1; j<W+1; j++)
        t[0][j] = false;

for ( int i=1; i<n+1; i++)
 for ( int j=1; j<W+1; j++)
  { if (arr[i-1]<=j)
       t[i][j]= t[i-1][j-arr[i-1]] || t[i-1][j];
   Else
       t[i][j]= t[i-1][j];
  }
     Return t[n][Sum];
```

Equal Sum Partition

arr[] = {1, 5, 11, 5}

O/p : True / False

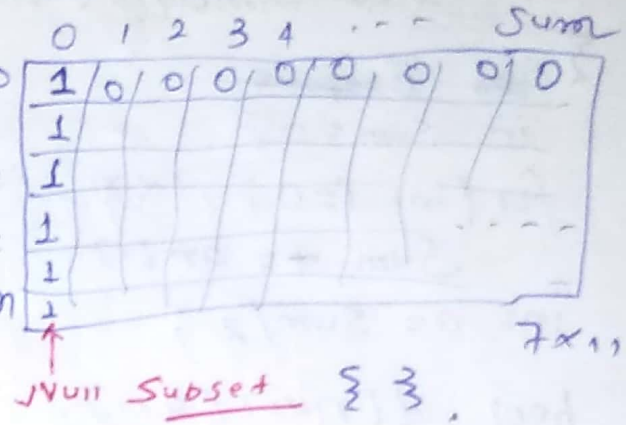
```
boolean SubsetSum (int arr[], int sum, int n)
{ bool t[n+1][sum+1];
  for(int i=0; i<n+1; i++)
      t[i][0] = true;
  for(int j=1; j<sum+1; j++)
      t[0][j] = false;
  for(int i=1; i<n+1; i++)
  for(int j=1; j<sum+1; j++)
    { if(arr[i-1]<=j)
        t[i][j] = t[i-1][j] || t[i-1][j-arr[i-1]];
      Else
        t[i][j] = t[i-1][j];
    }
return t[n][sum];
}
boolean CaualSumpartition (int arr[], int n)
  { int Sum = 0;
    for(int i=0; i<n; i++)
      Sum = Sum + arr[i];
    if (Sum % 2 == 0)
    return SubsetSum (arr, sum/2, n);
    Else return false; }
```

Count of Subset Sum of given Sum

input :- arr[] 2 3 5 6 8 10

Sum : 10

t[n+1][sum+1] = t[7][11]




```
int count(int arr[], int sum, int n)
{ int t[n+1][sum+1];
for(int i=0; i<n+1; i++)
    t[i][0] = 1;
for(int j=1; j<sum+1; j++)
    t[0][j] = 0;
for(int i=1; i<n+1; i++)
 for(int j=1; j<sum+1 ; j++)
  { if(arr[i-1] <= j)
      t[i][j] = t[i-1][j] + t[i-1][j-arr[i-1]];
    else
      t[i][j] = t[i-1][j];
  }
 return t[n][sum];
}
```

Minimum Subset Sum difference

arr[] : 1 6 11 5 ⇒ {1+6+5} {11}

12 − 11 = 1

OP :- 5

P_1 { 3 } S_1 P_2 { 3 } S_2

$S_1 - S_2 = min$

min abs($S_1 - S_2$)

we have to find position of array into two so that it sum ~~of~~ difference become minimum.

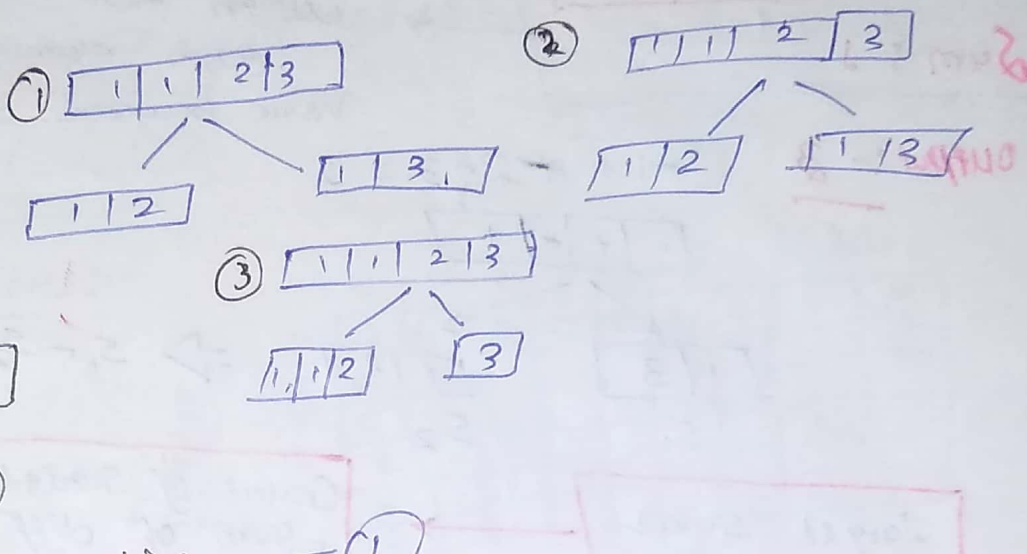
```
int SubsetSumdiff (int arr[], int n)
{
  int Sum = 0;
  for (int i=0; i<n; i++)
     Sum += arr[i];
  int a = Sum/2;
  bool t[n+1][a+1];
  for (int i=0; i<n+1; i++)
        t[i][0] = true;
  for (int j=1; j<a+1; j++)
        t[0][j] = false;
  for (int i=1; i<n+1; i++)
   for (int j=1; j<a+1; j++)
    { if (arr[i-1] <= j)
        t[i][j] = t[i-1][j] || t[i-1][j-arr[i-1]];
      else
        t[i][j] = t[i-1][j];
    }
int result;
 for (int i=a; i>=0; i--)
    if (t[n][i] == true)
       { result = i;
         break;
       }
return Sum - 2*result;
}
```

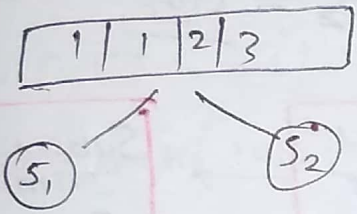
Count the number of Subset Sum given difference

Input arr[] = | 1 | 1 | 2 | 3 |

diff = 1;

Output = 3





Sum(S1) - Sum(S2) = diff — (1)

Sum(S1) + Sum(S2) = Sum(arr) — (2)

2 Sum(S1) = diff + Sum(arr)

$\Rightarrow Sum(S_1) = \frac{diff + Sum(arr)}{2}$

Count of no. of Subset Sum of given diff = Count of Subset Sum of (S_1)

```
int Sum = (diff + Sum of arr) / 2 ;
if ((diff + sum of arr) % 2) == 0
    return Count of subset sum (arr, sum);
Else return 0;
```

Target Sum :-

arr: | 1 | 1 | 2 | 3 |

Sum : 1

→ Count no of ways to assign '+' Or '-' in front of each element so that expression value become Sum value.

output :- 3 $-1+1+2+3 = 1$

| - | + | - | + |

| 1 | 3 | S_1 | | 1 | 2 | S_2

$\Rightarrow S_1 - S_2 = diff$

target sum → Count of Subset Sum of diff given → Subset sum of S_1

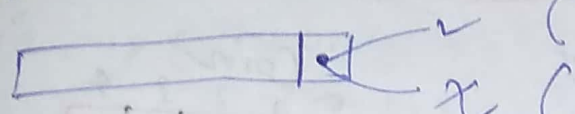
```
int sum = (diff + sum of arr) / 2 ;
if ((diff + sum of arr) / 2) == 0
    return Count of Subset sum (arr, sum);
Else return 0;
```

Unbounded Knapsack (5)

↳ Infinite no. of copy of each element.

1) Rod cutting
2) Coin change 1 (maximum no. ways)
3) Coin change 2 (minimum no. of coin used)
4) maximum ribbon cut

gn Unbounded knapsack



(can be processed later again)

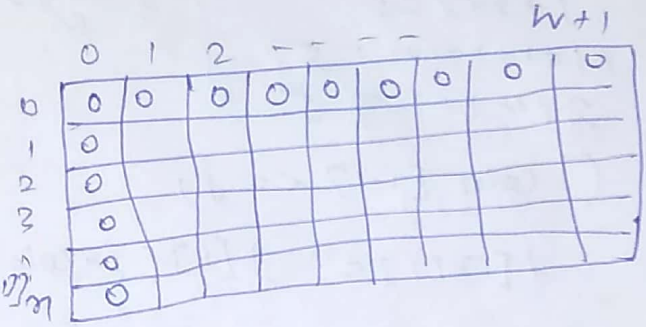
(processed)

Input wt[] :- | 1 | | | |

val[] :- | | | | |

W = _ :-

t[n+1][W+1]



Unbounded knapsack :-

```
if ( wt[i-1] <= j)
t[i][j] = max( val[i-1] + t[i][j-wt[i-1]],
               t[i-1][j] );
else
t[i][j] = t[i-1][j];
```

Rod Cutting

maximum profit by proper cutting ~~property~~ of rod.

Length[] :- | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

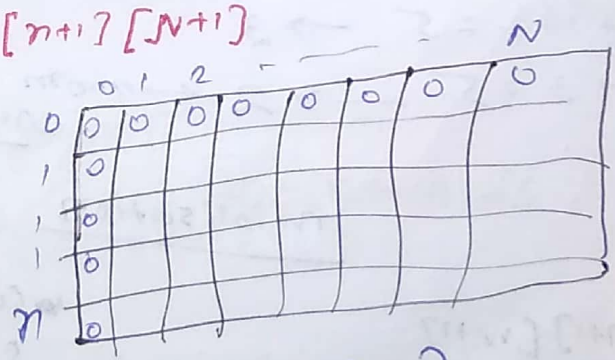
Price[] :- | 1 | 5 | 8 | 9 | 10 | 17 | 19 | 20 |

N :- 8

↓

Length of rod

t[n+1][N+1]




```
if ( Length[i-1] <= j)
t[i][j] = max( price[i-1] + t[i][j-length[i-1]], t[i-1][j] )
else
t[i][j] = t[i-1][j];
```

Coin change-1 (maxm no of ways)

Coin[]: | 1 | 2 | 3 | — infinite no. of supply

Sum :- 5

| 2 | 3 |, | 1 | 1 | 3 | } 5 ways.

1+2+2=5
1+1+1+1+1=5
2+1+1+1=5

Sum \ Coin	0	1	2	3	4
0	1	1	1	1	1
1	0				
2	0				
3	0				
4	0				
5	0				

```
if ( coin [i-1] <= j)
    t[i][j] = t[i][j-coin[i-1]] + t[i-1][j];

Else  t[i][j] = t[i-1][j];
```

Coin-change 2 (minm no. of coin used)

Coin[]: | 1 | 2 | 3 |

Sum : 5

1+1+1+1+1=5 → 5
1+1+1+2=5 → 4
1+1+3=5 → 3
1+2+2=5 → 3
2+3=5 → 2 ← minm no. of coin.

initialisation

t[n+1][w+1]

t[n+1][sum+1]

t[4][6]

Coin[]:-

Sum :-

	0	1	2	3	4	5
0	∞	∞	∞	∞	∞	∞
1	a					
2	a					
3	a					

(sum)

INT_MAX = ∞ - 1

(n)

```
for(int j=1; j< sum+1 ; j++)
  { if( j% arr[0] ==0)
        t[1][j] = j/arr[0];
    else
        t[1][j] = INT_MAX -1;
  }

for (      )
for(       )
  { if( coin[i-1] <=j)
    t[i][j] = min ( t[i][j-coin[i-1]] +1,
                    t[i-1][j] );
    else
      t[i][j] = t[i-1][j];
  }
```

Longest Common Subsequence (LCS)

1) Longest Common Substring
ii) Print Longest Common subsequence
iii) Shortest Common Supersubsequence
iv) print SCS
v) minm no. of insertion and deletion to make string a to string b.

vi) Longest repeating subsequence.
vii) Length of largest subsequence of a which is substring in b.
viii) subsequence pattern matching
ix) Count how many times a appears as subsequence in b.

x) Longest palidromic Subsequence

xi) " " Substring

xii) Count of palidromic Substring

xiii) minm no. of deletion in a string to make it a palidromic.

xiv) minm no. of insertion in string to make it a palidromic.

Longest Common Subsequence / LCS

I/P

x :- a b c d g h — n → length of string

y :- a b e d f h r — m

LCS :- a b d h

O/P :- 4

in Substring character must be continuous.

Recursion :-

```
int LCsequence(string x, string y)
    if (n==0 || m==0)
        return 0;

int LCS(string X, string Y, int n, int m)
{
    if (n==0 || m==0)
        return 0;
    if (X[n-1]==Y[m-1])
        return 1+LCS(X,Y,n-1,m-1);
    else
        return max(LCS(X,Y,n-1,m), LCS(X,Y,n,m-1));
}
```

memoization :-

```
t[n+1][m+1]
int t[n+1][m+1];
memset(t, -1, sizeof(t));  ⟶ in main
int Lcs(string x, string y, int n, int m)
{
    if (n==0 || m==0)
        return 0;
    if (t[n][m]!=-1)
        return t[n][m];
    if (x[n-1]==y[m-1])
        return t[n][m] = 1+ Lcs(x,y,n-1,m-1);
    else
        return t[n][m] = max(Lcs(x,y,n,m-1), Lcs(x,y,n-1,m));
}
```

Top down :-

```
for(int i=0; i<n+1; i++)
    t[i][0] = 0;
for(int j=1; j<m+1; j++)
    t[0][j] = 0;
for(int i=1; i<n+1; i++)
for(int j=1; j<m+1; j++)
{   if(x[i-1]==y[j-1])
        t[i][j] = 1+ t[i-1][j-1];
    else
        t[i][j] = max(t[i-1][j], t[i][j-1]);
} return t[n][m];
```

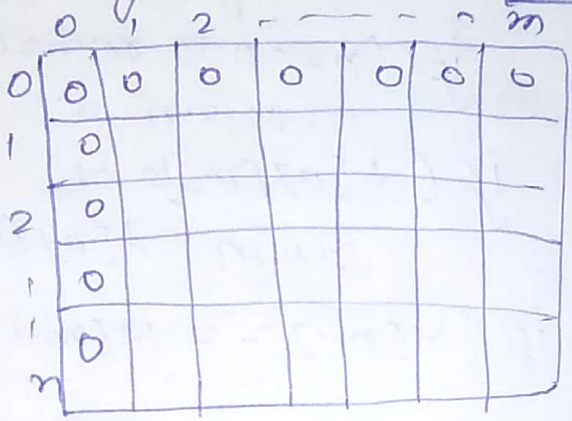
Longest Common Substring

I/P x: a b c d e → n
y: a b f c e → m

Continious Subset

O/P :- 2 (ab) ← Length of Longest Common Substring

t[n+1][m+1]




```
for(int i=1; i<n+1; i++)
  for(int j=1; j<m+1; j++)
    { if(x[i-1]==y[j-1])
        t[i][j]=1+t[i-1][j-1];
      else
        t[i][j]=0;
    }
  }
int max=INT_MIN;
for(int i=1; i<n+1; i++)
  for(int j=1; j<m+1; j++)
    if(max<t[i][j])
      max=t[i][j];

return max;
}
```

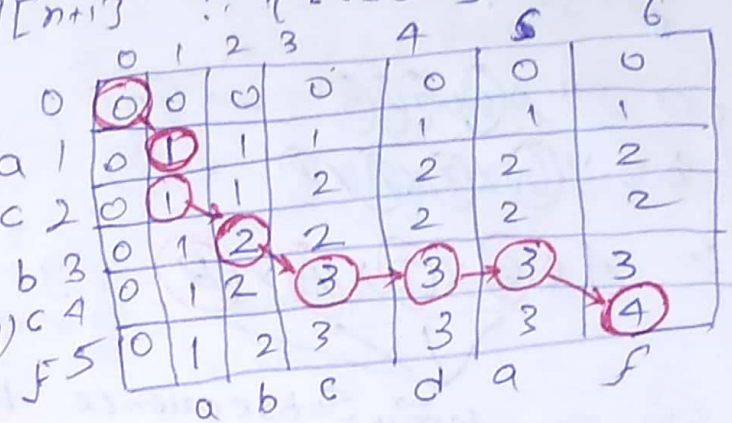
Print LCS (subsequence) b/w two strings

I/P x: a a b c f → m = 5

y: a b c d a f → n = 6

O/P :- abcf

t[m+1][n+1] :: t[6][7]




```
int i = m, j = n; String s;
while (t[i][j])
{ if (x[i-1] == y[j-1])
  { s.push_back(a[i-1]);
    i--;
    j--;
  }
  else
  { if (t[i-1][j] > t[i][j-1])
    { i--;
    }
    else
      j--;
  }
}
reverse(s.begin(), s.end());
return s;
```

Shortest Common Supersequence length

i/p a = "geek" → (m) we have to merge a and b
b = "eke" → (n) so that it contain both string.

o/p "geeke".

I/p { a = "AGGTAB" → (m)
b = "GXTXAYB" → (n)

m n LCS
↑ ↑ ↑
o/p :- AGGXTXAYB = 9 = 6+7-4 = 9

Subsequence written only once, in output
longest

So, length of shortest common supersequence
= m+n - longest subsequence length

Code :-
return (m+n) - LCS(a, b, m, n);

Minimum no. of insertion and deletion to make string a to string b

I/p :- a : heap → m = 4
b : pea → n = 3

o/p :- p e a = 3 = insertion :- n - LCS
deletion :- m - LCS
= 4+3-2(2)
= 3

Return m+n - 2 * LCS(a, b, m, n);

Longest Palidromic Subsequence (LPS)

I/P s: agbcba s' = abcbga

O/P 5

(a) g (b) (c) (b) (a)

String s';
s' = reverse (s.begin(), s.end());

s = string (s.rbegin(), s.rend());

return Lcs (s, s', s.length(), s'.length());

↓
Longest Common Subsequence.

Minm no of deltion in string to make its Palidromic.

I/P s = "agbcba" → n = 6

O/P :- 6 - 5 = 1.

s' = "abcbga"

Lcs = a b c b a → 5

Output :-

return n - LPS(s)

↓
Longest palidromic Subsequence.

Print Shortest Common Supersequence

I/P a: a c b c f → m=(5)
b: a b c d a f → n=6

O/P :- "a c b c d a f" , worst:- "a c b c f a b c d a f"
LCS :- a b c f

Similar Question print LCS (longest Common Subsequence)

	ϕ	a	b	c	d	a	f
ϕ	0	0	0	0	0	0	0
a	0	1	1	1	1	1	1
c	0	1	1	2	2	2	2
b	0	1	2	2	2	2	2
c	0	1	2	3	3	3	3
f	0	1	2	3	3	3	4

LCS t[6][7]

```
int i=m;
int j=n;
String s.
while (i>0 && j>0)
{ if (a[i-1]==b[j-1])
    { s.push_back(a[i-1]);
      i--; j--;
    }
  else { if (t[i][j-1] > t[i-1][j])
         { s.push_back(b[j-1]);
           j--;
         }
       else { s.push_back(a[i-1]);
              i--;
            }
       }
}
```

```
}
}
while ( i>0)
  { s.push_back ( a[i-1]);
      i--;
  }
while (j>0)
  { s.push_back (b[j-1]);
    j--;
  }
return string (s.begin(), s.end())
}
```

Longest Repeating Subsequence length

Str = "AABEBCDD" → (m)

o/p :- 3 (ABD)

(A) (A) (B) E (B) C (D) (D)

ABD repeat 2 times.

Str = "AABEBCDD" → (m)

Str = "AABEBCDD" → (n)

```
for (int i=1; i<m+1; i++)
  for(int j=1; j<m+1; j++)
    { if ( str[i-1] == str[j-1] && i!=j)
        { t[i][j] = 1+ t[i-1][j-1];
        }
      else
        { t[i][j] = max ( t[i-1][j], t[i][j-1]);
        }
    }
    return (t[m][m]);
```

Sequence Pattern matching

I/P a = "AXY"
b = "ADXCPY"

O/P - T/F → boolean.

(A) D (X) C P (Y) → true

lcs :- "AXY"

if length of lcs == a.length()
then return true;
else return false;

Minimum no. of ~~deletion~~ insertion in string to make it palindromic

I/P :- S = "(a)c(b)(c)(b)d(a)"

O/P :- 2.

(a)d(b)(c)(b)c(a)

longest palidromic subsequence
:- "abcba"

min^m of deletion = S.length() - length of LPS

min^m no. of insertion = min^m of deletion
= S.length() - length of LPS.

(a) c d (b) (c) (b) d c (a)
↑ insertion ↑ insertion

2nd part

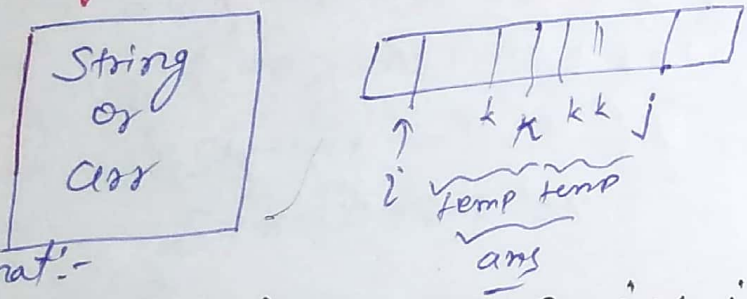
Matrix Chain multiplication

↳ Nuttanix

1. mcm
2. Printing mcm
3. Evaluate Expr to True / Boolean parenthesisation
4. min / max value of an Expr
5. Palindromic partitioning
6. Scramble String
7. Egg Dropping problem

mcm

identification :-



30, 50
10, 30

format :-

```
int Solve ( int arr[], int i, int j)
{ if (i>j)
    return 0;
  for ( int k = i; k<j ; k++)
  { // Calculate temp answer
    temp_ans = Solve (arr, i, k) + Solve (arr, k+1, j)
    ans ← fun (temp_ans);
  }
  return ans;
}
```

mcm

arr[] = { 40 20 30 10 30 }

A_1 A_2 A_3 A_4

40×20 20×30 30×10 10×30

$A_1A_2A_3A_4$ Calculate this by minm cost (less multiplication)

$A_{a\times b} B_{b\times c} = C_{a\times c}$

↓

Cost $(a\times c)\times b = acb$

multiply such that cost is minm.

A → 10 * 30

B → 30 * 5

C → 5 * 60

ABC

↓

(AB)C

Cost = 10×30×5 + 10×5×60

= 4500

A(BC)

Cost = 10×30×60 + 30×5×60

= 22000

Ex arr[]: 40 20 30 10 30

i ↑ j

A_1 → 40×20

A_2 → 20×30

A_3 → 30×10

A_4 → 10×30

A_i = arr[i-1] × arr[i]

= 40 × 20

```
int Solve ( int arr[], int i, int j)
{
    if (i >= j)
        return 0;
    for(int k = i; k <= j-1; k++)
    {
        int temp1
```

```
int min = INT_MAX;
for (int k = i; k <= j-1; k++)
{ int temp = Solve(arr, i, k) + Solve(arr, k+1, j)
                + arr[i-1] * arr[k] * arr[j];
  if (min > temp)
    min = temp;
}
return min;
}
```

memoized code bottom up

```
int t[Size+1][Size+1];
memset(t, -1, sizeof(t)); <- in main;
Solve(arr, 1, size-1) <- in main,
int solve(int arr[], int i, int j)
{ if (i == j)
    return 0;
  if (t[i][j] != -1)
    return t[i][j];
  int min = INT_MAX;
for (int k=i; k<j; k++)
  { int temp = Solve(arr, i, k) + Solve(arr, k+1, j)
                + arr[i-1] * arr[k] * arr[j];
    if (min > temp)
      min = temp;
  }
  return t[i][j] = min; }
```

Palidromic partitionaty

min^m no. of partition of string to make resultant substring palidroom.

I/p :- nitin

O/p :- 2


```
int t[size+1][size+1];
memset(t, -1, sizeof(t));
Solve(arr, 0, size-1);
```
→ in main()

```
int Solve(int arr[], int i, int j)
{ if(i>=j)
    return 0;
  if(t[i][j] != -1)
    return t[i][j];
  if(isPalindrom(arr, i, j))
    return 0;
  int min = INT_MAX;
  for(int k = i; k <= j-1; k++)
  { int temp = Solve(arr, i, k) + Solve(arr, k+1, j) + 1;
    if(min > temp)
      min = temp;
  }
  return t[i][j] = min;
}
```

Allocate minimum no. of pages

arr[]: 10 20 30 40 → number of page in each book

K: 2 → no. of students.

→ Array sorted not required.

```
#include <bits/stdc++.h>
using namespace std;
bool isValid(int arr[], int n, int k, int ans)
{ int student=1; int sum=0;
  for(int i=0; i<n; i++)
  { sum += arr[i];
    if(sum>ans)
    { student++;
      sum=arr[i];
    }
    if(student>k)
      return false;
  }
  return true;
}
int solve(int arr[], int n, int k, int m, int sum)
{ int start=m;
  int e=sum; int ans=INT_MAX;
  while(start<=e)
  { int mid=start+(e-start)/2;
```

```
    if (isValid(arr, n, k, mid))
    { ans = min(ans, mid);
      e = mid - 1;
    }
    else
    { start = mid + 1;
    }
  }
return ans;
}
int main()
{ int k, n; cin >> k >> n;
  int arr[n]; int m = INT_MIN, sum = 0;
  for (int i = 0; i < n; i++)
    { cin >> arr[i];
      m = max(m, arr[i]);
      sum += arr[i]; }
  cout << solve(arr, n, k, m, sum) << endl;
}
return 0;
}
```

longest increasing subsequence

3 4 -1 0 6 2 3 = 4

2 5 1 8 3 = 3

```
int t[n+1]; int j; t[0]=0;
for(int i=0; i<n; i++)
    t[i+1]=1;
for(int i=1; i<n; i++)
{   j=0;
    for(; j<i; j++)
    {  if(arr[i] >= arr[j])
           t[i+1] = max(t[i+1], t[j+1]+1);
    }
}
return max_element(t+1, t+n+1);
}
```