LECTURE 21

→ DYNAMIC PROGRAMMING

(1) Coin exchange problem :-

given coins of denominations

$1 = d_1 < d_2 < d_3 < \ldots\ldots < d_k$

d_k = infinite # of each type.

Otherwise, I cannot pay 'n' always.
eg: $d_1 = 2, d_2 = 5$
then 8 XX
not possible!

We need to find the minimum # of coins required to pay for 'n'.

→ greedy algorithm does not always give correct answer!

→ eg: $d_1 = 1$, $d_2 = 10$, $d_3 = 25$.
pay ₹30.
greedy → 25, 1, 1, 1, 1, 1
optimal → 10, 10, 10

→ eg: $d_1 = 1$, $d_2 = 7$, $d_3 = 10$
pay ₹14
greedy → 10, 1, 1, 1, 1
optimal → 7, 7

→ eg:- ATM m/c, dispense ₹n, then minimum # of coins required to pay 'n'?

→

- The problem was introduced in 1950 by Bellman as a method for optimizing control systems.

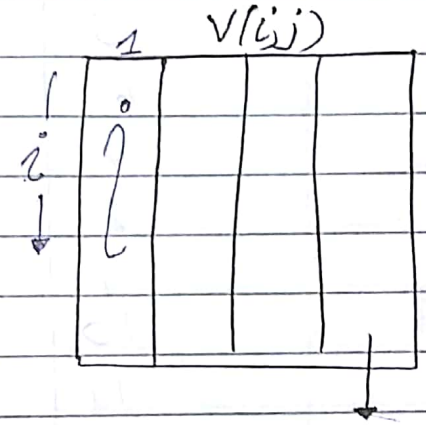
- definition:

given $d_1, d_2, d_3, \ldots d_k$
where $d_1 = 1$; and # of coins of denomination d_i are infinite

find $V(i,j)$ = min. # of coins required to pay 'i' using first 'j' coins.

amount I pay for it

# of coins allowed



$V(i,1) = i$ coins.
eg:- $V(5,1) = 5$

so,

$d_1, d_2, \ldots d_{i-1}, d_i, d_{i+1} \ldots d_k$ $V(n,k)$

looks at the coins used before

I'm here

main idea :-

j^{th} coin

pick j^{th} coin

do not pick j^{th} coin

$1 + V(i-d_j, j)$

$V(i, j-1)$

# of coins goes up by one.

same as it was with $(j-1)$ coins

So, $V(i,j) = \min\{V(i,j-1), V(i-d_j, j)+1\}$

$i \geq d_j$

eg:- $d_1 = 1,\ d_2 = 7,\ d_3 = 10$

i \ j	1	2	3
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	1	1
8	8	2	2
9	9	3	3
10	10	4	1
11	11	5	2
12	12	6	3
13	13	7	4
14	14	2	2
15	15	3	3

eg:- $V(14,2) = \min(V(14,1), 1+V(7,2))$

$V(14,2) = 1+1 = 2$

→

Code:-

```
for i = 1 to n
    V(i,0) = 0
    V(i,1) = i

for j = 1 to k
for i = 1 to n
    if (V(i,j-1) <(V(i-dj, j) +1))
        V(i,j) = V(i,j-1)
    else
        V(i,j) = 1+ V(i-dj,j)
```

} I can exchange/interchange these two loops, it does not matter.

final answer = V(n,k)

Time Complexity = O(nk)
Space Complexity = O(nk).

↳ what all coins did I use? ↓

```
i = n,  j = k
while (j > 1)
{
    if (V(i,j) == V(i,j-1))
        j = j-1;
    else {
        print("Used %d coins:", j);
        i = i-dj;
    }
}
Print("Pay by %d coins of type 1", i);
```

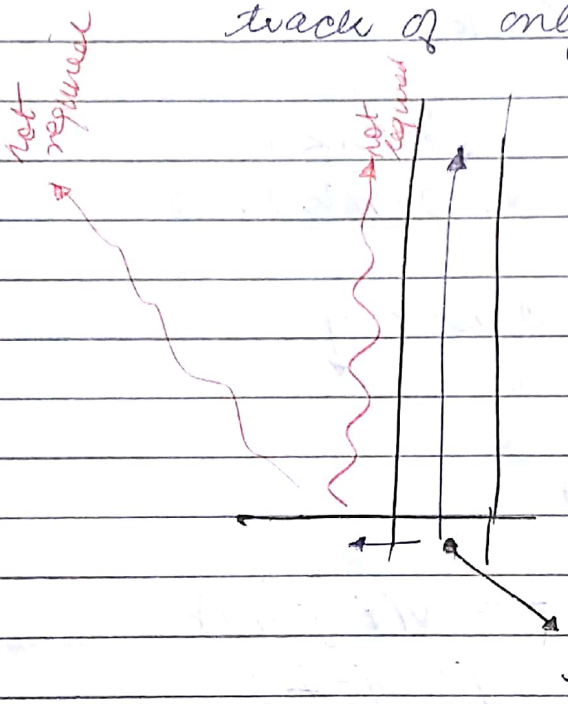
O(n)=
O(n+k)

now, $O(nk)$ time is fine, but $O(nk)$ space is not?

eg: $n = 10^6$
$k = 10^3$

↳ We don't need the 1st column; I already need the entry from ~~some~~ previous row or some entry just above current column above it;

↳ so, at any time I need to keep a track of only one + one = two columns.



So, I can manage with only one column.

$V \longrightarrow n$ - length array.

→

```
for i = 1 to n
    V[i] = 1

for j = 2 to k
{ for i = 2 to n
  {
      if (i >= dj & (V(i-dj)+1) < V(i))
          V(i) = V(i-dj)+1
      else ;
  }
}
```

Time = O(nk)
Space = O(n)

↳ Here, I need backtracking to find what all coins I need. →

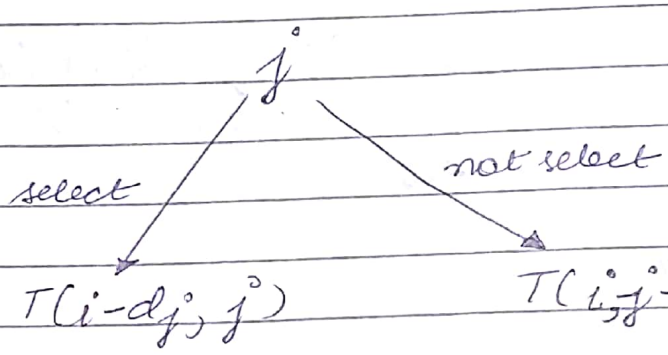
```
V(n) = minimum # of coins
i = n,   j = k

while (j > 1)
{
    if (V(i) == 1 + V(i-dj))
    { print ("jth coin used");
      i = i - dj;
    }
    else { j = j - 1; }
}
```

time complexity ~~remains~~ becomes ~~same~~ O(n+k)
but this doesnot work all the time!

↳ It is space minimization & not the time.

↳ modified problem :-
know total # of ways to pay.

$$T(0, j) = 0$$
$$T(i, 1) = 1$$



$$T(i,j) = T(i-d_j, j) + T(i, j-1)$$

eg: $d_1 = 1$, $d_2 = 4$, $d_3 = 11$
pay 14 →
10, 1, 1, 1, 1
11, 1, 1, 1
4, 4, 4, 1, 1.

↳ do I need 2nd table here or I can still manipulate 1D table?

↓

yes, I can still manipulate 1D table.

↳ Steps to approach a DP problem :-
- Characterize the structure of the optimal solution.
- Recursively define the value of optimal solution.
- Compute the value of optimal solution in a bottom up fashion.

Compute an optimal solution from computed/stored information.

eg:- fibonacci:-

```
F[0] = 0
F[1] = 1
for i = 2 to n
    fi = (fi-1 + fi-2) % 10.
```

tracks previous two values

space = Constant
time = O(n).

→ 0/1 Knapsack :- (Repetition of item not allowed)

Given n-items,

int. weights	w_1	w_2	.....	w_n
values	v_1	v_2	.....	v_n

a knapsack of integer capacity 'w'
find most valuable subset of items that fit into the knapsack.

x_i = boolean variable.

x_i → 0 (not picking i^{th} item)
→ 1 (picking i^{th} item)

now, our objective is to find max value of $\sum x_i v_i$
given that condition $\sum x_i w_i$ fits in the knapsack.

i.e. $\max (\sum x_i v_i)$ when $\sum x_i w_i < W$

this is an ILP (Integer linear Program) and an NP Hard Problem.

eg:-

item	wt	value	val/wt
1	2	12	6
2	1	10	10
3	3	21	7
4	2	15	7.5

this works for fractional knapsack using greedy approach

Consider instance defined by first 'i' items and capacity j ($j \leq W$).

Let $V[i, j]$ be optimal value of such an instance.

$V[i, j]$ = max profit value I can make using just 'i' items and a knapsack of size 'j'.

item 'i'

size of knapsack

$$V(1, j) = \begin{cases} v_1 & j \geq w_1 \\ 0 & \text{otherwise.} \end{cases}$$

final answer will be stored in $V(n, W)$

$V(i, j)$

select i^{th} item → $V(i) + V(i-1, j-w_i)$

reject i^{th} item → $V(i-1, j)$

so, $V(i,j) = \max\left(V(i-1, j),\ V(i) + V(i-1, j-w_i)\right)$

$j \geq w_i$

eg: for above example, if knapsack = 8 kg.

item \ iteration (knapsack wt)	1	2	3	4	5	6	7	8
1	0	12	12	12	12	12	12	12
2	10	12	22	22	22	22	22	22
3	10	12	22	31	33	43	43	43
4	10	15	25	31	37	46	48	58

final answer

if $(V_j == V_i + v(j-w_i))$

values don't change here, $v_{ij} = \max(v_{i-1}, j)$

$T(n) = O(nW)$

Space $= O(nW)$

↳ backtrack to find what all items I chose → $O(n+W)$

↳ Can we reduce space & time? → Yes.

$V(i, j)$ → not required actually.

↳ though, it can be done in $O(n+W)$, it is still not a good solution (this problem is

↓

W could be too huge, say 10^9.

↳ if profit = very small and size of knapsack = 10^9 = W

$n = 100$,

then space = 10^9 = W

↓

this cannot be solved!

algorithm works iff O(W) space is adjustable to RAM and Time $O(nW)$ is fine.

↳ W is large, i.e. $\sum v_i = P \leq 10^6$

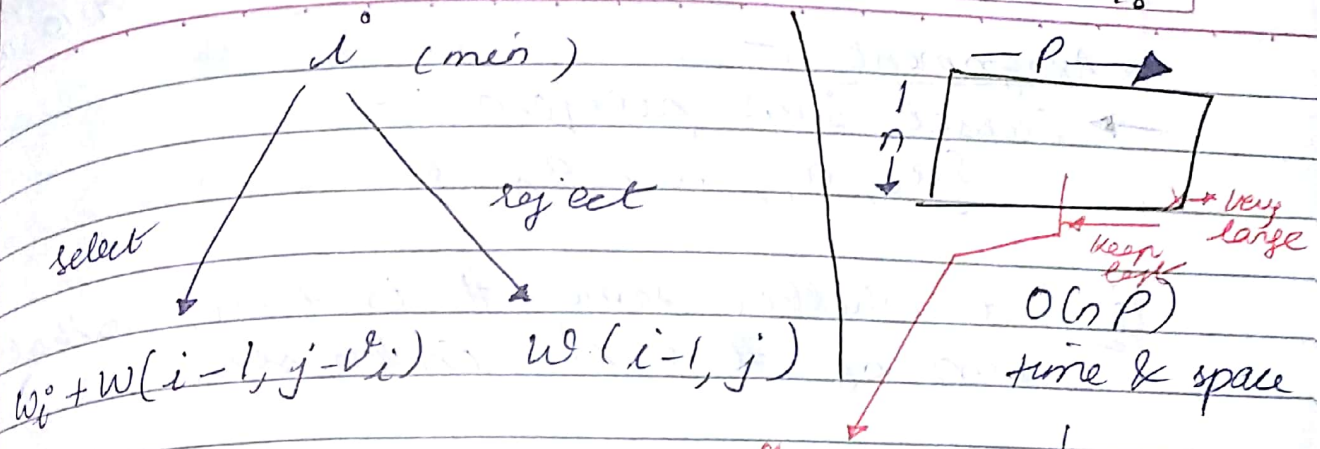
↓

then I can redefine my space

i, j ⟶ now should now represent price,

so, W(i, j) is minimum size knapsack required to make a profit of atleast 'j' using first 'i' items

$$W(1, j) = \begin{cases} w & j \leq V_1 \\ \infty & \text{otherwise} \end{cases}$$



$W(n, P) \Longrightarrow P = \sum v_i$

given — v_i / knapsack wt W
w_i

then,

$V_{i,j}$ = minimum size knapsack such that profit 'j' is achieved using first 'i' items.

max profit I make is of size W.

↳ backtrack and find items to be picked ?
O(nP) time & space.

not possible with two columns. You can only compute, not backtrack.

if (P << W) then

if (W << P) then previous solution works fine.

↳ if P = W = very large, cannot be solved in O(n) time.

→ Assignment :-

→ Subset sum problem :-

$\{a_0, a_1, \ldots, a_{n-1}\}$

(i) find whether some # is there, so that sum of # = 'x' in the set.

eg: {37, 10, 14}

find 19 → {14, 3, 2} and so on.

(ii) define x_i → 0 (not choose i^{th} value)
→ 1 (choose i^{th} value)

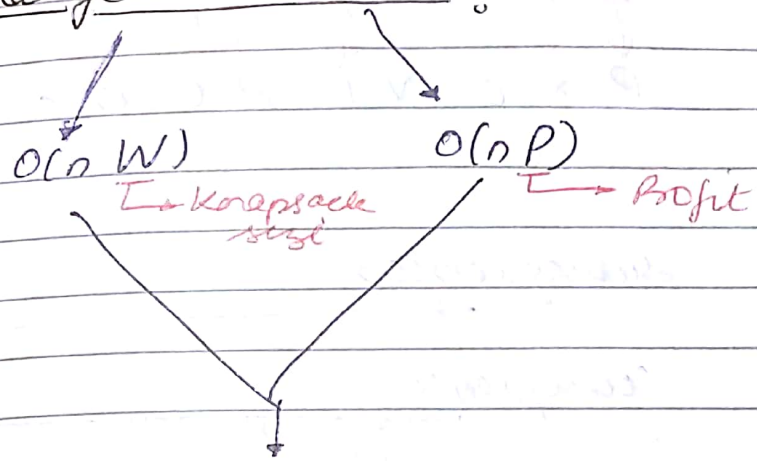
such that $\Sigma x_i w_i = x$

(iii) if Σa_i's is small, do something to reduce.

(iv) find largest y, such that $\Sigma a_i x_i = y \leq x$ (no repetition here)

LECTURE - 22

→ Coin exchange continued :-



min (these two), then we implement that part.

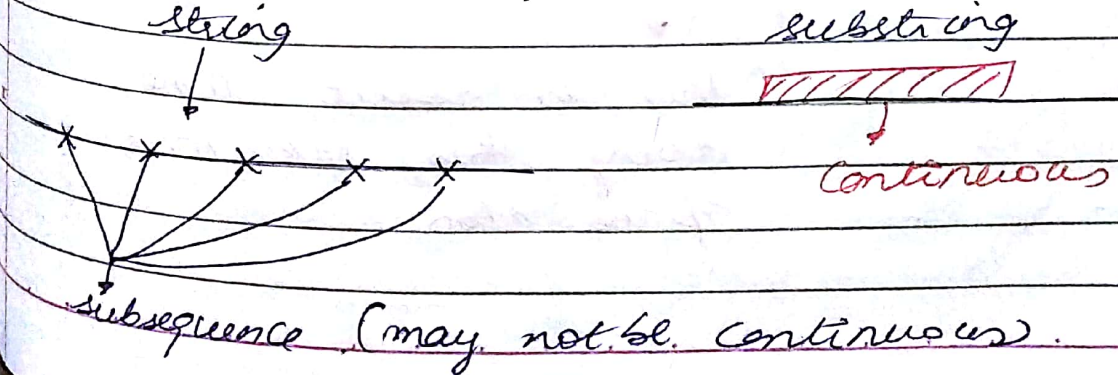
if both are very large enough, we can't search; we can only compute the length

→ LONGEST COMMON SUBSEQUENCE :- (LCS)

↳ LCS is a string 's' obtained by deleting zero or more symbols from original string 'S'.

eg :- president has subsequences → pred
sdn
predent

↳ LCS is to find max. length subsequence between two sequence.



eg:- president & providence.

P r e s i d e n t
P r o v i d e n c e

LCS is "PRIDEN"

so, subsequences → length 'n'

subsequences → m

subsequence → s_m.

→ $O(2^n)$ intersection → exponential solution

hence NPH problem

Dynamic Programming → $O(n^2)$

for string matching :- $O(n^3)$

↓

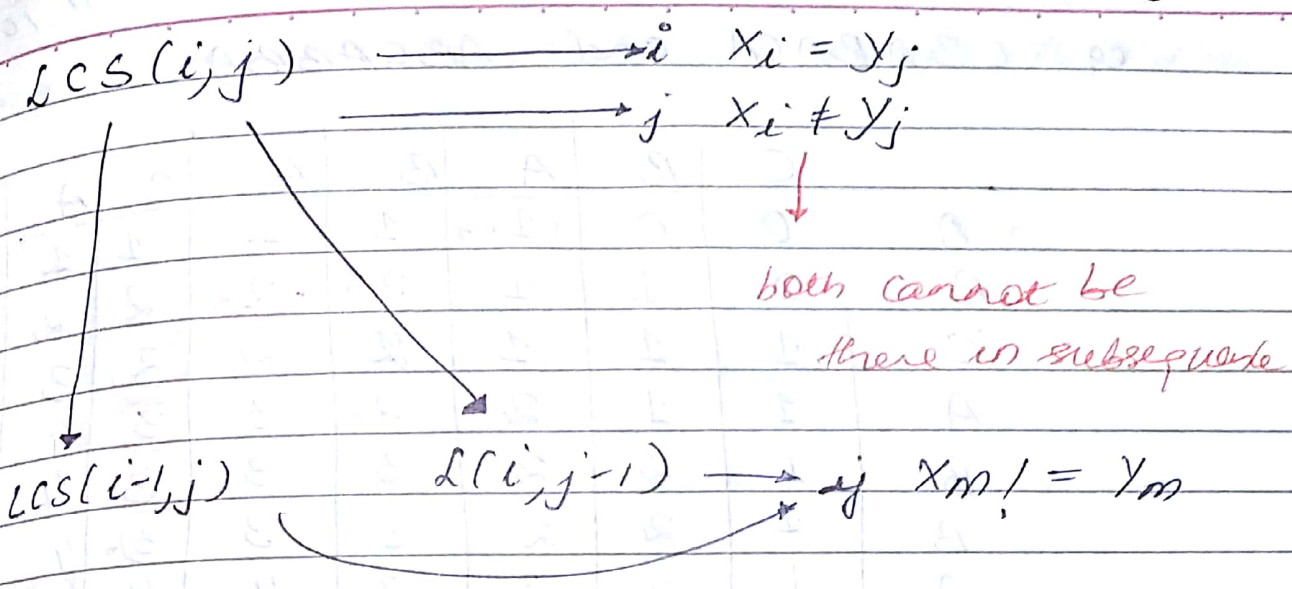
expected value = $(n \log n)$

$O(n)$ → suffix tree

$O(n \log n)$ → hashing on substrings.

↓

this is about time, doing this, takes more space also.



so $L(m,n) = \max(L(m,n-1), L(m-1,n))$

else,

$$L(m,n) = 1 + L(m-1,n-1)$$

where, $L(i,j)$ = LCS between x_1 and x_i and $y_1 - y_j$

$$\therefore LCS(i,j) = \begin{cases} 1 & x \in \{y_1, y_2, \ldots y_j\} \\ 0 & \text{otherwise.} \end{cases}$$

$$LCS(i,1) = \begin{cases} 1 & y_i \in \{x_1, x_2, \ldots x_i\} \\ 0 & \text{otherwise.} \end{cases}$$

```
for i = 2 to n
{   for j = 2 to n
    {
        if (x_i == x_j)
            L(i,j) = 1 + L(i-1, j-1)
        else if (L(i-1,j) > L(i, j-1))
            L(i,j) = L(i-1,j)
        else
            L(i,j) = L(i,j-1)
    }
}
```

O(mn)

eg: CBABACA and ABCABBA

	C	B	A	B	A	C	A
A	0	0	1	1	1	1	1
B	0	1	1	2	2	2	2
C	1	1	1	2	2	3	3
A	1	1	2	2	3	3	4
B	1	2	2	3	3	3	4
B	1	2	2	3	3	3	4
A	1	2	3	3	4	4	4

answer

$$\frac{i-1,j-1 \;\;|\;\; i-1,j}{i,j-1 \;\;|\;\; i,j}$$

so,

Backtracking

```
i=n, j=m
while (i>1 && j>1)
{
    if (xi == yi)
    {  Print (xi)
       i=i-1;
       j=j-1
    }
    else if (L(i-1,j) < L(i,j-1))
       j=j-1
    else
       i=i-1
}
```

O(n+m)

↳ the LCS obtained by backtracking as in above code is NOT a unique sequence. There may be EXPONENTIAL # of sequences

↳ Assignment :- generate a string which has exponential # of subsequences

↳ I'm interested in length only, not the sequence

↓

I need only (current) same column & previous column.

↓

Space reduced to O(n). backtracking is not possible then. from O(nm)

↳ It's variant will be asked in next test.

↳ LONGEST PALINDROME SUBSEQUENCE :-

given a sequence of length 'n', Compute the length of the longest palindrome.

eg: madam, race car

↳ induction on length → easy to solve.

i.e. given length, I can solve it.

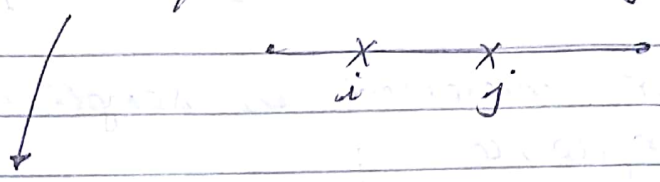
1 dimension, 2 dimension?

↳ state?

↳ what to store in table?

↳ Length of longest palindrome?

$T(i,j)$ = longest length palindrome subsequence b/w i & j



of string $x_i, x_{i+1}, \ldots x_j$

final answer at $T(0, n-1)$.

$T(i,i) = 1$.

$$T(i,i+1) = \begin{cases} 2 & \text{if } x_i = x_{i+1} \\ 1 & \text{otherwise.} \end{cases}$$

General case:-

if $x_i == x_j$

$L(i,j) = 2 + L(i+1, j-1)$

else if $L(i+1,j) > L(i,j-1)$

$L(i,j) = L(i+1,j)$

else

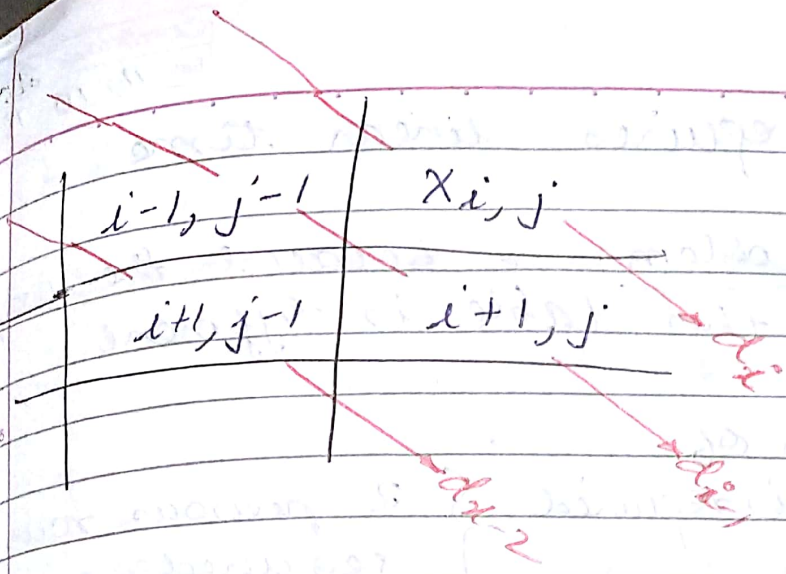
$L(i,j) = L(i,j-1)$

eg: ABCABBA and ABCABBA.

	A	B	C	A	B	B	A
A	1	1	1	3	3	3	5
B	X	1	1	1	3	3	4
C	X	X	1	1	1	2	4
A	X	X	X	1	1	2	4
B	X	X	X	X	1	2	2
B	X	X	X	X	X	1	1
A	X	X	X	X	X	X	1

Palindrome LCS:- AB X BA

where X = any one of {C, BA}



I need three columns (2 previous diagonals for it)

↳ I need n×n table and need to fill the upper half only.

$S(n) = O(n^2)$
$T(n) = O(n^2)$

↳ In case of LCS, changing loops does not matter.

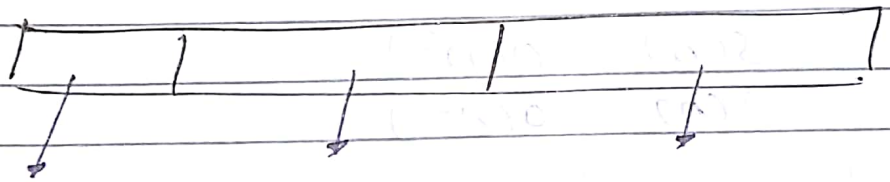
```
i=0 to n
A(i,i) = 1

for l = 1 to n-1          → because (0,0) has been taken care of
  for i = 0 to n-1-l
    { j = i+l.
      if (Xi == Xj)
          T(i,j) = 2+T(i+1,j-1)
      else if T(i+1,j) > T(i,j-1)
          T(i,j) = T(i+1,j)
      else
          T(i,j) = T(i,j-1)
    }
}
answer at ---> T(0,n-1)
```

↳ Backtrack requires linear time..

↳ important problem → because the way we wrote this table is different.

if only length of string is required { 2 previous rows required + current row.

so, $l = 3n$ (for above case)



where d_x = current diagonal

gives $r(i,j)$,
I should be able to write code & eq^{n}s in 'r'.

figure out 3 locations
↓
if overflow
↓
return & start from beginning

↳ but if you dont want this, i.e. need backtracking also, { $T(n) = O(n^2)$. $S(n) = O(n^2)$

↳ Assignment :- try above eg. & write code for it

LECTURE-23

↳ Longest Palindromic Subsequence :-

S
S^R ⟩→ LPS ✓ (easy to implement)

↳ MATRIX CHAIN MULTIPLICATION :-

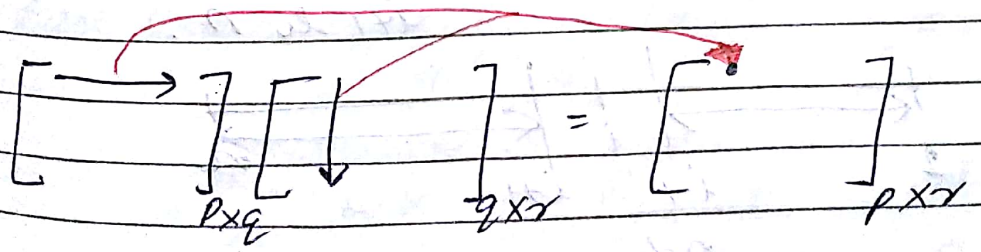
given $A_1, A_2, A_3, A_4, \ldots A_n$.
Compute the fastest way i.e. minimum # of multiplications to multiply $A_1 \text{---} A_n$.

suppose 'C' = $A_{p \times q} \times B_{q \times r}$

$C_{p \times r} = A_{p \times q} \times B_{q \times r}$

# of computations = $p \times q \times r$

↳ matrix multiplication is ~~commutative~~ not commutative and is associative,
i.e. $AB \neq BA$ but $A(BC) = (AB)C$



q-multiplications & q-1 additions.

so, $A_1(P_0, P_1)$. $A_2[P_1, P_2)$ ... $A_n(P_{n-1}, P_n)$

$P_0, P_1, P_2, \ldots, P_n$

eg:- different parenthesization will have different # of multiplications for product of multiple matrices.

$A(10, 100), B(100, 5), C(5, 50)$

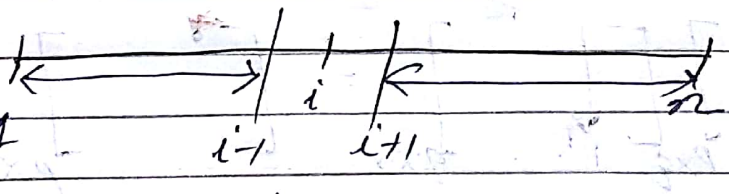
$(AB)C = 10 \times 100 \times 5 + 5 \times 50 \times 50$
$= 7500$

$A(BC) = 10 \times 100 \times 50 + 100 \times 5 \times 50$
$= 75000$

↳ Proof:- Brute force ways to multiply = exponential

for matrices given,
important is last multiplication I do (i)

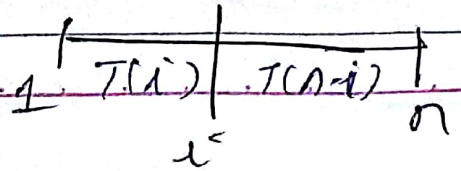
previous iterations → I've done matrix
1 to i
i+1 to n.



$n-1$

$$T(n) = \sum_{i=2} T(i) \cdot T(n-i)$$

↓
total #06 ways to multiply n-matrices



$$T(n) = \sum_{i=2}^{n-1} T(i) \cdot T(n-i) > T(n-2) + T(n-2)$$

$T(2) = 1$ (# of ways to multiply 2)

$T(3) = 2$

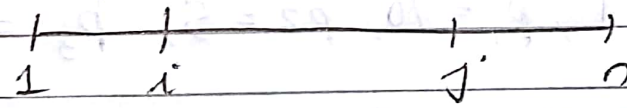
$T(4) = 6$

So, $T(n) > \sum_{i=2}^{n-1} 2T(n-2) > 2^k \cdot T(n-2k) > 2^{\frac{n-2}{2}}$

So, $T(n) = \Omega(2^n)$, so brute force does not work here!

So, $T(100) \geq 2^{49}$

↳ given matrices from 1 to n,



$M(i,j)$ = minimum # of multiplications required to multiply matrices $A_i, A_{i+1}, \ldots, A_j$

Base case :- $M(i, i+1) \longrightarrow p_{i-1} \times p_i \times p_{i+1}$

$M(i, i) = 0$

i^{th} matrix $\longrightarrow p_{i-1} \cdot p_i$ dimensions

$(i+1)^{th}$ matrix $\longrightarrow p_i, p_{i+1}$ dimensions

↳ first I want to compute $M(1,n)$

⟶

$T(i,k)$

Cost = # of computation A_{i-k} + Computation of $A_{k+1} - j$ + #[$A_{i-k} * A_{k+1-j}$

$T(k+1, j)$

where i ——— k ——— j $\quad i \leq k < j$

Best way is :- $T(i,k) + T(k+1,j) + P_{i-1} \cdot P_k \cdot P_j$

Let $M[i,j]$ = min. # of matrix multiplications then, $M[1,n]$ is answer.

$M(i,j) = \min\left(m(i,k) + m(k+1,j) + P_{i-1} \times P_k \times P_j\right)$

eg: $A_1(1,10) \times A_2(10 \times 5) \times A_3(5 \times 2) \times A_4(2 \times 25) \times A_5(25 \times 4)$

$\Rightarrow p_0 = 1,\ p_1 = 10,\ p_2 = 5,\ p_3 = 2,\ p_4 = 25,\ p_5 = 4$

$A_1 \times A_2 \times A_3 \times A_4 \times A_5$
$= 1 \times 10 \times 5 \times 2 \times 25 \times 4$

	1	2	3	4	5
1	0	50	60	310	268
2		0	100	600	380
3			0	250	240
4				0	200
5					0

2–4 → (row 2)

k = 2, 3

k = 2 → $A_2(A_3 A_4) = 250 + 10 \times 5 \times 25 = 1500$

k = 3 → $(A_2 A_3) A_4 = 100 + 10 \times 2 \times 50 = 600$

So, for final answer A(1,5) (1X10 ⟶ 10X4

k = 1 ⟶ 1X10 ⟶ 10X4

$A_1 \times (A_2 \text{ —— } A_5)$

= 380 + 1X10X4 = 420.

k = 2 ⟶ $(A_1 \times A_2)(A_3 \times A_4 \times A_5)$

1X5 5X4

= 50 + 240 + 1X5X4 = 310.

k = 3 ⟶ $(A \times A_2 \times A_3) \times (A_4 \times A_5)$

m(1,3) + M(4,5) + 1X2X4

= 60 + 200 + 8 = 268.

answer

k = 4 ⟶ $(A_1 \text{ —— } A_4) \times A_5$

= 310 + ...

↳ Have a look at Base case you're considering, whether it is

0 —— n-1

or

1 —— n ⟶ I am assuming this (default)

↳ Code :-

M(i, i) = 0

M(i, i+1) = $P_{i-1} \times P_i \times P_{i+1}$

for l = 3 to n

for i = 1 to n-l

j = i+l-1

T[i,j] = T[i+1, j] + P(i-1)XP(i)XP(j)

for k = i+1 to j

if [T(i,j) > T(i,k) + T(k+1,j) + $P_{i-1} \cdot P_k \cdot P_j$

T(i,j) = T(i,k) + T(k+1,j) + $P_{i-1} \cdot P_k \cdot P_j$

$T(n) = O(n^3)$
$S(n) = O(n^2)$ ⟶ I fill only the matrix
$O(n^2) \times$ lookup $O(n)$
$T(n) = O(n^3)$.

↳ Backtracking :- 268 (answer)
↓
how this min # of computation was reached?

then computation I do is when I do backtracking
$O(n^2)$ ⟶ $T(n)$
↳ we also store the value of 'k' → called predecessor (π)

$$M(i,j) = \min_{i \le k \le j} \left(\text{---} \overset{i}{\text{|}} \quad \overset{k}{\times} \quad \overset{j}{\text{|}} \text{---} \right)$$

then

$\pi(i,j)$ = arg min ( )
predecessor $i \le k < j$

$\min = \min_{1 \le i \le n} a_i$

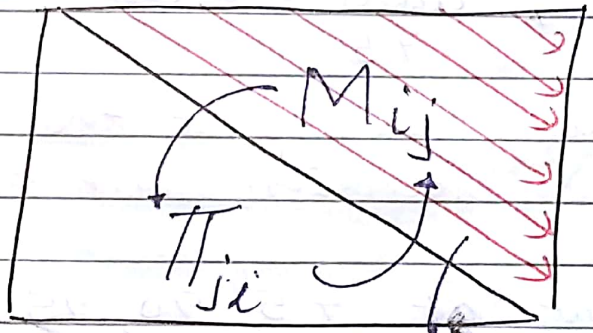
$k = \arg\min_{1 \le i \le n} a_i$

so, $a[k]$ is minimum.

↳ So, If I maintain the predecessor also, then I can find the multiplication order in $O(n)$

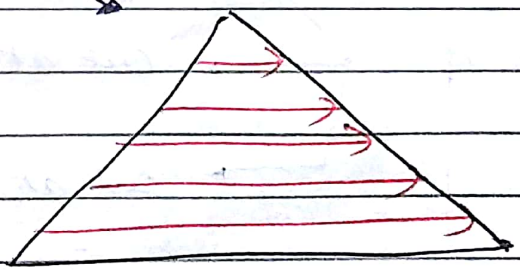
```
MM(i,j)
{if(j>i)
 {
    k = π(i,j)
    MM(i,k)
    MM(k+1,j)
    print k
 }
}
```

↳ But, I'm not using the lower half of the matrix, I can use it as π



$\pi(n) = O(n^2)$

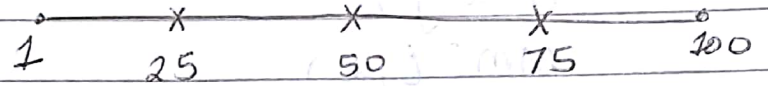
↳ I can compute all this in $O(n^3)$ time and $O(n^2)$ space.

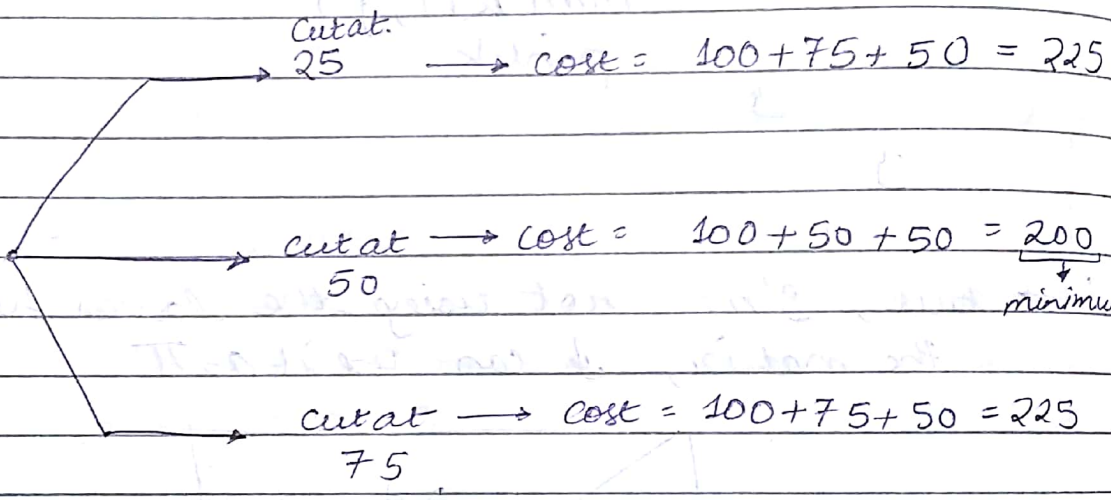


↳ this cannot be optimized.

LECTURE-24

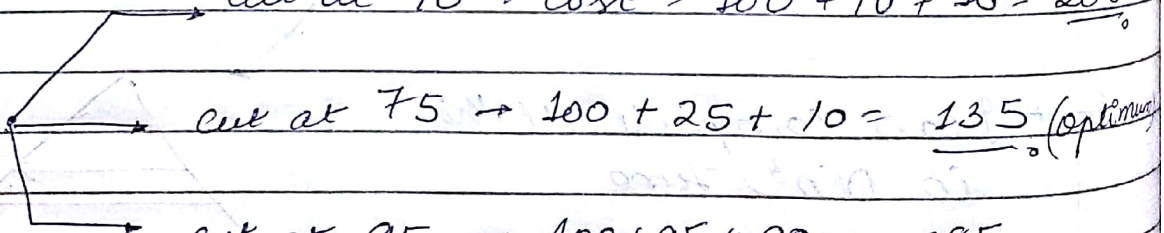
↳ **Rod cutting Problem** :-
given a rod of length 'n', find the way in which cutting cost is minimum
eg: n = 100, cut at 25, 50, 75





In this case, cutting at the centre gives optimum solution, but that's not always true

eg:- cut at 75, 90, 95, n = 100



↳ So, we cannot generalize it, to which side we need to start with.

↳ Consider a generic problem,
you've been given n-indices where you need to cut the rod in (n+1) pieces.
then, $l(i)$ = location I need to cut
$l(0)$ = 1 (left end point)
$l(n+1)$ = 100 / length of the rod.

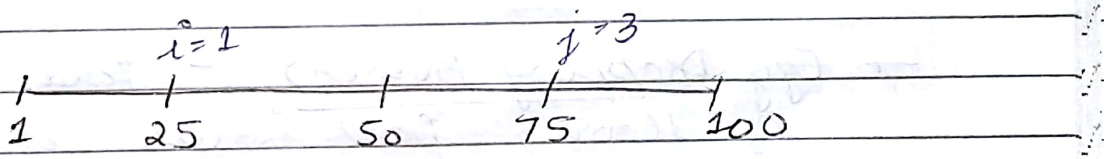
so,

$l(i)$	1	25	50	75	100
i	0	1	2	3	4

Induction is on the # of cuts in the graph.

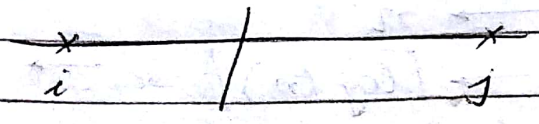
$T(i, j)$ = cost of cutting from 'i' to 'j'
$0 \le i \le j \le n+1$.
# of cuts = $j - i - 1$.

eg:- $i = 1$, $j = 3$



# of cuts = $j - i - 1$.

so, $T(i, i+1) = 0$ (no cuts required)
$T(i, i+2) = l(i+2) - l(i)$
I need only one cut at location $i+1$.
$T(i, j) = l(j) - l(i) + \min_{i<k<j} \{T(i,k) + T(k,j)\}$



k → is a random location b/w i & j
and $T(0, n+1)$ = final answer
$T(n) = O(n^3)$ → visit n^2 states of time
$S(n) = O(n^2)$

: : (fill diagonally)

so, h = 2 to n
i = 0
j = h to n
if (j - i > 2)
k = i + 1 to j and find min
else
L(i, j) = A(j) - A(i)

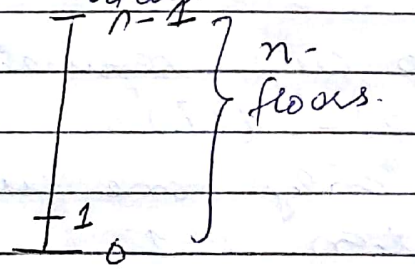
→ Backtracking :- find where we did the change.

→ π(i, j) → store the value of 'k' it shows the value of parent selected.

Instead of a new 2D matrix, use L(j, i) to store values for backtracking.

O(n)

→ Egg Dropping Problem :- Find the lowest # of floor, from which if an egg is dropped, it breaks in minimum # of trials.



K eggs are similar.

drop(egg)

breaks — does not break.

# of trials

(n, 1) → n

(n, ∞) → ⌈log(n)⌉ → i.e. I don't need more than 'n' eggs in any case.

floors — eggs — trials with eggs

$\lceil \log_2 n \rceil \leq min \leq n$

$\log n < min \leq n$

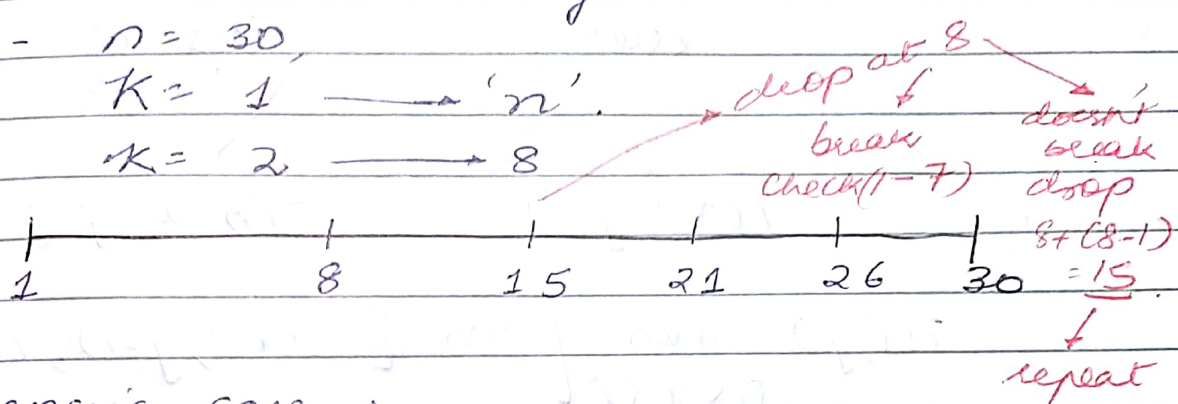
since, we do not need more than $\log_2 n$ eggs, then

$1 < K < \log n$

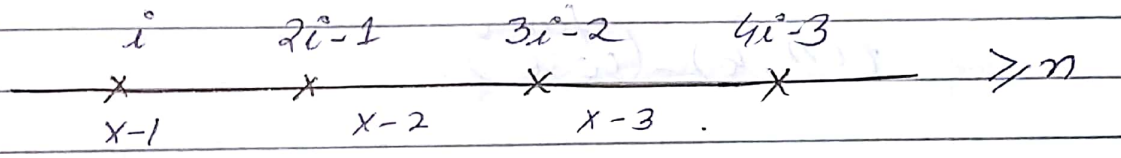
eg:- $n = 30$,

$K = 1 \longrightarrow$ 'n'.

$K = 2 \longrightarrow 8$



so, generic case →



$1 + 2 + \ldots + i \geq n$

$\frac{i(i+1)}{2} \geq n$ ← considering max 'i' trials

↳ Smallest 'i' such that above holds true.

eg: $n = 15$.

we need 5 trials.

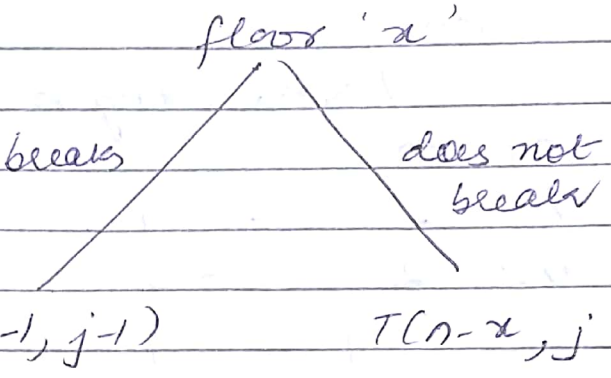
so, $n = 30 \longrightarrow$ eggs $\longrightarrow$ trials required

30	1	30
30	2	8
30	3	6 (Binary + 2 egg part)
30	4	6 (Binary + Binary + 2 egg part)
30	5	5

given, $T(i, j)$ ⟶ $T(n, k)$ needed.

$T(i, 1) = i \quad \forall i$

i → # of floors, j → # of eggs



$$T(i,j) = \min_{0<x<i} \left[\max \{ T(x-1, j-1), T(n-x, j) \} \right] + 1$$

$T(n, k) = (\text{final cost})$.

$T[i, j] = k$ ⟶ where you drop the first egg
↳ $O(n)$

$T(n) = O(n^2 k)$

$S(n) = O(nk)$

↳ **Homework**

B = Boolean variable in length 'n'

0 0 0 0 ... 0 1 1 - - - 1 ⟶

find the point where '0' switches to '1'. Check that there is no '1' b/w 0's and no '0' b/w 1's.

↳ **Longest Increasing Subsequence (LIS) :-**

sort the sequence and apply LCS

↳ $T(n) = O(n^2)$

↳ $S(n) = O(n^2)$

LIS(i) = length of LIS ending at a_i (including a_i)
LIS(0) = 1.
and LIS(n-1) = n, length of longest sequence which includes last element.

LIS = $\max_{0 \le i \le i-1}$ LIS(i) ⟶ not necessarily includes LIS(n-1), a(n-1) as last term, I don't actually know, what is the last term.

$$LIS(i) = 1 + \max_{0 \le j < i \text{ such that } a_j \le a_i} LIS(j)$$

→ to keep a track of increasing sequence

i = 1 to n (i<n) smallest value = 1
j = 0 to i (j<i) becz, smallest LIS = 1 (i's length)
if $a_j \le a_i$ && LIS(j)+1 > LIS(i)
LIS(i) = LIS(j)+1
$\pi(i) = j$

Keep a track of π value, then backtracking is done in O(n);
so, it can be solved in O(n log n). (sort & solve)

$T(n) = O(n^2)$
$S(n) = O(n)$ ⟶ but what if $n = 10^6$?

so, $\forall i$, what is the first location where I get a # smaller than this one?
↳ Using stack → O(n)

LIS values	1	2	1	2
	2	3	1	10

but max LIS = 3 (2, 3, 10).

eg:- Find the longest sequence increasing (when number comes, you have an option to add it to right or left of smallest).

12	7	10	8	1	6	5	14	4	9	10	1
0	1	0	1	0	1	1	0	1	1	1	1

What ever is in between 2 # has to be discarded, trust what # to include is the sequence has to be carefully chosen.

actual → 2, 4, 5, 6, 7, 8, 9, 10 ⟶ LIS = 8

1, 7, 12, 14 (length 4)
10 & 8 are discarded becz anything b/w 1 & 14 will be discarded

$LIS(i)$ = a_i ... $n-1$

$LDS(i)$ = a_i (included) ... $n-1$

$$\max_{0 \le i \le n-1} LIS(i) + LDS(i) - 1$$

$$\forall i \quad \max_{\forall i} LIS(i) + LDS(i) - 1$$

$\forall i$ $\begin{cases} LIS(i) = 1, & \pi_1(i) = n \\ LDS(i) = 1, & \pi_2(i) = n \end{cases}$

for $i = n-1$ to 0
  for $j = i+1$ to n
    if $a_j \geq a_i$ && $LIS(j)+1 > LIS(i)$
      $LIS(i) = LIS(j)+1$
      $\pi_1(i) = j$
    if $a_j \leq a_i$ && $LDS(j)+1 > LDS(i)$
      $LDS(i) = LDS(j)+1$
      $\pi_2(j) = j$

π_1 and π_2 gives elements in LIS & LDS.

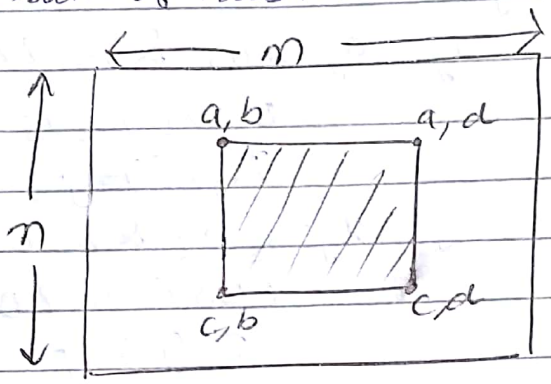
$T(n) = \Theta(n^2)$
$S(n) = \Theta(n)$ ⟶ 4 arrays of length 'n'.

where LIS = LIS till 'i' from 0, a_i included
LDS = LDS from i to n-1 including a_i

LECTURE - 25

MAXIMUM SUBMATRIX PROBLEM:-

given a matrix 'm' having +ve & -ve nos.
find a submatrix so that the sum of submatrix is max.

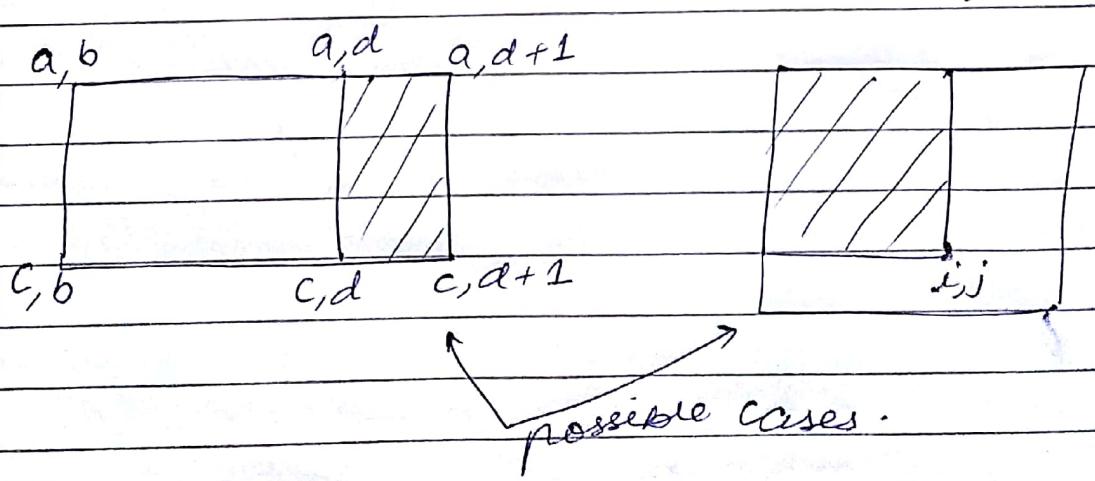


initial maxsum = -∞

```
for a = 0 to n
    for b = 0 to m
        for c = a to n
            for d = b to m
                for i = a to c+1
                    for j = b to d+1
                        sum = sum + A[i,j]
                    if (sum > max)
                        sum = max
```

iterating over all possible matrices.

$T(n) = O(n^3 m^3)$



possible cases.

Value of submatrix where one end point is (0,0) and other is (i,j)

sum = m[c,d] − m[a−1,d] + m[c,b−1]
+ m[a−1,b−1]

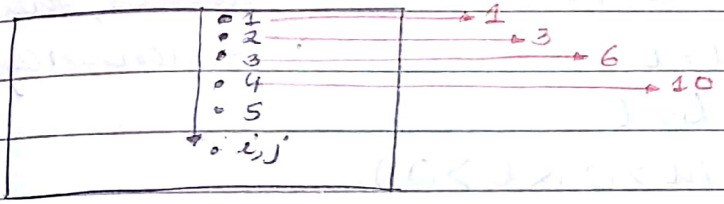
↓

$O(m^2n^2) < O(n^3m^3)$

↓

can replace inner two loops
pre computations = O(mn)

↳ create a matrix that stores column sum



m(i,j) represent sum of all elements above it in the same column.

```
i = 0
for j = 1 to m                                   } O(m)
    m[i,j] = A[i,j]
for i = 1 to n
    for j = 0 to m                               } O(mn)
        m[i,j] = m[i-1,j] + A[i,j]
```

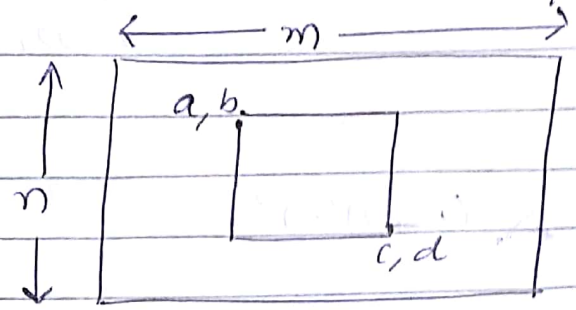
```
for i = 0 to n
    for j = 1 to m                               } O(nm)
        m[i,j] = m[i,j-1] + m[i,j]
```

pre-computation

so, T(n) = $O(n^2m^2)$

so, given 'A' we can compute M in O(nm)

↳ now I want a square matrix



$a \rightarrow 0$ to n
$b \rightarrow 0$ to m

↓

c and d don't have abitary values, there is a dependency.

```
for a = 0 to n
  { for b = 0 to m
      { for l = 0 to n
          { c = a + l
            d = b + l
            if (a > 0 & b > 0)
                sum = m(c,d) - m(a-1,d) - m(c,b-1) + m[a-1][b-1]
            else if (a > 0)
                sum = m[c,d] - m[c,b-1]
            else if (b > 0)
                sum = m[c,d] - m[a-1,d]
            else
                sum = m[c,d]
          }
      }
  }
```

→ $T(n) = O(n^2 m)$
given $n < m$.
otherwise
$T(n) = O(m^2 n)$

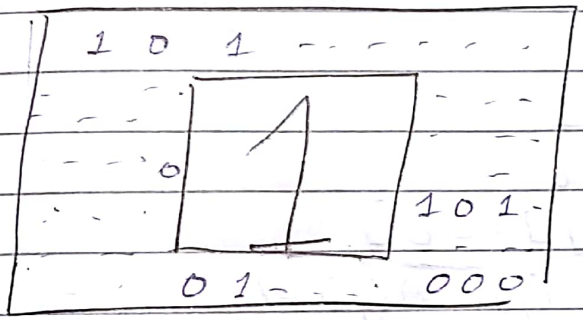
So, rectangular ⟶ $O(n^2 m^2)$
square ⟶ $O(n^2 m)$
fixed size square → $O(nm)$
highest sum subarray ⟶ $O(n)$

2→ given an $n \times m$ matrix, we need a $k \times k$ matrix

```
a = 0 to n-k
    b = 0 to m-k
        c = a+k
        d = b+k
        :
```

} $O(nm)$

eg: Binary Matrix :-
find a rectangular matrix where everything is '1' & of max. size.



eg seen: # of elements = $(c-a) \cdot (d-b)$
then all entries are one.

$O(n^2m^2)$ algorithm → I've only 0 & 1's so I should be able to answer it faster

↓

consider the last row, take sum of all 1's in the column & then the array we get, represents it with histogram 101

$O(m) \times O(n) = O(nm)$

Sol^n:

```
i = n-1
    for j = 0 to m
    {       A(j) = m(i,j)
            k = i
            while (k>0 && m(k,j)==1)
            {       k--;
                    A(j) = A(j) + m(k,j)
            }
    }
    max = findBestSol$^n$(A, 0, m-1)


i = n-2 to -1
    j = 0 to m
        if [A[j] == 0]
                A(j) = A(j) - 1;
        else {
                if (m(i,j)==0)
                        A(j) = 0
                k = i
                while (k>0 && m[k,j]==1]
                { A(j) = A(j) + 1
                  k = k-1
                }
        }

sol$^n$ = findBestSol$^n$(A, 0, m-1) ⟶ O(m)
if (sol>max)
        max = sol
```

↳ The claim is that total time I spend in updating 'vector A'

A ⟶ O(nm).

we need a square sub matrix such that every entry is 1 in rectangle of $n \times m$

$mn = min(h, l)$

$h = min(r(i), l(i))$

$l = r(i) - l(i) + 1$

→ this gives a square matrix.

the way we do 'a' & 'b' etc. is still the same.

↳ I also want to find $K \times K$ where all entries are 1 → trivial.

If return value is $> K^2$

return yes

else return no.

similarly, all zeroes: