Given an Array of size N.
Find the sum of all the subarrays.

$A_4 = [1\ 3\ 0\ 7]$

$|A| = n$

# of subarrays $= \frac{n(n+1)}{2}$

[1] [1 3] [1 3 0] [1 3 0 7]
1 4 4 11
[3] [3 0] [3 0 7]
3 3 10
[0] [0 7]
0 7
[7]
7

total sum = 1+4+4+11+3+3+10+7
= 50

---

(1) Brute Force → $O(n^2)$
(2) Prefix sum sum / carry forward → $O(n^2)$

$|A| = n$
How many subarrays? $\frac{n(n+1)}{2} = O(n^2)$

for each subarray → $O(n^2)$
do something → $O(1)$
$\geq O(n^2)$

---

invert the problem. → pattern

instead of calculating the sum of elements of subarrays.

for each element : O(n)

do something. → O(1)

~~find the contribution of the element.~~

---

for each subarray

for each element in it

$s += e$

Inverting the problem

for each element

for each subarray containing this elment

$s += e$

---

given an element → what all subarrays contain this element.

∵ for all such subarrays we are doing the same thing

↳ we just need the count of the subarrays

for each element

for each subarray containing this elment

$s += e$

⟹

for each element

k = # of subarrays containing this element

$s += k * e$

for each element

k = # of subarrays containing this element

$S += k * e$ → contribution of that element to the total sum.

= element value * # of time it occurs in subarrays.

$A_4 = [1\ 3\ 0\ 7]$ $n = 4$

$(i+1)(n-i)$

i	$(i+1)(n-i)$
0	$(0+1)(4-0) = 4$
1	$(1+1)(4-1) = 6$
2	$(2+1)(4-2) = 6$
3	$(3+1)(4-3) = 4$

[1] [1 3] [1 3 0] [1 3 0 7]

[3] [3 0] [3 0 7]

[0] [0 7]

[7]

total sum = $4*1 + 6*3 + 6*0 + 4*7 = 4 + 18 + 0 + 28 = 50$

$A_4 = [1\ 1\ 1]$

i	$(i+1)(n-i)$
0	$(0+1)(4-0) = 4$
1	$(1+1)(4-1) = 6$
2	$(2+1)(4-2) = 6$
3	$(3+1)(4-3) = 4$

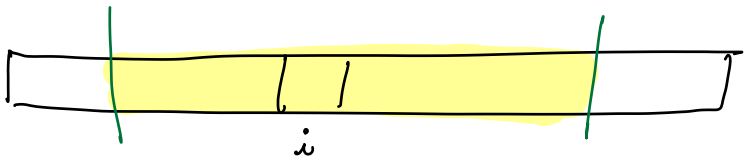
[1] [1 1] [1 1 1]

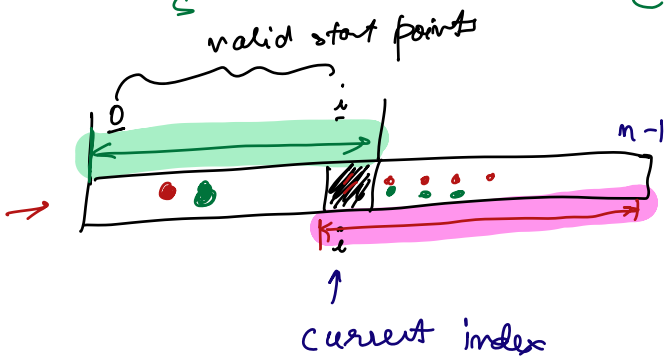
[1] [1 1]

[1]

given an element at index i

how many subarrays contain it?





if a subarray contains A[i] then

it must start b/w $\{0 -- i\}$ → $(i+1)$

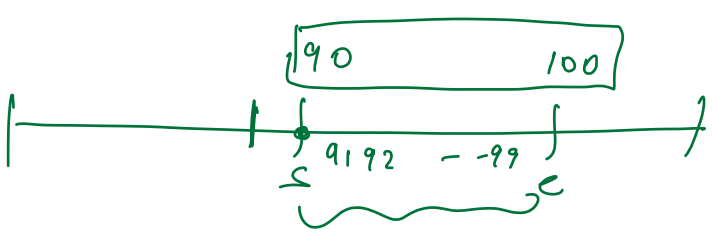
& end b/w $\{i -- n-1\}$ → $(n-1)-(i)+1$

= total no of ways! $5+5+5$

$= 15$

$2 * 5$

If the choices are independent for all possibilities → multiply.



~~100-90~~

$100-89$

$100-90+1$

$e-(s-1) = e-s+1$

Start → $(i+1)$

End → $(n-1) - (i-1)$

$= (n-i)$

# of subarrays containing the element $A[i]$

is $= \underbrace{(i+1) * (n-i)}_{R}$

```
s = 0
for i in range(n):
    s += (i+1)*(n-i) * A[i]
return s.
```

TC $O(n)$

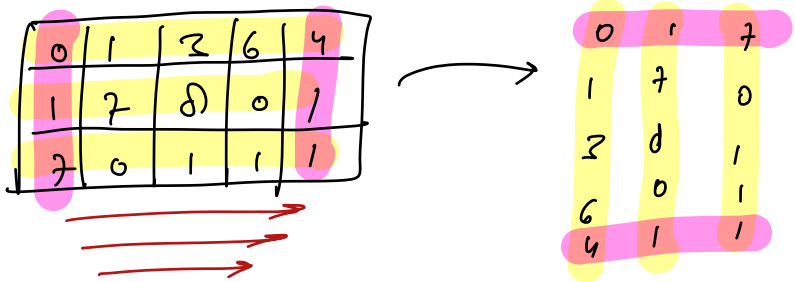
SC $O(1)$

Kadane's algo → find the subarray with max sum in $O(n)$ time & $O(1)$ space in a single pass.

---

given a Matrix of size $N \times m$

↳ transpose the matrix



① return a new matrix which is a transpose.

```
for i ← 0 -- n-1
    for j ← 0 -- m-1
        x = A[i][j]        → row wise
```

```
for j ← 0 -- m-1
    for i ← 0 -- n-1
        = A[i][j]          → col wise.
```

$M[i][j] \longrightarrow M'[j][i]$

$n \times m \qquad m \times n$

loop in any manner.

$M2[i][j] = M[j][i]$

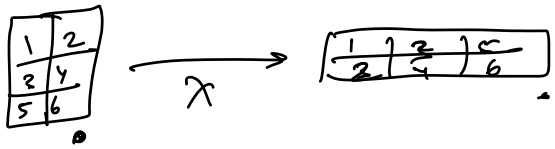
TC $O(n \cdot m)$

SC $O(n \cdot m)$

② in-place

can only be done for square matrices

$n = m$



---

in-place transpose a square matrix

```
for i in range(n):
    for j in range(i+1, m):
        M[i][j], M[j][i] = M[j][i], M[i][j]
```

→ restrict to elements above the diagonal.

a, b

```
temp = a
a = b
b = temp
```

swapping two variables

a, b = b, a

→ tuple unpacking to swap variables

avoid:

```
a = a + b
b = a - b
a = a - b
```

swapping without temp variable → Never ever do this!

work only 25% of the time.

Why do we ignore constants or lower order terms

in asymptotic notation → big oh.

Rotate the matrix

given a $M_{m \times n}$ → Rotate 90° clockwise

$M =$

1	2	3
4	5	6
7	8	9
10	11	12

4x3

i,j → j,i

transpose → $M' =$

1	4	7	10
2	5	8	11
3	6	9	12

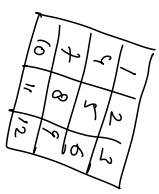
3x4

Rotate 90° clockwise

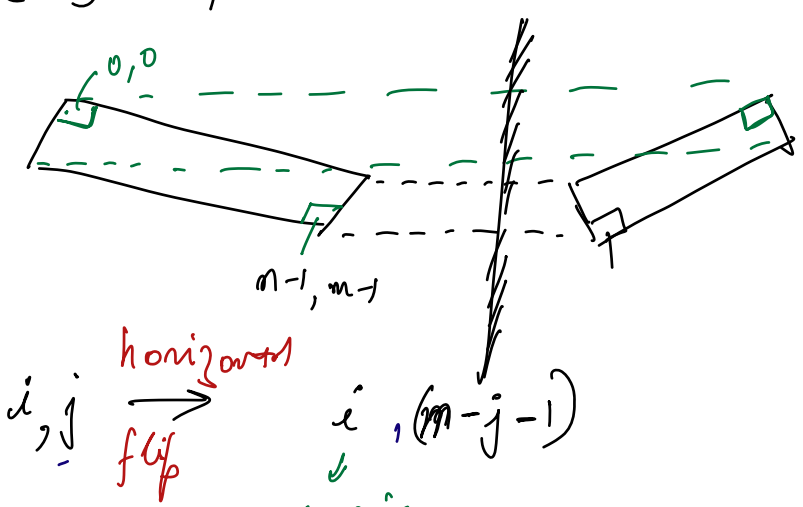
i,j → i, m-j-1 flip horizontal

10	7	4	1
11	8	5	2
12	9	6	3

3x4



rot(M) ≠ M^T

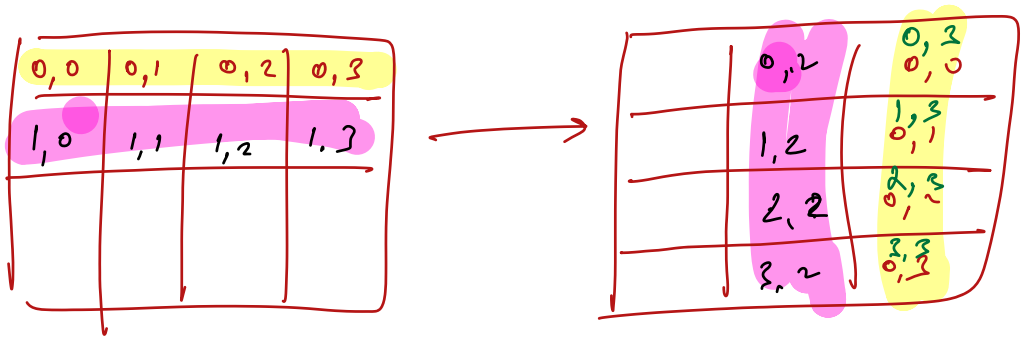


$(i, j \rightarrow j, i)$

+

$(i, j \rightarrow i, (m-j-1))$

$i, j \rightarrow (j, n-i-1)$



$0,0 \rightarrow 0,3$

$0,1 \rightarrow 1,3$

$0,2 \rightarrow 2,3$

$0,3 \rightarrow 3,3$

$1,0 \rightarrow 0,2$

$1,1 \rightarrow 1,2$

$1,2 \rightarrow 2,2$

$1,3 \rightarrow 3,2$

---

rotate 90° ⇒ transpose + flip horizontal.

$(i, j) \rightarrow (j, i)$

$(n, m) \rightarrow (m, n)$

transpose

$i, j \rightarrow i, (m-j-1)$

$\rightarrow j, n-i-1$

flip horizontal

$i, j \rightarrow j, n-i-1$

Rotate

Not - inplace

TC - O(n·m)

SC - O(n·m)

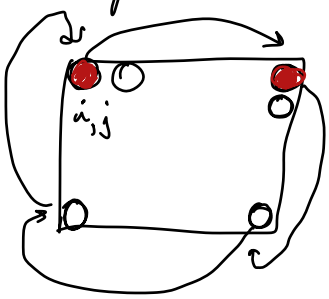
loop over each cell → O(n·m)

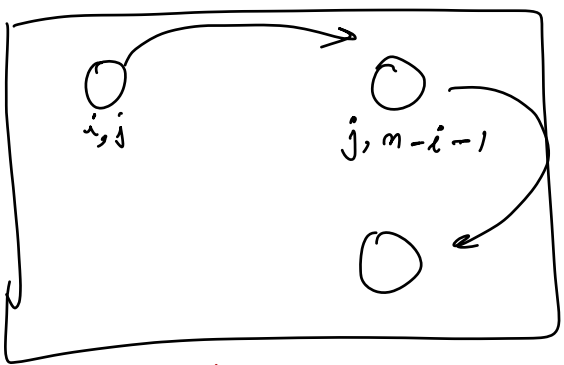
m2[j][n-i-1] = m[i][j]

In-place → only for square matrices.

TC - O(n·m)

SC - O(1)

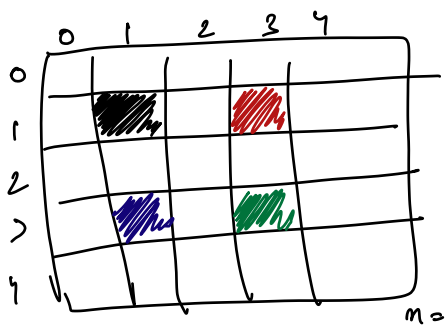




i, j → j, (n-i-1)

(i, j) → (n-i-1), (n-j-1)

n = m = 5



(i, j) (1, 1)

(j, n-i-1) (1, 3)

((n-i-1), (n-j-1)) (3, 3)

((n-j-1), i) (3, 1)

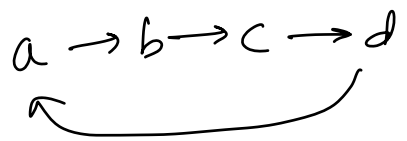
n=5
m=5

(n-i-1), (n-j-1) → (n-j-1), n - (n-i-1) - 1

$= (n-j-1),\ i$

$(n-j-1),\ i \longrightarrow i - \left(n-\left((n-j-1)\right)-1\right)$

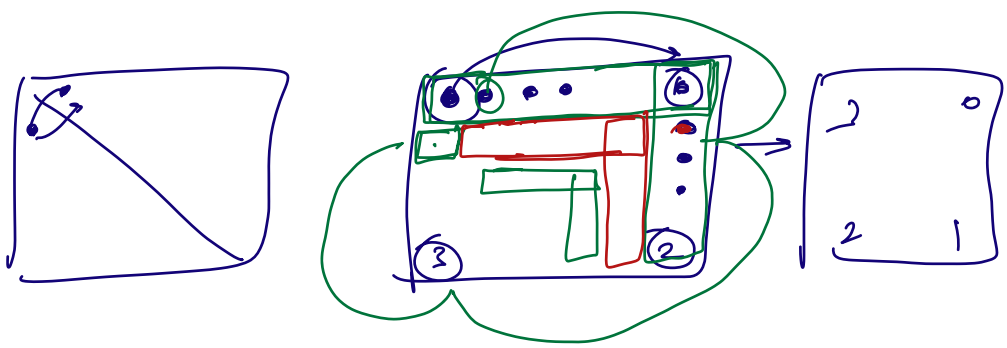
$= i, j$

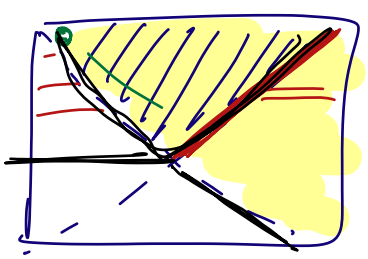



```
t = a
a = b
b = c
c = d
d = t
```

a, b, c, d = b, c, d, a

↳ works.





row <= col

$i \rightarrow (n-i)$

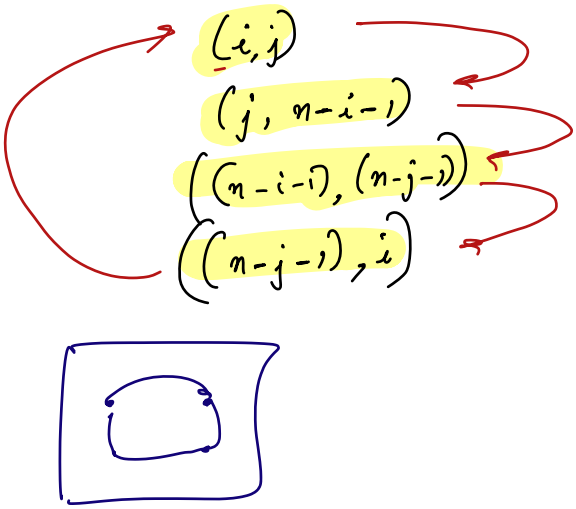
```
for row in range (n/2):
    for col in range (row, n-row):
        4 way Shift.
```

↳ off by one.

$n-i$

$(n-i-1)$

Rotation



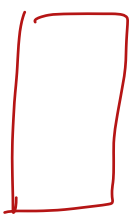
Rotation = transpose + flip horizontal

transpose → i, j → j, i

flip horizontal → i, j → i, n-j-1

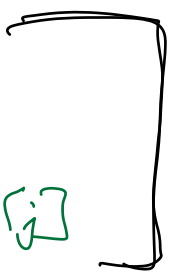
Rotation

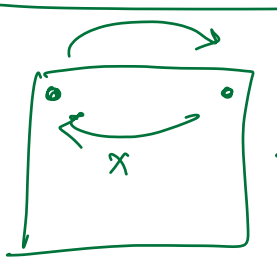
i, j → j, (n-i-1)

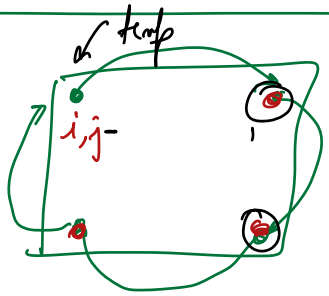


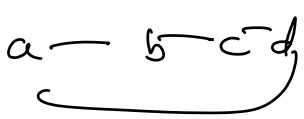
not-in-place

```
for i in range(n)
    for j in range(n)
        M2[j][n-i-1] = M1[i][j]
```









t = a
a = b
b = c
c = d
d = t

given several Christmas Trees.

↳ height

↳ cost.

	0	1	2	3	4	5	6
height	3	5	2	7	8	9	15
cost	10	5	12	15	20	10	3

(0, 1, 3) → 30

(2, 5, 6) → 25

(0, 1, 6) → 18

(1, 5, 6) → 18

choose 3 christmas trees in order (L → R)

① increasing height

② minimize the total cost.

find the min cost achievable.

	0	1	2	3	4	5	6	7	8	9
heights	10	5	8	2	3	4	6	9	1	5
costs	2	3	8	4	15	99	2	4	1	5

① increasing order of heights from L → R

② min cost.

(1, 6, 7) → 3+2+4 = 9 ⟶

(3, 6, 7) → 4+2+4 = 10

Brute Force Solution

go over all possible choices of 3 trees

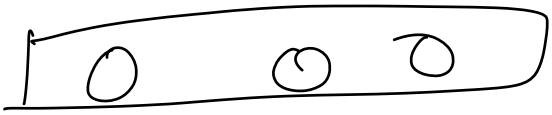
↳ $n_{C_3} = \frac{n(n-1)(n-2)}{6} = O(n^3)$

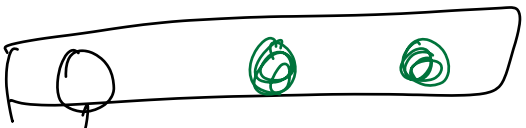
```
ans = float('inf')
for i in range(n):          → O(n)
    for j in range(i+1, n):     → O(n)
        for k in range(j+1, n):     → O(n)
            if H[i] < H[j] < H[k]:
                ans = min(cost, C[i]+C[j]+C[k])
return ans
```

TC	SC
$O(n^3)$	$O(1)$

optimize:

let's try to fix one or more values & see if we can find the rest efficiently

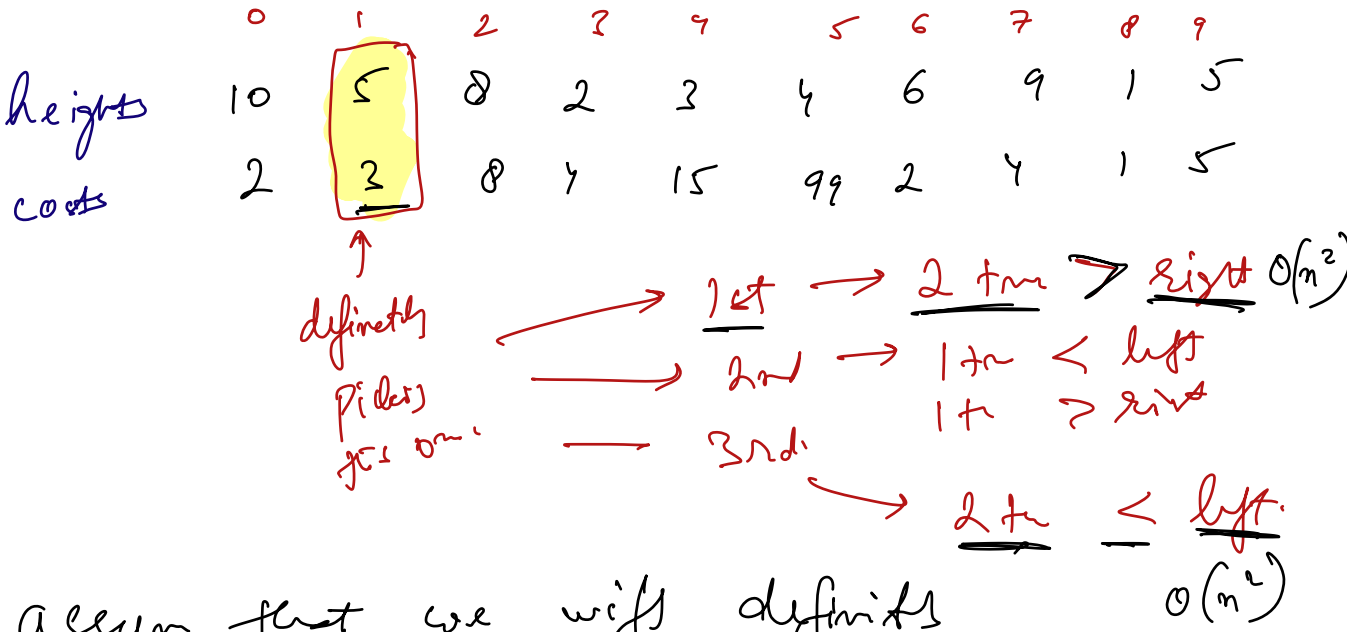




↳ go over all possible values.

```
for i in range(n):
    first = i
```

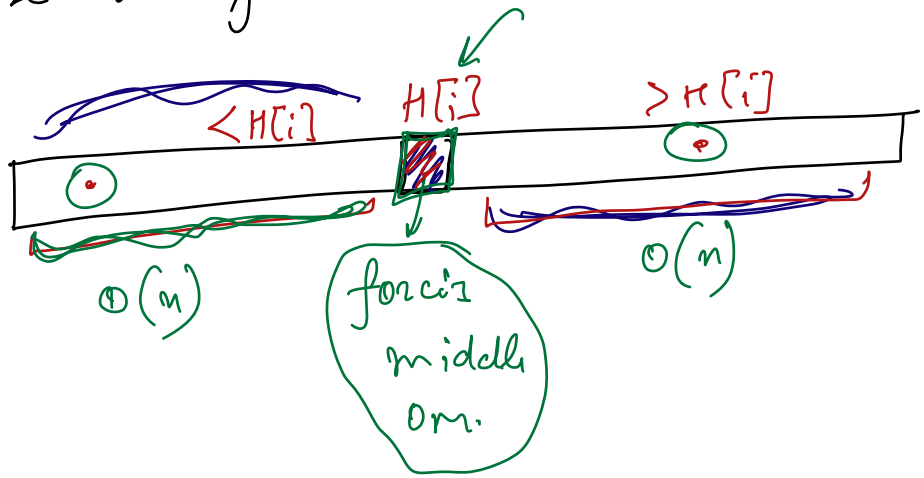
find the remaining two values efficiently $< O(n^2)$



assume that we will definitely choose Tree indexed at i

---

assume that we select a tree & we force it to be the middle tree




```
for i in range(n):
    h2 = H[i]
```

tree 1 → min cost tree from (0 -- i-1) such that h < h2

from 3 → min cost to from (i+1 .. n-1)
so h > h2

---

heights = [. .]
costs = [. .]
n = len(heights)

ans = ∞
range (1, n-1)
for t2 in range(n): → O(n)
    h2 = heights[t2], c2 = costs[t2]
    c1 = ∞
    for t1 in range(t2): → O(n) scan min cost from on left
        if heights[t1] < h2:
            c1 = min(c1, costs[t1])

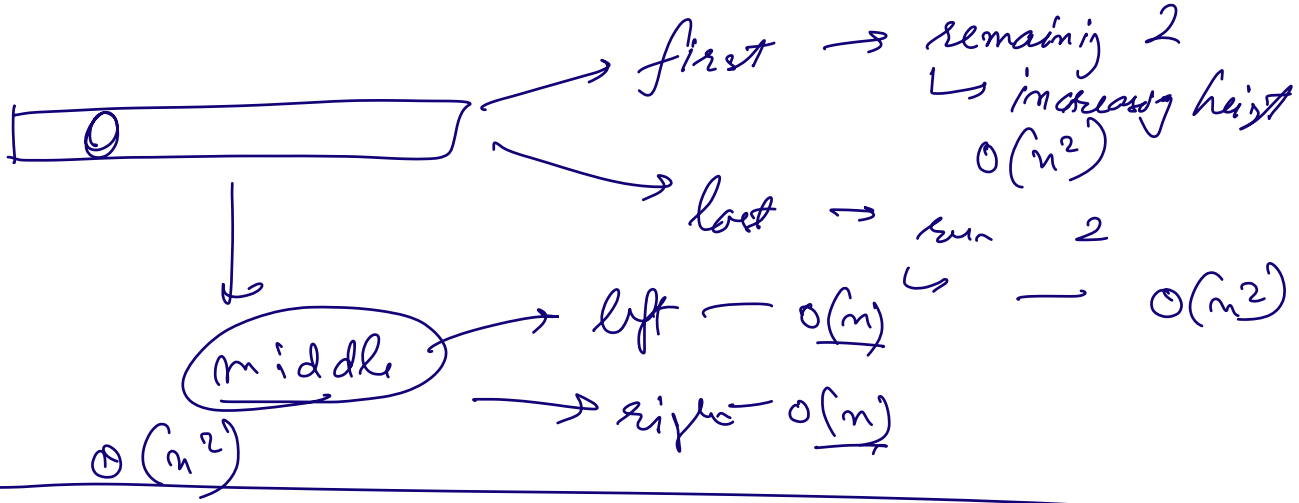
    c3 = ∞
    for t3 in range(t2+1, n): → O(n)
        if heights[t3] > h2:
            c3 = min(c3, costs[t3])

    cost = c1 + c2 + c3
    ans = min(ans, cost)

return ans

Rf → $O(n^3)$ → choose a triplet.

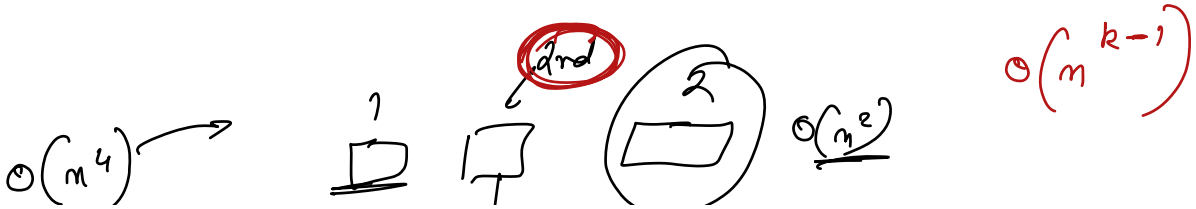
can I fix one thing & decide the rest efficiently?



$O(n * (n+m))$

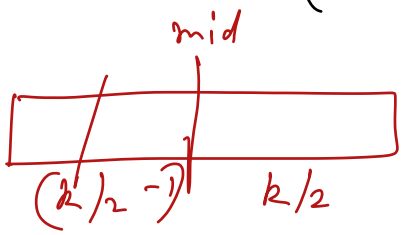
$O(n+2n)$

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



$O(n)$ $O(n)$

$$O(n) + \left(O(n) + O(n^2) \right) = O(n^3)$$

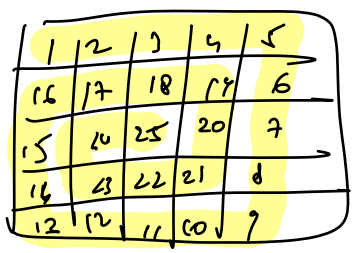


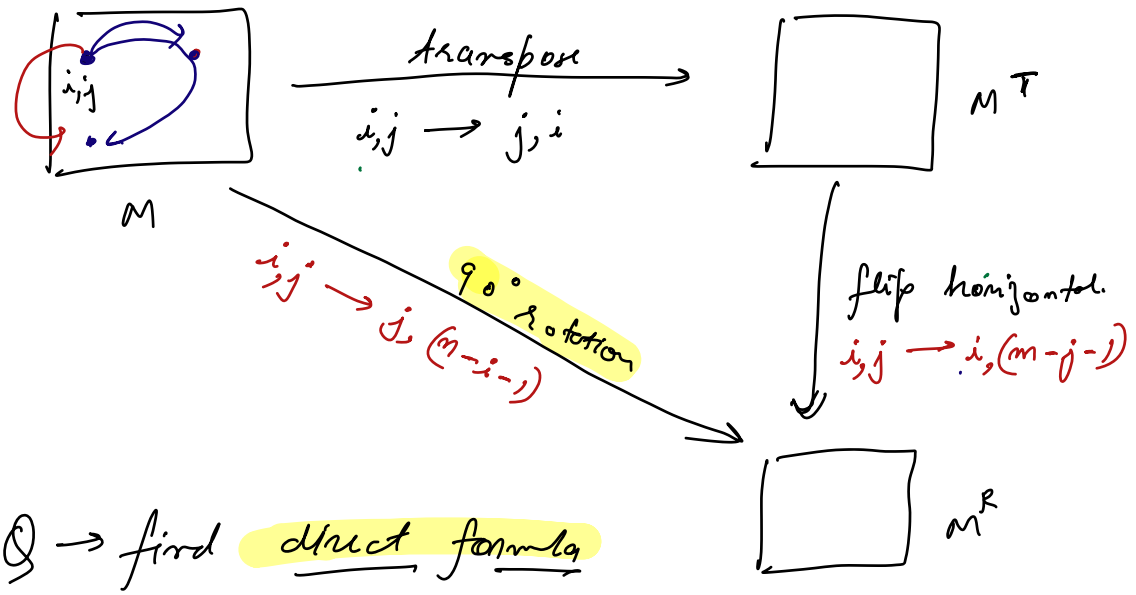
---

① Extra class — Python — looping → print patterns.

given a matrix of size N×N

fill it with $1 - N^2$ in spiral order





Q → find direct formula

vs brute down & solve.

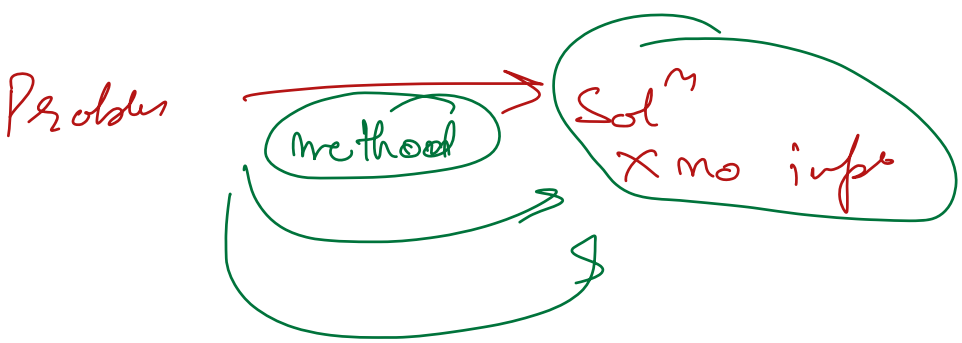
270° rotation
−90° rot

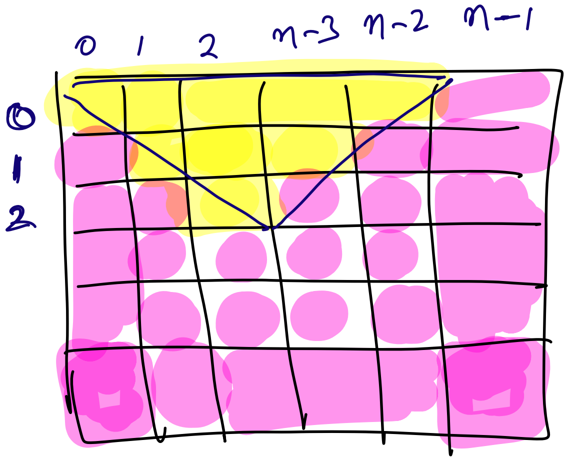
M^T = Transp(m)

ans = flip_Hori(M^T)

did not solve 90°

90° → 3 times → solve ✓





col → (i, (n-i-1))

Why do we ignore constants in asymptotic notation?

Problem —algorithm→ solⁿ
solve

What is the best algorithm? → most efficient TC

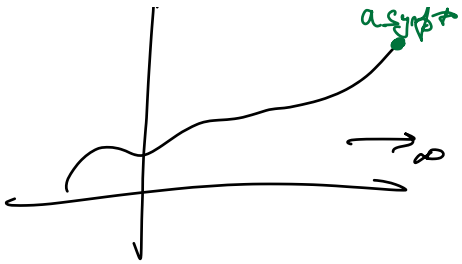
Avijit → 3s —Google chat→ ∅ run the → 30s
Mohit → 10s —pocket watch→ ∅ on the same hardware → 0.01s

as the input gets larger how does the algo scale?

10,000

Avijit —Mac→ 30s —1 trillion→ 40s
10,000
Mohit —mac→ 0.01 ——→ never finishes

(1) we have to look at the asymptote

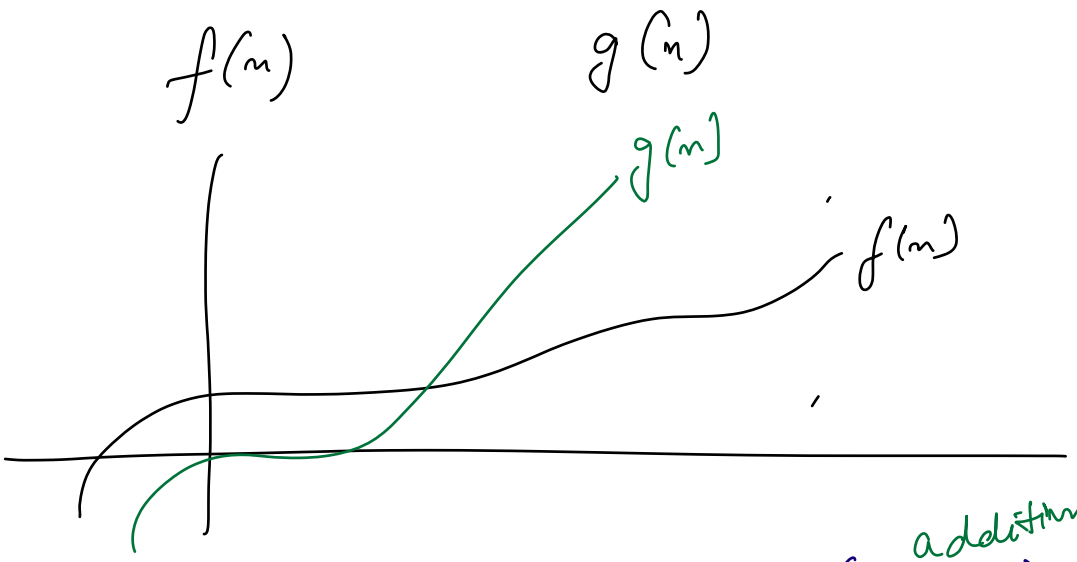


$\lim_{\text{inpsize} \to \infty}$ (Perform)

(2) Hardware / Lang / Coder → should be ignored.

↳ induce constant additive or multiplicative factors.

given two function



NASA → 1 day to boot ↓ → (+ t) additive

MAC → bot ↓

NASA → ^

MAC → 100 x

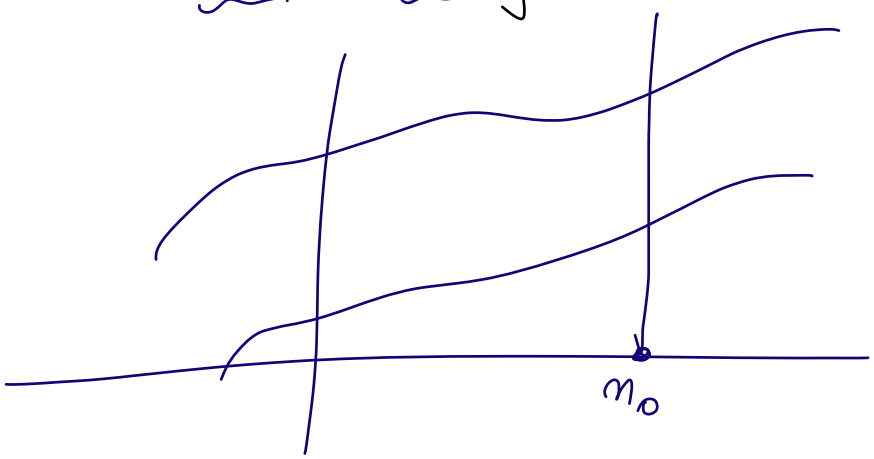
(* Const) → mult

Big-Oh

$$O(f(n)) = \{ g(n) \mid \exists n_0, \forall n > n_0, \exists c, g(n) \le c \cdot f(n) \}$$

↓

set of all function that are **at-most** as time taking as $f(n)$ upper bound



$$O(\lg n) << O(n) << O(n^2) << O(2^n) << O(n!) << O(n^n)$$

$$O(\lg n^{10}) < O(\lg n^{11})$$

$$O(n^{2 \cdot \lg n}) < O(n^{2 \cdot n})$$

② isn't this bad practically? → yes!

$A_1 \rightarrow O(1) \rightarrow 10^{100}$ steps const

$A_2 \rightarrow O(n^2)$ ⇝ computer ~ 10^8 operation/second.

1 sec to operate

$n^2 > 10^{100}$

$n \geq 10^{50}$ 10,000 BRRBR

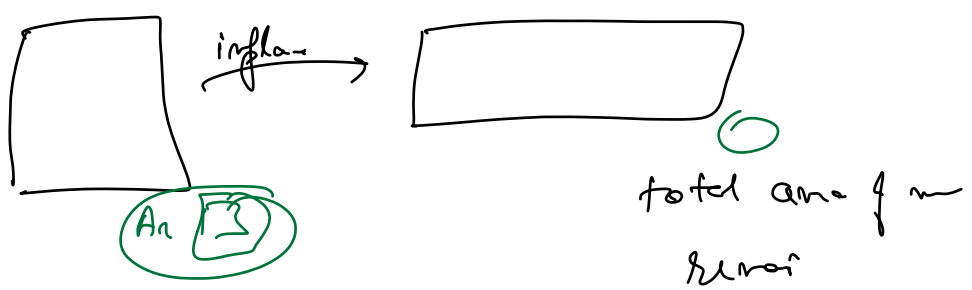
Adv. DSA

Maths — Comb
Mod
Prime
Bit manip

Array — 1D
2D

Slide
2 pointer

Dynamic Program → Recursion + caching.



Interviewer tells you that $N = 10^5$

↳ $O(n^2)$ ⟶ ✓ is

↳ should you solve it ⟶ yes.

---

assumption ⟶ Know coding

↳ learn problem solving

Adv DSA

Beg / Intern

no Know

loop / if-else / var

algo ⟶ code

↓ ↓

thinking speaking a lang

Intern start after Adv DSA

↳ 3

Int ⟶ 5

Beg ⟶ 6.

Academy

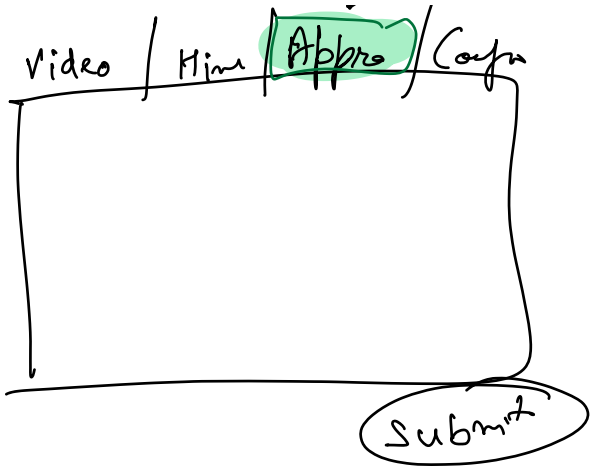
DS ML

DSML

avg 270%

Salary inc

10 LPA ⟶ 27 LPA


Video
Approve
Submit