Space Complexity → Rate of growth of space wrt i/p.

int → 4 Bytes

long → 8 Bytes

```
{  int x = 2;    → 4B
   long y = x*2; → 8B
}
```

Total memory = 12 B → O(1)

Find the Space Complexity [Big(O)] of the below program.

```
func(int N) { // 4 bytes → i/p
    int arr[10]; // 40 Bytes
    int x; // 4 bytes
    int y; // 4 bytes
    long z; // 8 bytes
    int[] arr = new int[N]; // 4 * N bytes
}
```

Total = (54 + 4N) Bytes

SC = O(N)

Find the Space Complexity [Big(O)] of the below program.

```
func(int N) { // 4 bytes → i/p
    int x = N; // 4 bytes
    int y = x * x; // 4 bytes
    long z = x + y; // 8 bytes
    int[] arr = new int[N]; // 4 * N bytes
    long[][] l = new long[N][N]; // 8 * N * N bytes
}
```

Total = (16 + 4N + 8N²) B

SC = O(N²)

Input → Algorithm → Output

Space apart from i/p & o/p.

Find the Space Complexity [Big(O)] of the below program.

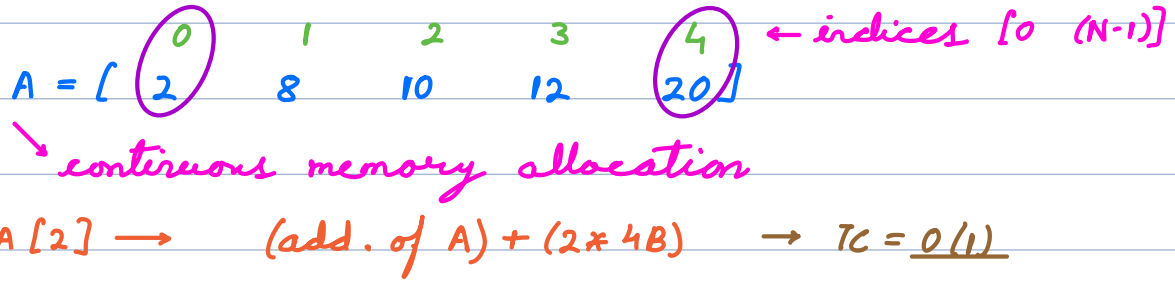
```
int maxArr(int arr[], int N) {   → 4B
    int ans = arr[0];            → N*4 Bytes
    for(i from 1 to N-1) {       → 4B
        ans = max(ans, arr[i]);  ← 4B
    }
    return ans;
}
```

i/p

o/p

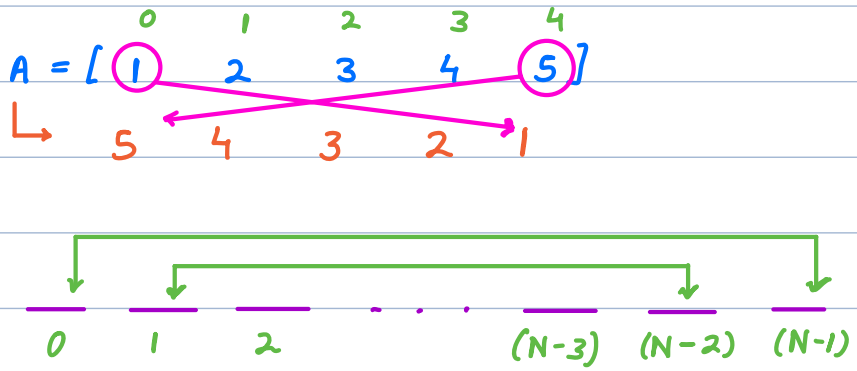
4B → SC = O(1)

Introduction to Arrays




```
for i → 0 to (N-1) {
    print (A[i])
}
```

Q → Reverse the given integer array.



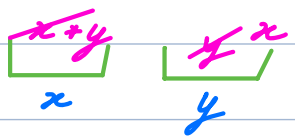
2 Pointers

i = 0 j = N-1

```
while (i < j) {
    t = A[i]
    A[i] = A[j]
    A[j] = t
    i++
    j--
}
```

Swap x, y without 3th variable

```
x = x + y
y = x - y
x = x - y
```



TC = O(N) SC = O(1)

Q → Reverse the array from index l to r.

	0	1	2	3	4
A =	1	2	3	4	5
		3	2		

l = 1 r = 2

```
i = l        j = r
while (i < j) {
    t = A[i]
    A[i] = A[j]
    A[j] = t
    i++
    j--
}
```

TC = O(N) SC = O(1)

Q → Rotate the given array from right to left (clockwise) K times.

	0	1	2	3	4	5	6	7
A =	9	5	8	2	3	1	7	8

K =								
1	8	9	5	8	2	3	1	7
2	7	8	9	5	8	2	3	1
3	1	7	8	9	5	8	2	3

Bruteforce → K = K % N

```
for k → 1 to K {
    t = A[N-1]
    for i → (N-2) to 0 {   ←
        A[i+1] = A[i]
    }
    A[0] = t
```

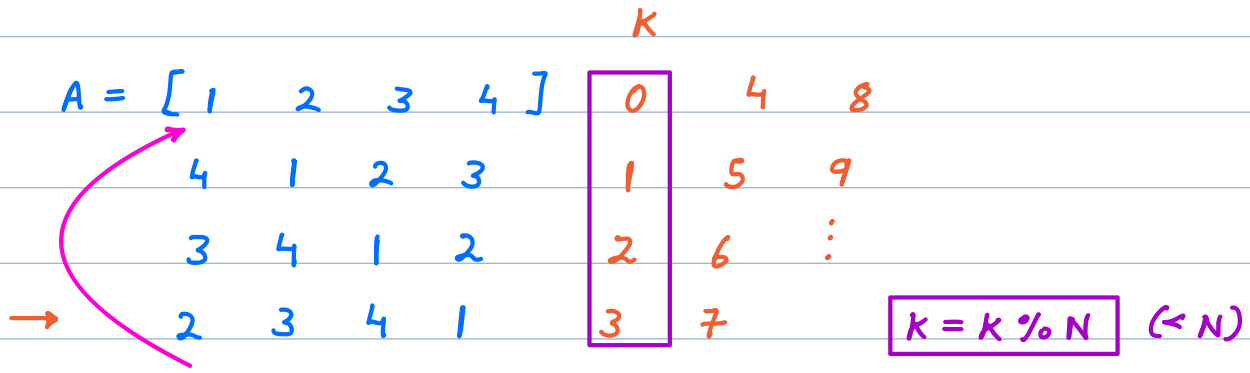
}

TC = O(K * N) SC = O(1)

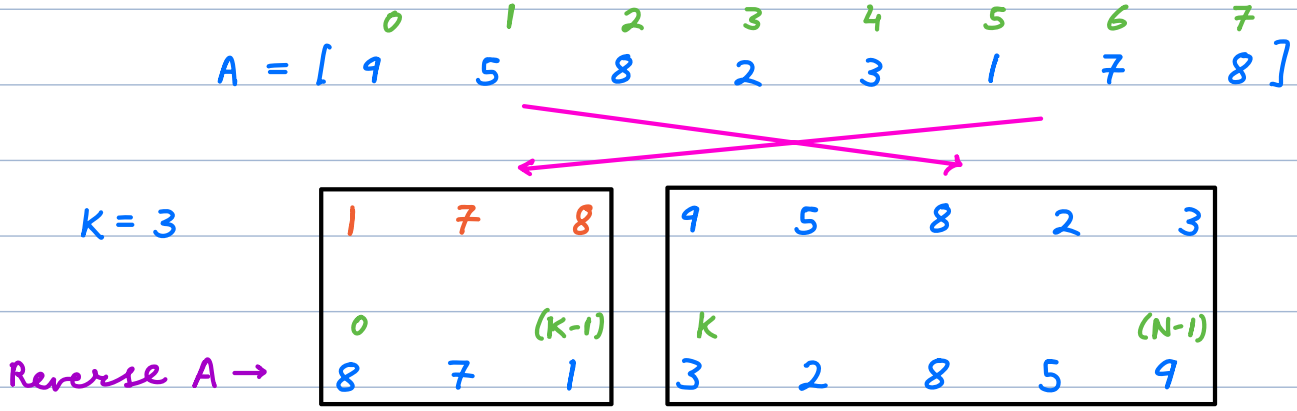
< O(N²)

TC / SC → wrt input (not always N)

K = 15 N = 4 → K = 15 % 4 = 3 ✓



SC = O(1)



Steps → K = K % N TC

1) reverse (A, 0, N-1) → O(N)

2) reverse (A, 0, K-1) → O(K)

3) reverse (A, K, N-1) → O(N-K)

Total TC = O(N + K + N - K) = O(2N) = O(N)

SC = O(1)

Dynamic Array

`int a[] = new int[10];` // 10 length integer array

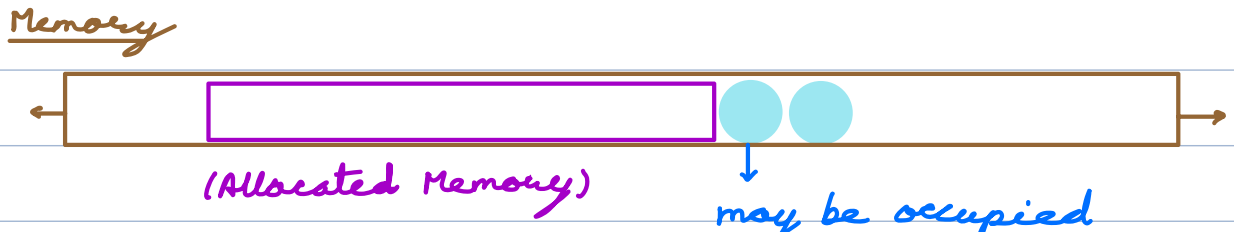
Static array → pre-defined length

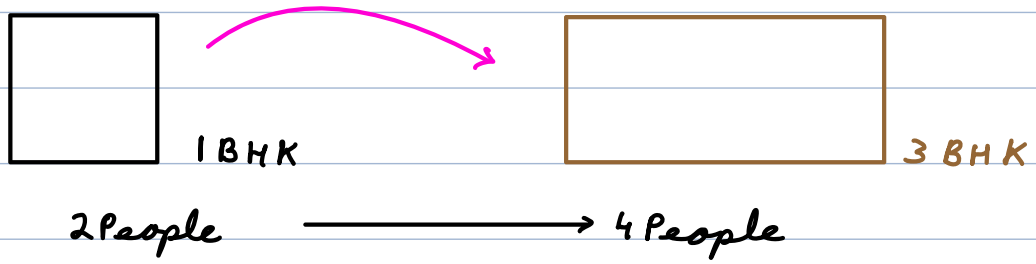
Sometimes defining length at the start might not be possible → dynamic array

↳ Eg → Java ArrayList, C++ vector, etc.

Access → TC = O(1)

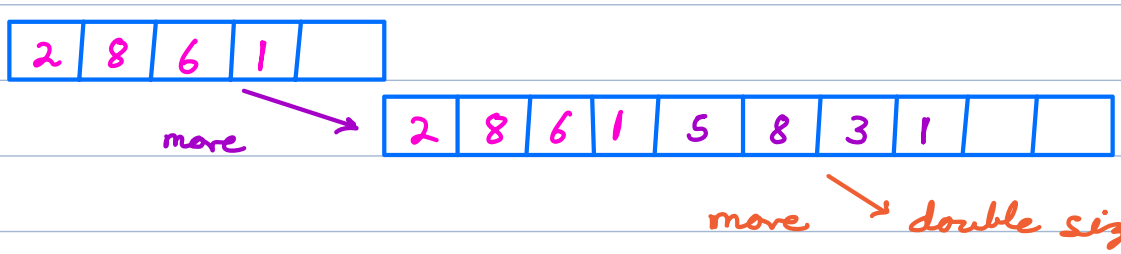
⇒ continuous memory allocation





Load factor (0 1) = 0.8 (eg.)

⇒ if 80% memory is occupied find a double size memory.



if ((current size * load factor) is full)
find another location with double size.

if copy K elements ⇒ insert K more elements

copy + insert

⇒ TC to insert K element = O(2*K)

⇒ TC to insert 1 element = O(2) → O(1)

Q → Given an array with daily profit / loss from a particular stock. Calculate the total profit / loss over a given range of days.
The function should be able to handle multiple queries for different ranges.

0	1	2	3	4	5	6
-5	10	20	40	50	-10	8

A = [-5 10 20 40 50 -10 8]

Queries

Start	End	
1	4	→ 120
2	5	→ 100
0	0	→ -5

Given an integer array with N elements & Q queries, for every find sum of elements from index L to R.

0	1	2	3	4	5	6
-5	10	20	40	50	-10	8

A = [-5 10 20 40 50 -10 8]

Queries

L	R	
1	4	→ 120
2	5	→ 100
0	0	→ -5

Range Sum Queries

Bruteforce → ∀ Queries, iterate from index L to R.

TC = O(N * Q)

SC = O(1)

```
for i → 0 to (Q-1) {
    // L, R
    for i → L to R {
```

Sum += A[i]

}

}

Cricket

Over →	0	1	2	3	4	5	6	7	8	9	10
Score →		2	8	14	29	31	49	65	79	88	97

Runs scored in 7^{th} over → 65 − 49 = 16

Runs scored from 6^{th} to 10^{th} over → 97 − 31 = 66

Runs scored in 10^{th} over → 97 − 88 = 9

Runs scored from 3^{rd} to 6^{th} over → 49 − 8 = 41

Runs scored from 4^{th} to 9^{th} over → 88 − 14 = 74

Prefix Sum → $P[i] = A[0] + A[1] + \ldots + A[i]$

$P[2] = A[0] + A[1] + A[2]$

$P[3] = A[0] + A[1] + A[2] + A[3]$

$= P[2] + A[3]$

$P[i] = P[i-1] + A[i]$

$P[0] = A[0]$

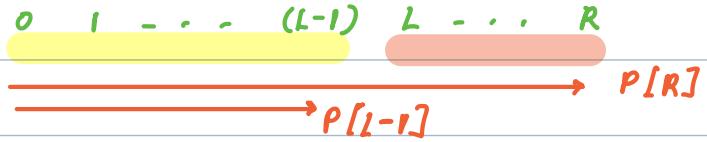
```
P[0] = A[0]
for i → 1 to (N-1) {
    P[i] = P[i-1] + A[i]
}
```

TC = O(N)

	0	1	2	3	4	5
A = [	10	32	6	12	20	1]
✓ P → [	10	42	48	60	80	81]

Sum (L to R)

P[R] − P[L−1]



sum (0 — 3) → P[3] - P[0-1]

index out of bound error

```
for i → 0 to (Q-1) {
    // L , R
    if (L == 0)  print (P[R])
    else  print (P[R] - P[L-1])
}
```

TC = O(Q)

TC = O(N + Q) SC = O(N)

O(1) → Use A[] to store prefix sum.

~~A[0] = A[0]~~

```
for i → 1 to (N-1) {
    A[i] = A[i-1] + A[i]
}
```

	0	1	2	3	4	5
A = [	10	~~32~~ 42	~~6~~ 48	~~12~~ 60	~~20~~ 80	~~1~~ 81]

A[i] = P[i] - P[i-1]

Q → Given an integer array & Q queries (L, R)
Find the sum of all even indexed elements.

	0	1	2	3	4	5
A = [	2	3	1	6	4	5]

Queries

1 3 → 1

2 5 → 5

3 3 → 0

Bruteforce → TC = O(N * Q)

SC = O(1)

P[i] → A[0] + A[2] + A[4] . . .

	0	1	2	3	4	5
A = [	2	3	1	6	4	5]
P →	2	2	3	3	7	7

	0	1	2	3	4
A = [	2	4	3	1	5]
P →	2	2	5	5	10

```
P[0] = A[0]
for i → 1 to (N-1) {
    if (i % 2 == 0)    P[i] = P[i-1] + A[i]
    else               P[i] = P[i-1]
}
```

```
for i → 0 to (Q-1) {  // Queries → L = query[i][0]
    // L, R                       R = query[i][1]
    if (L == 0)  print (P[R])
    else         print (P[R] - P[L-1])
}
```

TC = O(N + Q) SC = O(N) → O(1)

Find the sum of all odd indexed elements.

```
P[0] = 0
for i → 1 to (N-1) {
    if (i % 2 == 1)    P[i] = P[i-1] + A[i]
    else               P[i] = P[i-1]
}
```

Q → Given an integer array, count the no. of special index i.e. index after removing which sum of all even indexed elements = sum of all odd indexed elements.

i	A = [3 (0)	2 (1)	7 (2)	6 (3)	-2 (4)]	Seven	Sodd
0 →	2	7	6	-2		8	≠ 5
1 →	3	7	6	-2		9	≠ 5
2 →	3	2	6	-2		9	≠ 0
3 →	3	2	7	-2		10	≠ 0
4 →	3	2	7	6		10	≠ 8

Ans = 0

A = [4 (0) 1 (1) ~~3~~ (2) 7 (3) 10 (4)]

7 10

odd sum = 1 + 10 = 11

A = [2 (0) 3 (1) 1 (2) ~~4~~ (3) 0 (4) -1 (5) 2 (6) -2 (7) 10 (8) 8 (9)]

0 -1 2 -2 10 8

odd sum = 3 + 0 + 2 + 10 = 15

∀ elements > i^{th} index → odd index move to even
even index move to odd

A = [2 (0) 3 (1) 1 (2) ~~4~~ (3) 0 (4) -1 (5) 2 (6) -2 (7) 10 (8) 8 (9)]

0 -1 2 -2 10 8

even sum = 2 + 1 + (-1) + (-2) + 8 = 8

check i^{th} index

Seven = S even (0 — i-1) + S odd (i+1 — N-1)

Sodd = S odd (0 — i-1) + S even (i+1 — N-1)

$Sum_{even} = evenP[i-1] + (oddP[N-1] - oddP[i])$

$Sum_{odd} = oddP[i-1] + (evenP[N-1] - evenP[i])$

```
// calculate oddP[] & evenP[]
cnt = 0
for i → 0 to (N-1) {
    if (i == 0) {
        se = oddP[N-1] - oddP[i]
        so = evenP[N-1] - evenP[i]
    } else {
        se = evenP[i-1] + (oddP[N-1] - oddP[i])
        so = oddP[i-1] + (evenP[N-1] - evenP[i])
    }
    if (se == so) cnt++
}
return cnt
```

TC = O(N) SC = O(N)

SCALER

Array - Carry forward & Subarrays

TABLE OF CONTENTS

1. Count ‘a-g’ pairs
2. Sub-arrays
3. Print a Subarray
4. Print all Subarrays
5. Min-Max

Notes

Count 'a-g' pairs

< *Question* > : Given a string s of lowercase characters, return the count of pairs (i, j) such that i < j and s[i] is 'a' and s[j] is 'g'.

s = "a b e g a g"
0 1 2 3 4 5

Ans = 3

i	j
0	3
0	5
4	5

s = "a c g d g a g"
0 1 2 3 4 5 6

Ans = 4

i	j
0	2
0	4
0	6
5	6

s = "b c a g g a a g"
0 1 2 3 4 5 6 7

Ans = 5

i	j
2	3
2	4
2	7
5	7
6	7

BF Idea

```
cnt = 0
for i → 0 to (N-1) {
    for j → 0 to (N-1) {
        if (i < j && s[i] == 'a' && s[j] == 'g')
            cnt++
    }
}
} return cnt
```

TC = O(N²) SC = O(1)

```
cnt = 0
for i → 0 to (N-2) {
    if (s[i] == 'a') {
        for j → i+1 to (N-1) {
            if (s[j] == 'g')
                cnt++
        }
    }
}
} return cnt
```

count # 'g' from (i+1) to (N-1).

ans += cnt_g[i+1]

TC = O(N²) → O(N) SC = O(1) → O(N)

---

s = " b c a g g a a g "
0 1 2 3 4 5 6 7

cnt_g → [3 3 3 3 2 1 1 1]

L ← R

ans = 0 + 3 + 1 + 1 = 5

```
∀i, if (s[i]=='g')
      cnt_g[i] = cnt_g[i+1]+1
    else  // s[i] ≠ 'g'
      cnt_g[i] = cnt_g[i+1]
```

Idea Carry Forward → calculate & use

	0	1	2	3	4	5	6	7
str →	b	c	a	g	g	a	a	g

L ⟵ R

'g' → cnt 3 3 3 3 2 1 1 1

'a' → ans = 0 + 1 + 1 + 3 = 5

```
cnt = 0
ans = 0
for i → (N-1) to 0 {
    if (s[i] == 'g') cnt++ // calculate
    if (s[i] == 'a') ans += cnt // use
}
```

TC = O(N) SC = O(1)

i < j s[i] = 'a' s[j] = 'g'

count #'a' from left to right

0 —— i

cnt = 0 ↗

ans = 0

c a a g x g

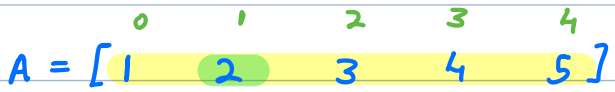
cnt = + 2

ans = 2 + 2 = 4

```
for i → 0 to N-1 {
    if (s[i] == 'a') cnt++ // calculate
    if (s[i] == 'g') ans += cnt // use
}
```

TC = O(N) SC = O(1)

Subarrays → continuous part of array



single element ✓

complete array ✓

Example: arr[ ] → [2 4 1 6 -3 7 8 4]

a. [1, 6, 8]

b. [1, 4]

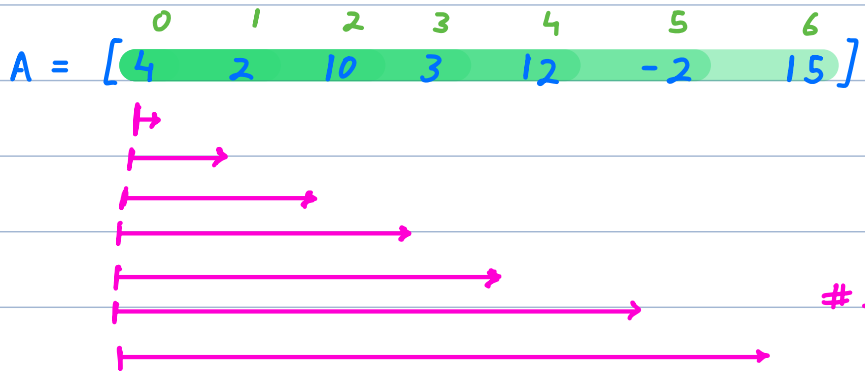
c. [6, 1, 4, 2]

d. [7, 8, 4,] ✓

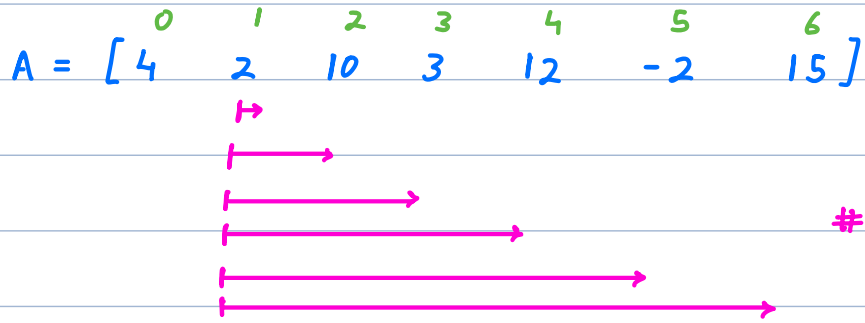
Representation of a subarray

1) L R — start & end

2) L (length) — start & length



subarrays starting from index 0 = N



Total number of subarrays

subarrays starting from ↓ = ↓

0	N
1	N-1
2	N-2
⋮	
(N-1)	1

N * (N+1) / 2

< Question > : Given an array, si and ei. Print from si to ei. $si \le ei$

arr → [4 2 10 3 12 -2 15] si = 2, ei = 5

0 1 2 3 4 5 6

o/p → 10 3 12 -2

- void printSubarray(arr, si, ei) {

```
for i → si to ei {
    print ( arr[i] )
}
```

}

print 1 subarray → T.C O(N)

SC = O(1)

< Question > : Print all the possible sub-arrays of the given array.

[5, 7, 3, 2]

0 1 2 3

$\frac{N * (N+1)}{2} * N \rightarrow O(N^3)$

8:30 AM

O/P - [5]

[5, 7]

[5, 7, 3]

[5, 7, 3, 2]

[7]

[7, 3]

[7, 3, 2]

[3]

[3, 2]

[2]

Idea Consider all the subarrays & print Subarray( )

```
for st → 0 to (N-1) {
    for end → st to (N-1) {
        for i → st to end {
            print (A[i])
        } print ('\n')
    }
}
```

TC = $O(N^3)$ SC = $O(1)$

Min Max

< *Question* > : Given an array of N integers, return the length of smallest subarray which contains both maximum and minimum elements of the array.

$1 \le N \le 10^6$

arr[ ] → [2 2 6 4 5 1 5 2 6 4 1]

0 1 2 3 4 5 6 7 8 9 10

Ans = 3

arr[ ] → [1 2 3 1 3 4 6 4 6 3]

Ans = 4

0 1 2 3 4 5 6 7 8 9

arr[ ] → [8 8 8 8 8 8]

0 1 2 3 4 5

min = 8 max = 8 Ans = 1

Bruteforce → Find min & max of the array → O(N)

∀ subarrays, check if it contains min & max and take smallest length subarray as answer.

$\frac{N*(N+1)}{2}$ → O(N)

TC = O(N + N^3) = $O(N^3)$ SC = O(1)

Observation

1. There must be exactly one occurrence of min & max element.

[min min - - - - max]

2. Min and max elements should be the end point of subarray.

3. ✓ ∀max check closest min on left

case-1: [min - - - - max] → ∀min check closest max on right

case-2: [max - - - - min]

✓ ∀min check closest max on left

∀max check closest min on right

	0	1	2	3	4	5	6	7	8	9	10
arr[ ] → [	2	2	6	4	5	1	5	2	6	4	1]
	i	i	i	i	i	i	i	i	i	i	i

min = 1

max = 6

latest_min_index = ~~-1~~ 10

latest_max_index = ~~-1~~ 8

length = 10 - 8 + 1 = 3

ans = ~~4~~ 3

< / > Code

```
minA = A[0]        maxA = A[0]
for i → 1 to (N-1) {
    minA = min (minA, A[i])
    maxA = max (maxA, A[i])
}
l_min = -1         l_max = -1
ans = N
for i → 0 to (N-1) {
    if (A[i] == minA) {
        l_min = i          // calculating
        if (l_max != -1) {
            ans = min (ans, i - l_max + 1)
        }                          // use
    }                    [l_max    i]
    if (A[i] == maxA) {
        l_max = i          min ——— max
        if (l_min != -1) {  [l_min ——— i]
            ans = min (ans, i - l_min + 1)
        }
    }
}
return ans
```

TC = O(N) SC = O(1)

Q → Given an integer array, find the total sum of all possible subarrays.

A = [3 2 5] (indices 0 1 2)

3 → 3
3 2 → 5
3 2 5 → 10
2 → 2
2 5 → 7
5 → 5

32 (Ans)

A = [2 7] (indices 0 1)

2 → 2
2 7 → 9
7 → 7

18 (Ans)

Bruteforce → ∀ subarrays, calculate sum & add it in the answer.

```
ans = 0
for L → 0 to (N-1) {                    L — R
    for R → L to (N-1) {
        sum = 0
        for i → L to R {                subarray sum L — R
            sum += A[i]
        }
        ans += sum
    }
} return ans
```

TC = O(N^3)

SC = O(1)

Prefix Sum →

```
P[0] = A[0]
for i → 1 to (N-1) {
    P[i] = P[i-1] + A[i]
}

ans = 0
for L → 0 to (N-1) {          L — R
    for R → L to (N-1) {
        if (L == 0)   ans += P[R]
        else          ans += P[R] - P[L-1]
    }
} return ans
```

TC = $O(N + N^2) = O(N^2)$ SC = $O(N)$

A = [3 (0) 2 (1) 5 (2)]

L = 0:
- 0 — 0 : A[0] → ans += 5
- 0 — 1 : A[0] + A[1] → ans += 5
- 0 — 2 : A[0] + A[1] + A[2] → ans += 5

L = 1:
- A[1] → ans += 5
- A[1] + A[2] → ans += 5

L = 2:
- A[2] → ans += 5

Carry Forward

```
ans = 0
for L → 0 to (N-1) {
    sum = 0
    for R → L to (N-1) {
        sum += A[R]
        ans += sum
```

}

} return ans TC = O(N²) SC = O(1)

→ If one element is used multiple times to calculate the answer → Contribution Technique

$$Ans = \sum_{\forall i} \text{contribution of } A[i]$$

A = [3 2 5] (0 1 2)

3 → 3

3 2 → 5 3 * 3 = 9

3 2 5 → 10 2 * 4 = 8

2 → 2 5 * 3 = 15

2 5 → 7 32 (Ans)

5 → 5

32 (Ans)

Contribution of A[i] = A[i] * (# subarrays A[i] is part of)

A = [3 -2 4 -1 2 6] (0 1 2 3 4 5)

Ans = 10

L	R
0	1
0	2
0	3
0	4
0	5

L	R
1	1
1	2
1	3
1	4
1	5

i

A = [3 -2 4 -1 2 6] (0 1 2 3 4 5)

Ans = 12

L	R
0	2
0	3
0	4
0	5

L	R
1	2
1	3
1	4
1	5

L	R
2	2
2	3
2	4
2	5

subarrays with A[i]

starting index (L) = [0 i] → i - 0 + 1 = i + 1

ending index (R) = [i N-1] → N-1-i+1 = N-i

subarrays having A[i] = (i+1) * (N-i)

$$Ans = \sum_{\forall i} A[i] * (i+1) * (N-i)$$

```
ans = 0
for i → 0 to (N-1) {
    ans += A[i] * (i+1) * (N-i)
} return ans
```

TC = O(N) SC = O(1)

A = [3 2 5] (i = 0, 1, 2)

Ans = 3 * 1 * 3 +

2 * 2 * 2 +

5 * 3 * 1 = 9 + 8 + 15 = 32 ✓

Q → Find the # of subarrays with length K.

A = [3 -2 4 -1 2 6] (0 1 2 3 4 5)

N = 6

K = 3 → Ans = 4

K = 5 → Ans = 2

1st subarray → 0 ___ (K-1)

Last subarray → (N-K) ___ (N-1)

[st (N-1)]

N - ~~1~~ - st + ~~1~~ = K ⇒ st = N-K

values [(K-1) ____ (N-1)] → (N-1)-(K-1)+1 = N-K+1 (Ans)

N = 7 K = 4 Ans = N-K+1 = 7-4+1 = 4

Q → Print all start & end indices of length K.

	0	1	2	3	4	5
A = [	3	-2	4	-1	2	6]

N = 6 K = 3

L	R
0	2
1	3
2	4
3	5

```
for L → 0 to (N-K) {
    R = L + (K-1)
    print (L, R)
}
```

TC = O(N) SC = O(1)

Q → Given an integer array, find max subarray sum of subarrays of length K.

	0	1	2	3	4	5
A = [	3	-2	4	-1	2	6]

N = 6 K = 3

L	R	
0	2	5
1	3	1
2	4	5
3	5	7

[L R]

R - L + 1 = K

⇒ R = K - 1 + L

Ans = 7

Bruteforce

```
ans = Int_Min
for L → 0 to (N-K) {
    R = L + (K-1)
    Sum = 0
    for i → L to R {
        Sum += A[i]
    }
    ans = max(ans, sum)
}
```

Inner loop → TC = O(K)

SC = O(1)

TC = O((N-K+1) * K) → O(N²)

K = 1 : (N-1+1) * 1 = N

K = N : (N-N+1) * N = N

K = N/2 : (N - N/2 + 1) * N/2 = (N/2 + 1) * N/2 = N²/4 + N/2 → O(N²)

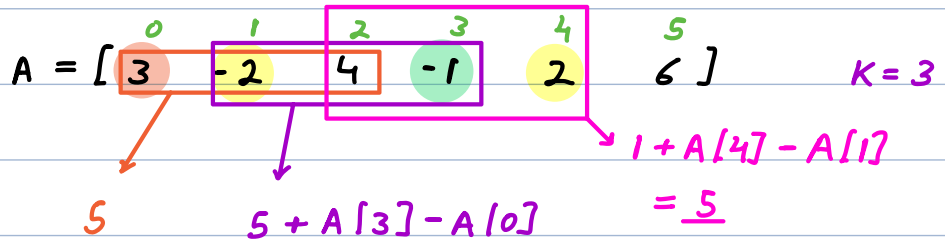
Prefix Sum

```
P[0] = A[0]
for i → 1 to (N-1) {
    P[i] = P[i-1] + A[i]
}

ans = Int_Min
for L → 0 to (N-K) {
    R = L + (K-1)
    if (L == 0) ans = max(ans, P[R])
    else ans = max(ans, P[R] - P[L-1])
}
```

return ans TC = O(N + (N-K+1)) → O(N)

SC = O(N)



Sliding Window
↘ Fixed length subarray

L	R	sum
0	2	A[0] + A[1] + A[2] = 5
1	3	sum + A[3] - A[0] = 1
2	4	sum + A[4] - A[1] = 5
3	5	sum + A[5] - A[2] = 7

max → 7 (Ans)

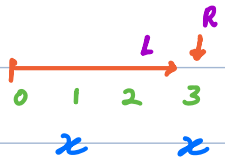
```
sum = 0
for i → 0 to (K-1) {
    sum += A[i]
}
ans = sum
for R → K to (N-1) {
    L1 = R - K
    sum += A[R] - A[L1]
    ans = max(ans, sum)
}
```

R - L + 1 = K ⇒ L = R - K + 1

L —— R

return ans TC = O(K + N - K) = O(N)

SC = O(1)



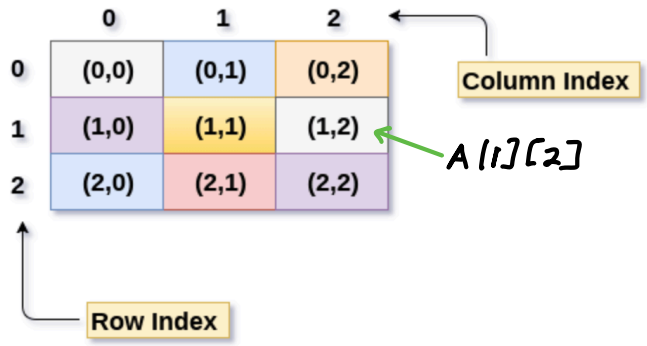
	0	1	2	3	4	5
A = [	3	2	4	2	3	4]
				L		R

vst [3] = true

Ans = $\sum_{\forall R}$ # subarrays ending at R

1 + 2 + 3 + 2 + 3 + 3 = 14

R - L + 1



A[N][M]

#rows #columns

Top right cell → A[0][M-1]

Bottom right cell → A[N-1][M-1]

Q → Given a 2D matrix, print row-wise sum.

A =

	0	1	2	3	O/P
0	1	2	3	4	→ 10
1	5	6	7	8	→ 26
2	9	10	11	12	→ 42

```
for i → 0 to (N-1) {
    sum = 0
    for j → 0 to (M-1) {
        sum += A[i][j]
    }
    print (sum)
}
```

TC = O(N * M) SC = O(1)

Q → Given a 2D matrix, find max of every column.

A =

	0	1	2	3
0	5	12	3	1
1	5	6	7	8
2	10	8	2	19

o/p → 10 12 7 19

```
for i → 0 to (M-1) {   // column
    m = A[0][i]
    for j → 1 to (N-1) {   // row
        m = max (m, A[j][i])
    }
    print (m)
}
```

TC = O(N * M) SC = O(1)

```
ArrayList < Integer > al[] = new ArrayList <>[N];
for i → 0 to (N-1) {
    al[i] = new ArrayList <>();   // al[i].get(j)
}
```

```
ArrayList < ArrayList <Integer>> al = new ArrayList <>();
```

Q → Given a 2D square matrix, print diagonals.

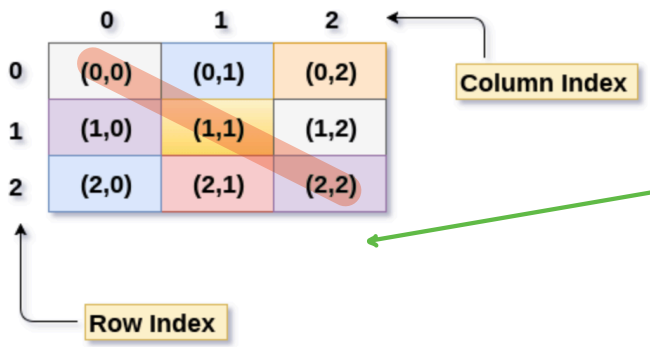
1. **Principal Diagonal:** From top left to bottom right.
2. **Anti Diagonal:** From top right to bottom left.

1	5	8
4	3	1
6	5	2

Principal Diagonal

1	5	8
4	3	1
6	5	2

Anti-Diagonal



Principal Diagonal

(i == j)

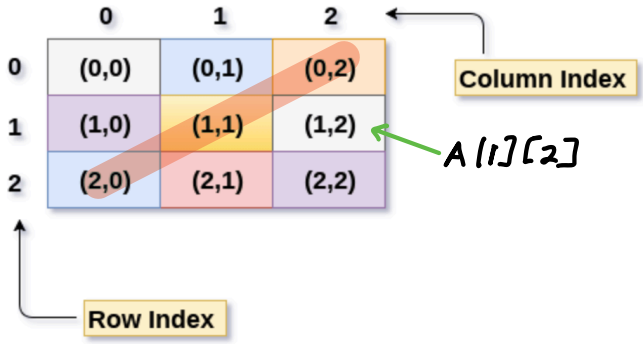
```
for i → 0 to (N-1) {
    print (A[i][i])
}
```

TC = O(N) SC = O(1)

2) Anti - Diagonal

r++ c--

(0 , (N-1))



0 + 2 = 2

1 + 1 = 2

2 + 0 = 2

$i + j = N-1$

$\Rightarrow j = N-1-i$

```
for i → 0 to (N-1) {
    print (A[i][N-1-i])
}
```

TC = O(N) SC = O(1)

Q → Given a 2D matrix, print all elements diagonally from right to left.

A =

	0	1	2	3
→0	5	12	3	1
1	5	6	7	8
2	10	8	2	19

o/p → 5

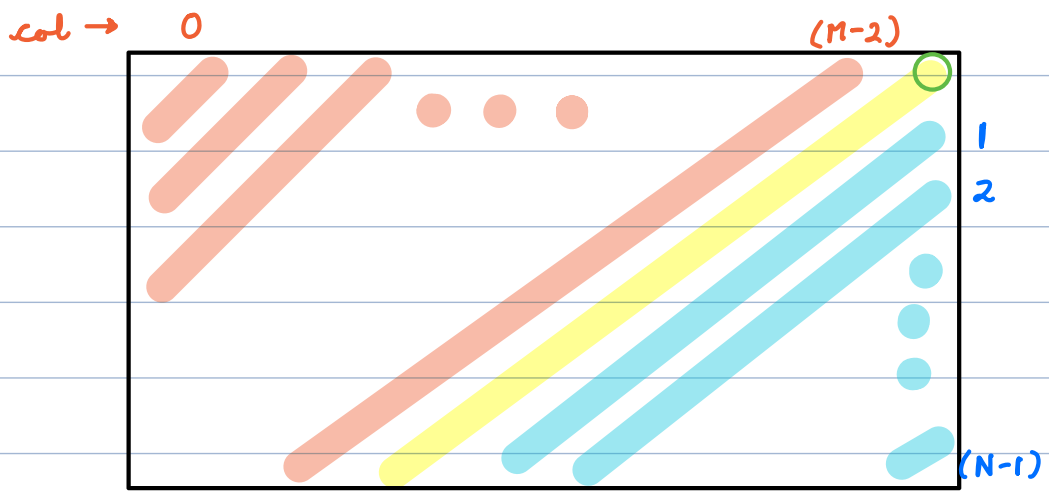
12 5

3 6 10

1 7 8

8 2

19



[0 (M-2)] → M-2-0+1 = M-1

[1 (N-1)] → N-1-1+1 = N-1

Total diagonals = M-1 + N-1 + 1 = M+N-1

(4*5) → 4+5-1 = 8

```
for c → 0 to (M-1) {
    i=0   j=c
    while (i<N && j>=0) {
        print(A[i][j])
        i++   j--
    } print("\n")
}
for r → 1 to (N-1) {
    i=r   j=M-1
    while (i<N && j>=0) {
        print(A[i][j])
        i++   j--
    } print("\n")
}
```

c = 0 1 2 3

i = 0 1 2 3

j = 3 2 1 0

r = 1 2

i = 2 3

j = 1 2

A =

	0	1	2	3
0	5	12	3	1
1	5	6	7	8
2	10	8	2	19

TC = O(N*M) SC = O(1)

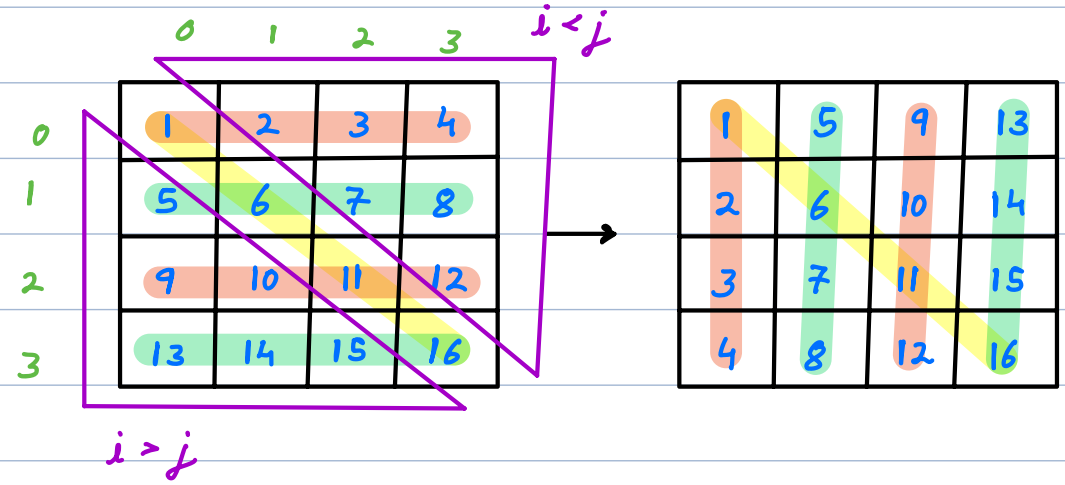
```
for i → 1 to N {
    for j → 1 to 10 {
        ...
    }
}
```

TC = O(N * 10) → O(N)

```
for c → 0 to (M-1) {
    i = 0   j = c
    while (i < N && j >= 0) {
        print (A[i][j])
        i++   j--
    } print ("\n")
}
```

c	while	# iterations
0	0 — N-1	1
	0 — 0	
1	0 — N-1	2
	1 — 0	
⋮		
3		

Q → Find transpose of given square matrix.



1) row ⟷ column i, j ⟷ j, i

2) no change in principal diagonal elements.

swap (x, y) → t = x

x = y

y = t

```
for i → 0 to (N-2) {
    for j → i+1 to (N-1) {
        swap ( A[i][j], A[j][i] )
    }
}
```

i = 0 j = 1

(0, 1) ⟷ (1, 0)

i = 1 j = 0

(1, 0) ↔ (0, 1)

#iterations

Swap twice ⇒ no swap

(N-1) + (N-2) + (N-3) + . . . 1

$= \frac{(N-1) * (N)}{2}$

TC = O(N²) SC = O(1)

Q → Rotate the given sq matrix 90° clock wise.

90° ↷

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

→

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

↓ transpose

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

Reverse Row →

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

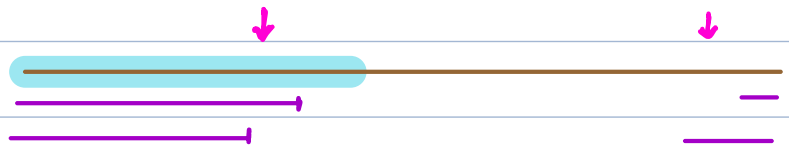
Sol → 1) Transpose the matrix
2) Revere every row
} 90° rotation

$TC = O(N^2 + N^2) = \underline{O(N^2)}$ $SC = \underline{O(1)}$

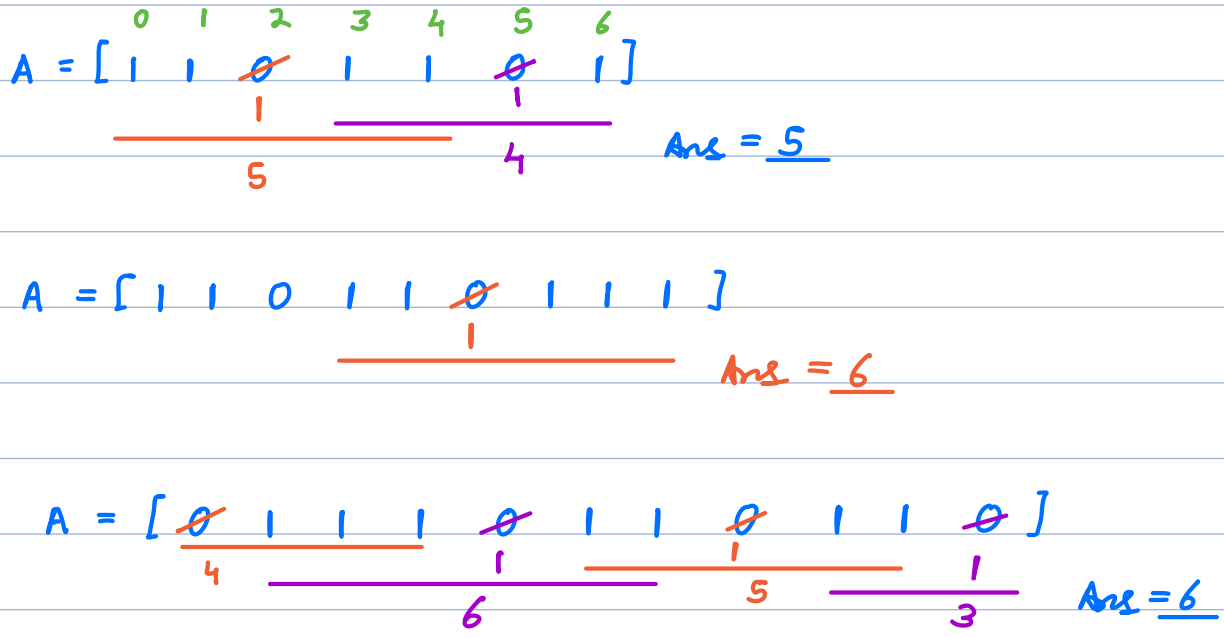
---

a.length → row length

a[0].length → col length



Q→ Given a binary array of 1 & 0.
Find max # of consecutive 1's that can be obtained after replacing atmost one 0 with 1.



Sol→ ∀ 0's, check max consecutive 1's we can have by updating this 0 with 1.

count 1's on left & right.

```
one = 0
for i → 0 to (N-1) {
    if (A[i] == 1)   one++
}
if (one == N)  return N

ans = 0
for i → 0 to (N-1) {
    if (A[i] == 0) {
        l = 0
```

```
        for j → (i-1) to 0 {   // for(j=i-1; j>=0; j--)
        |   if (A[j] == 1)  l++
        |   else break
        }

        r = 0
        for j → (i+1) to (N-1) {
        |   if (A[j] == 1)  r++
        |   else break
        }

        ans = max(ans, l + 1 + r)
    }
}
return ans
```

TC = O(3N) = O(N) SC = O(1)

A = [0 1 1 1 0 1 1 0 1 1 0]

Ans = 6

Q → Find max consecutive 1's by swapping atmost a 0 with 1.

A = [0 1 1 1 0 1 1 0 1 0]

Ans = 6

A = [1 1 0 1 1 0 1] (indices 0 1 2 3 4 5)

Ans = 5

```
one = 0
for i → 0 to (N-1) {
    if (A[i] == 1)  one++
}
if (one == N)  return N
```

```
ans = 0
for i → 0 to (N-1) {
    if (A[i] == 0) {
        l = 0
        for j → (i-1) to 0 {   // for (j = i-1; j >= 0; j--)
            if (A[j] == 1) l++
            else break
        }
        r = 0
        for j → (i+1) to (N-1) {
            if (A[j] == 1) r++
            else break
        }
        ans = max (ans, l + 1 + r)
    }
}
```

TC = O(3N) = O(N) SC = O(1)

```
if (ans > one) ans--
return ans
```

Q → Given an integer array find majority element i.e element with frequency > $\frac{N}{2}$.

A = [2 1 4] (0 1 2) Ans = -1

A = [2 2 4] Ans = 2

A = [3 4 3 2 4 4 4 4]

N = 8 freq(4) = 5 > $\frac{N}{2}$ Ans = 4

A = [4 6 5 3 4 5 6 4 4 4] Ans = -1

freq(4) = 5 ≯ $\frac{10}{2}$

Max # majority elements = 1

Bruteforce → ∀ A[i], calculate frequency & check if its > $\frac{N}{2}$.

TC = $O(N^2)$ SC = $O(1)$ → Two loops

TC = $O(N)$ SC = $O(N)$ → Precompute & store freq.

Observation → If we remove 2 distinct elements from the input then majority element remains same.

freq(majority element) > $\frac{N}{2}$	-1	0
freq(everything else) < $\frac{N}{2}$	-1	-2

A = [3 4 3 6 1 3 2 5 3 3 3] → 3 (Ans)

f = + 0 + 0 + 0 + 0 + x 3

m = ~~3~~ + x 3

A = [3 4 3 0 3 1 3]

f = + 0 + 0 + 0 1

m = 3

check freq(3) > N/2

```
m = -1
f = 0
for i → 0 to (N-1) {
    if (f == 0) {
        m = A[i]
        f++
    } else if (A[i] == m) f++
      else f--
}
```

0	1	2	3	4	5	6	7
1	3	1	8	8	8	1	3

A = [1 3 1 8 8 8 1 3]

f = 0 + 0 + 0 + 2 + 0

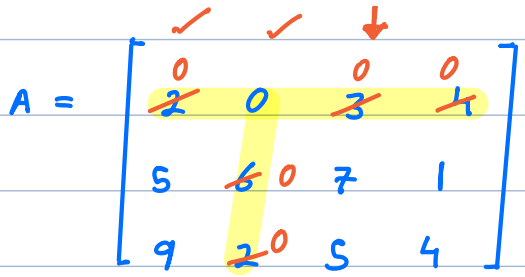
m = 8

```
f = 0
for i → 0 to (N-1) {
    if (A[i] == m) f++
}
if (f > N/2) return m
return -1
```

TC = O(N) SC = O(1)

Q → Given a 2D matrix of integers.
Update all elements in a row (i) & col (j) to 0 if A[i][j] = 0.

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 0 \\ 9 & 2 & 0 & 4 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



identify updated 0 vs original 0 ✓ -1

1) Update all row/col with (≠0) -1 where current ≠0.

2) Update all -1 to 0.

TC = O(N*M) SC = O(1)

SCALER

One Dimensional Array

TABLE OF CONTENTS

1. Max Subarray Sum
2. Prefix Sum
3. Rain Water Trapped

Stock

1	2	3	4	5	6	7
A = [-20	30	40	-10	50	-100	70]

Max profit in one trade → 30 + 40 - 10 + 50 = 110

< Question > : Given arr[N]. Find the maximum subarray sum out of all subarrays.

$1 \le N \le 10^5$

Example 1 : arr[ ] → [-3 , 2 , 4 , -1 , 3 , -4 , 3]

0 1 2 3 4 5 6

Ans = 8

Example 2 : arr[ ] → [4 , 5 , 2 , 1 , 6]

0 1 2 3 4

Ans = 18

Example 3 : arr[ ] → [-4 , -3 , -6 , -9 , -2]

Ans = -2

$x > y$

$-x < -y$

Bruteforce → ∀ subarray, calculate sum & take max.

$\frac{N*(N+1)}{2}$

travel → O(N)

Total TC = $O(N^3)$ SC = $O(1)$

O(1) → Prefix Sum ✓

Carry Forward ✓

```
ans = 0 (INT_MIN)                    // = A[0]
for i → 0 to (N-1) {
    sum = 0
    for j → i to (N-1) {             // i — j
        sum += A[j]
        ans = max(ans, sum)
    }
}
return ans
```

TC = $O(N^2)$ SC = $O(1)$

Observations

Case 1 — When all elements are positive

2	4	5	1	3

Ans = $\sum A[i]$ $\forall i$

Case 2 — When all elements are negative

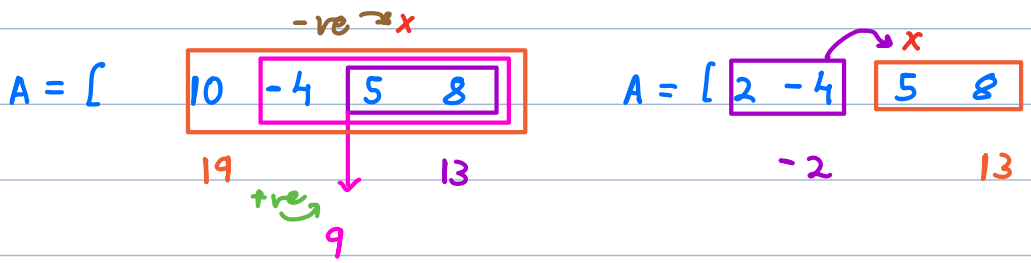
-7	-3	-11	-8	-15

Ans = $\max(A[i])$ $\forall i$

Case 3

	+ve	

Kadane's Algo



*Dry-Run

arr[ ] →	5	6	7	-3	2	-10	-12	8	12	-4	7	-2
	0	1	2	3	4	5	6	7	8	9	10	11

k = 0, l = 7, i = 11

sum = 5 11 18 15 17 7 -5 0 8 20 16 23 21

ans = 5 11 18 20 23

</> Code

```
ans = INT_MIN
sum = 0        l = 0
for i → 0 to (N-1) {
    sum += A[i]
    if (ans < sum) {
        ans = sum
        L = l      R = i
    }
    if (sum < 0) {
        sum = 0    l = i+1
    }
} return ans
```

Kadane's Algo

Subarray → L — R

	0	1	2 (i)
A =	-2	-1	-5

sum = 0 -2 0 -1 0 -5 0

ans = -2 -1

TC = O(N) SC = O(1)

Continuous Sum Query

< *Question* > : There are A beggars sitting in a row outside a temple. Each beggar has an empty pot initially. There are N devotees. Each devotee gives some fixed amount to beggars from indices l to r.

Given a 2-D Array B[N][3], where B[i][0] represents l [left index] l
B[i][1] represents r [right index] r
B[i][2] represents amount

Given an integer array, where every element is 0 initially. Update the array with multiple queries.
Query (l, r, x) → ∀ i → l to r A[i] += x

Example : A = 5 , B[N][3] →

l	r	Amount(x)
1	2	10
2	3	20
2	5	25

0	0	0	0	0
1	2	3	4	5

+10 +10
+20 +20
+25 +25 +25 +25

10 55 45 25 25 (Ans)

Query (i, x) →
Add x to all the numbers from i to (N-1).

A = [0 0 0 0 0]
0 1 2 3 4

+3 +3 +3 +3
+2 +2 +2 +2 +2
+1

2 5 5 5 6 (Ans)

(N-1)

I/P → 1 4 3
0 4 2
4 4 1

BF Idea

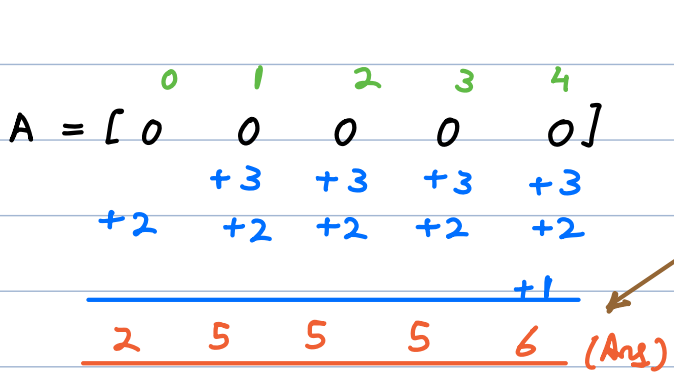
∀ query, travel from l to r & add x.

length of array → A

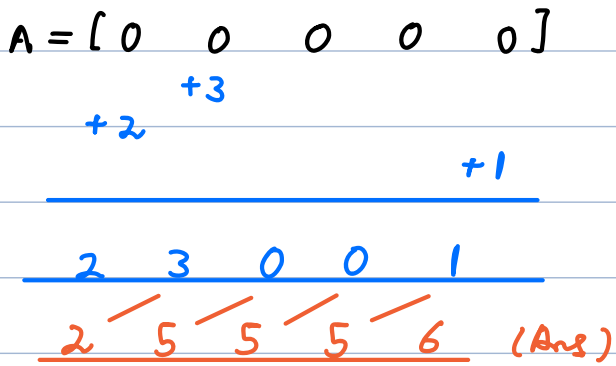
queries → N

```
for i → 0 to (N-1) {   // query    // For 1-based do -1
    l = B[i][0]    r = B[i][1]    x = B[i][2]
    for j → l to r {
        ans[j] += x
    }
}
return ans
```

TC = O(N * A) SC = O(1)



Add x from i ⟶ (N-1)



Prefix Sum

P[i] = A[i] + P[i-1]

```
for i → 0 to (N-1) {  // query
    l = B[i][0]      x = B[i][1]
    ans[l] += x
}

for i → 1 to (A-1) {
    ans[i] = ans[i-1] + ans[i]
}      P[i]   P[i-1]  + a[i]

return ans
```

TC = O(N + A) SC = O(1)

A = [0 0 0 0 0 0 0] (indices 0 1 2 3 4 5 6)

+2 at 1, -2 at 4, -5 at 4, +3 at 2, -3 at 6

0 2 3 0 -7 0 -3

0 2 5 5 -2 -2 -5

Queries

l	r	x
1	3	2
4	6	-5
2	5	3

```
for i → 0 to (N-1) {  // query
    l = B[i][0]      r = B[i][1]      x = B[i][2]
    ans[l] += x
    if (r != (N-1))   ans[r+1] -= x
}

for i → 1 to (A-1) {
    ans[i] = ans[i-1] + ans[i]
}      P[i]   P[i-1]  + a[i]

return ans
```

TC = O(N+A) SC = O(1)

< Question > : Given arr[N], where arr[i] → height of building.

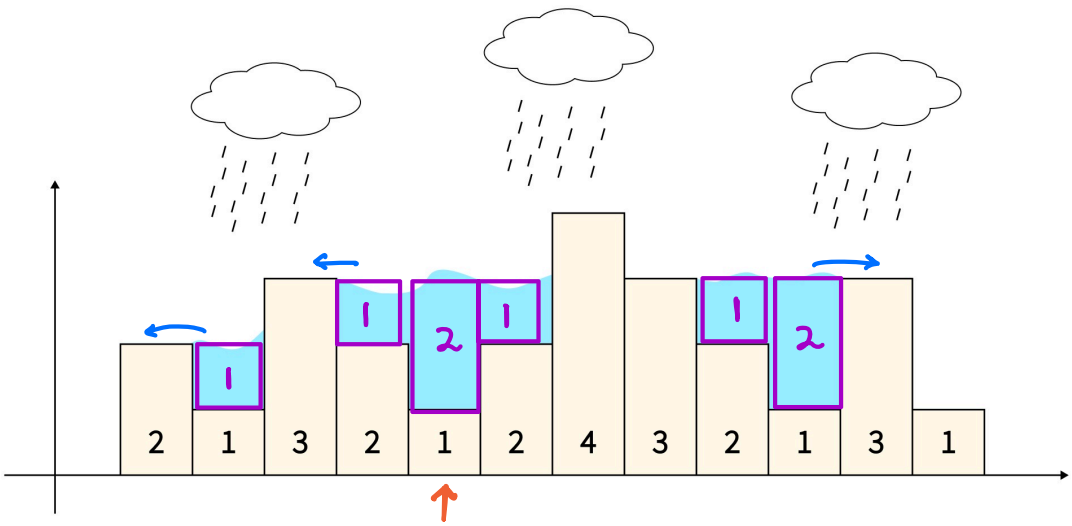
Base = 1 ∀ buildings

Return amount of water trapped on all the buildings.

Area = B * H

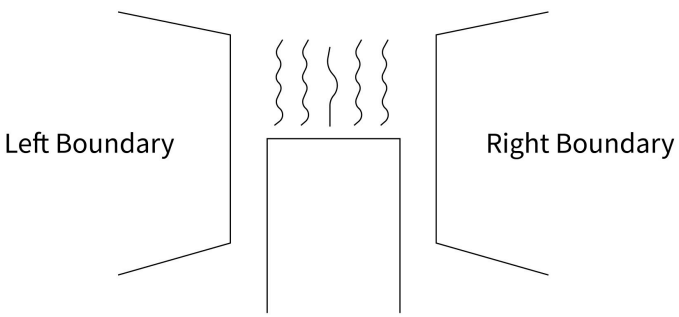
arr [2 1 3 2 1 2 4 3 2 1 3 1]

$1 \le N \le 10^5$

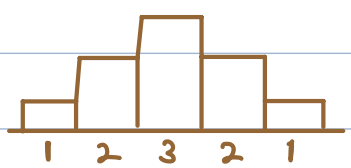


Ans = 8

Idea -1 Find the amount of water trapped on every building.



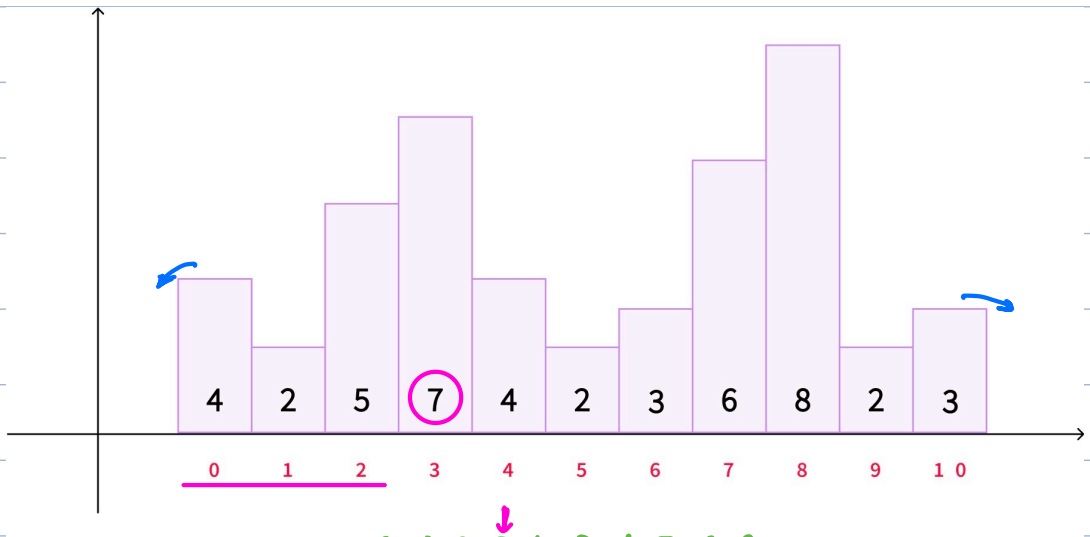
min (max A[i] on left , max A[i] on right) - A[i]



Ans = 0

Bruteforce → TC = O(N²)

SC = O(1)



0 1 2 3 4 5 6 7 8 9 10

→ lmax[] → [4 4 5 7 7 7 7 7 8 8 8]

→ rmax[] → [8 8 8 8 8 8 8 8 3 3]

```
lmax[0] = A[0]
for i → 1 to (N-1) {        // →
    lmax[i] = max(lmax[i-1], A[i])
}

rmax[N-1] = A[N-1]
for i → (N-2) to 0 {        // ←
    rmax[i] = max(rmax[i+1], A[i])
}
```

Prefix / Suffix → sum, max, min etc

```
ans = 0
for i → 0 to (N-1) {
    ans += min(lmax[i], rmax[i]) - A[i]
}
return ans
```

TC = O(N) SC = O(N)

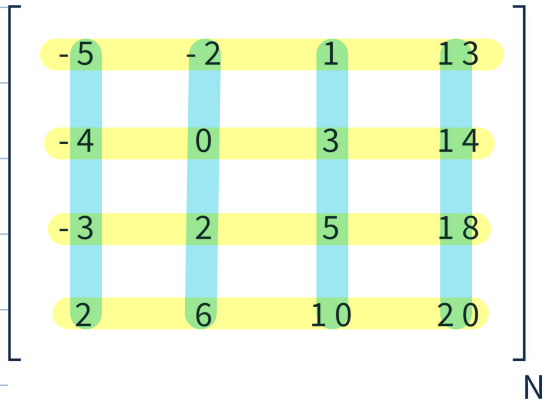
SCALER

2D Matrices

TABLE OF CONTENTS

1. Search in row-wise & column-wise sorted 2D array
2. Row with maximum number of 1's
3. Spiral Matrix
4. Sum of all sub-matrices sum

Search in row-wise & column-wise sorted 2D array

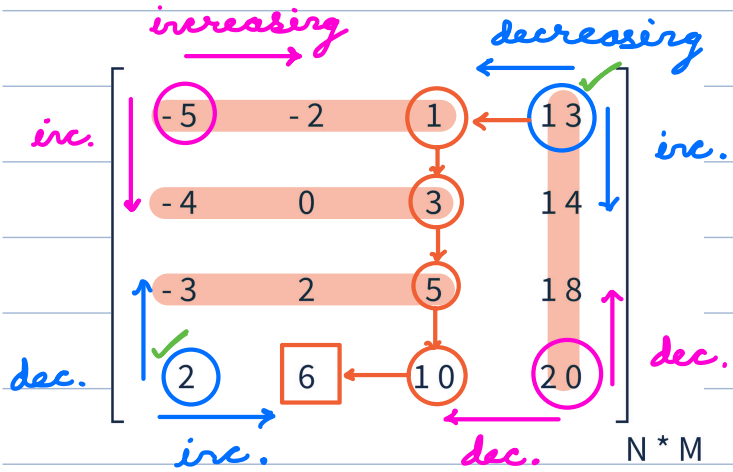


Search (6) → ✓ true

Search (15) → ✗ false

Bruteforce → Travel all cells.

TC = O(N * M) SC = O(1)



travel from corners.

Search (6) → true

Start → Top Right or Bottom Left

```
r = 0    c = M-1
while ( r < N && c >= 0 ) {
    if ( A[r][c] == K )   return true
    else if ( A[r][c] < K )   r++
    else      c--
}
return false
```

TC = O(N+M) SC = O(1)

Row with maximum number of 1's

Given a binary sorted matrix A of size N*N. Find the row with the maximum number of 1's [Only rows are sorted]

0 0 . . . 0 1 1 . . . 1

```
        0   1   2
A = [ 0 [ 0 , 1 , 1 ]
    1 [ 0 , 0 , 1 ]
    2 [ 0 , 1 , 1 ]  ]
```

Ans = 0

If multiple rows are present, return the first row.

```
        0   1   2   3
A = [ 0 [ 0 , 0 , 0 , 0 ]
    1 [ 0 , 0 , 0 , 1 ]
    2 [ 0 , 0 , 1 , 1 ]
    3 [ 0 , 1 , 1 , 1 ]  ]
```

Ans = 3

	0	1	2	3
0	0	1	1	1
1	0	0	0	1
2	1	1	1	1
3	1	1	1	1

Ans = 2

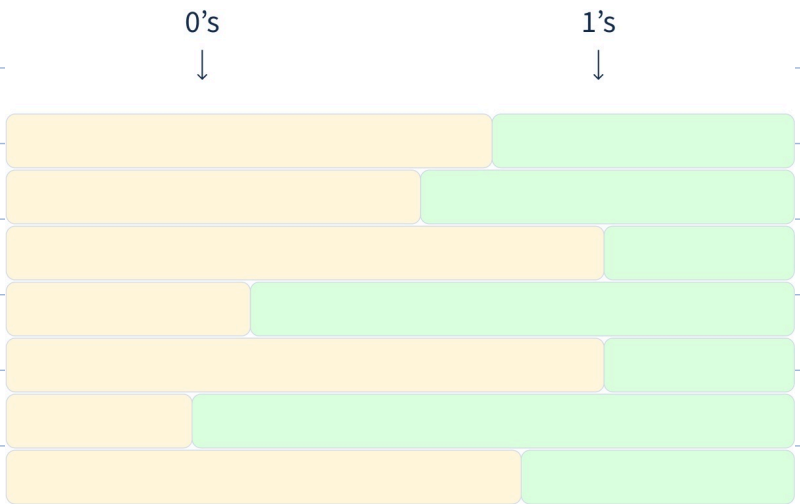
Bruteforce → ∀ rows, count #1's.

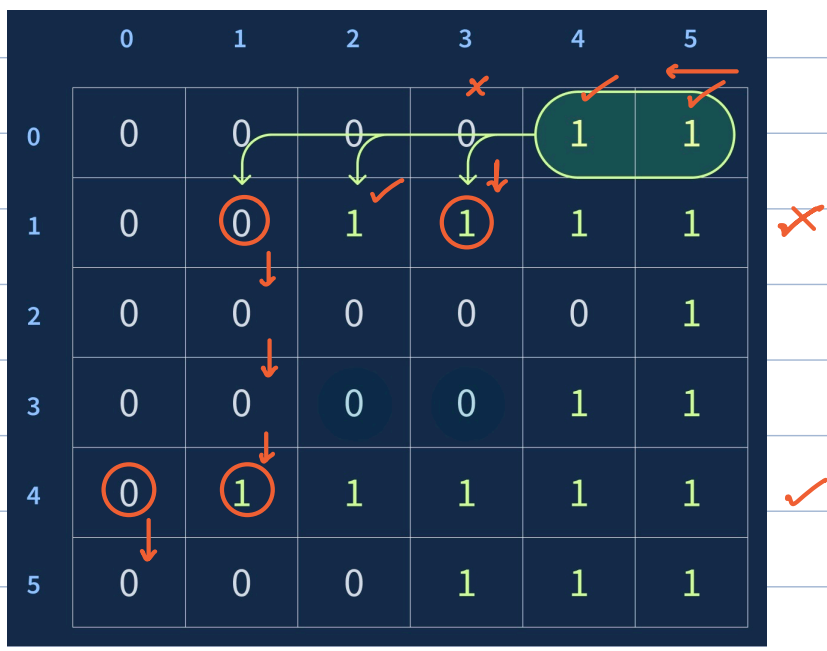
TC = $O(N^2)$ SC = $O(1)$

Observation

1) Row with min 0's ✓

2) First 1 can give #1's ✓





Ans = 4

```
ans = 0
r = 0      c = M-1
while (r < N && c >= 0) {
    while (c >= 0 && A[r][c] == 1) {
        c--
        ans = r
    }
    r++
}
return ans
```

	0	1	2	3
0	0	1	1	1
1	0	0	0	1
2	1	1	1	1
3	1	1	1	1

ans = ~~0~~ 2

TC = O(N + N) = O(N) SC = O(1)

Print Boundary Elements

mat[N][N] ⟶ Square Matrix

mat[5][5]

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

o/p → [1 , 2 , 3 , 4 , 5 , 10 , 15 , 20 , 25 , 24 , 23 , 22 , 21 , 16 , 11 , 6]

A = $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$

o/p → 1 2 3 6 9 8 7 4

mat[N][N]

mat[5][5]

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

```
r = 0    c = 0
while ( c < (N-1)) {                    →
    print ( A[r][c])
    c++
}
while ( r < (N-1)) {                    ↓
    print ( A[r][c])
    r++
}
while ( c > 0) {                        ←
    print ( A[r][c])
    c--
}
while ( r > 0) {                        ↑
    print ( A[r][c])
    r--
}
```

$TC = O(4 * (N-1)) \rightarrow \underline{O(N)}$

$SC = \underline{O(1)}$

Spiral Matrix

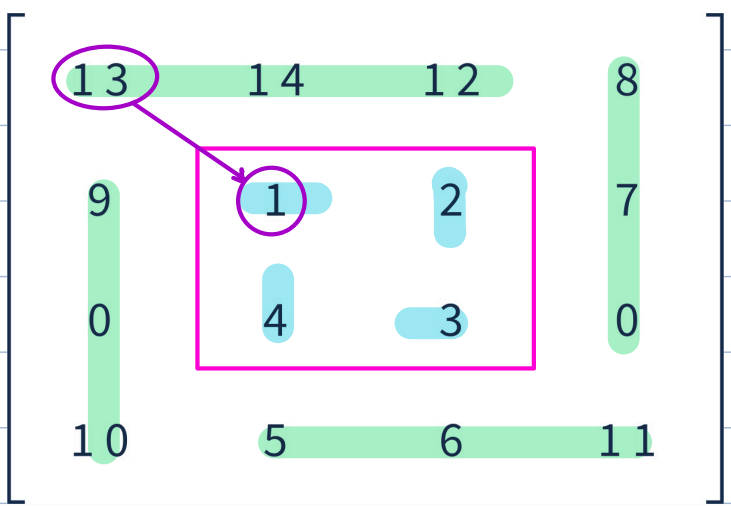
mat[N][N]

	0	1	2	3	4	5
0	1	2	3	4	5	6
1	7	8	9	10	11	12
2	13	14	15	16	17	18
3	19	20	21	22	23	24
4	25	26	27	28	29	30
5	31	32	33	34	35	36

o/p → [1 , 2 , 3 , 4 , 5 , 6 , 12 , 18 , 24 , 30 , 36 , 35 , 34 , 33 , 32 , 31 , 25 , 19 ,

13 , 7 , 8 , 9 , 10 , 11 , 17 , 23 , 29 , 28 , 27 , 26 , 20 , 14 , 15 , 16 , 22 , 21]

Quiz :



o/p → 13 14 12 8 7 0

11 6 5 10 0 9

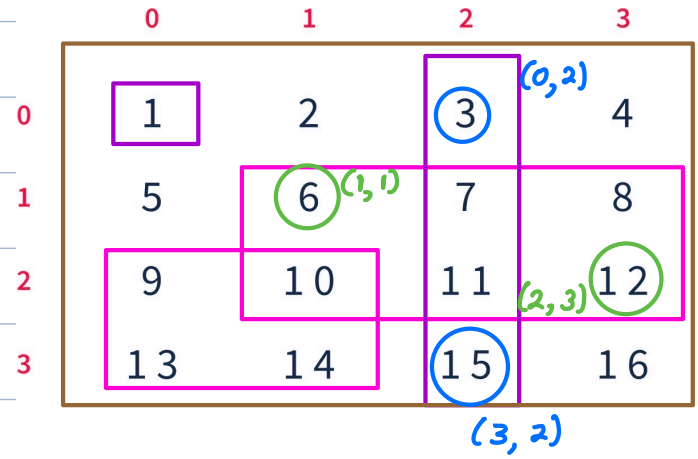
1 2 3 4

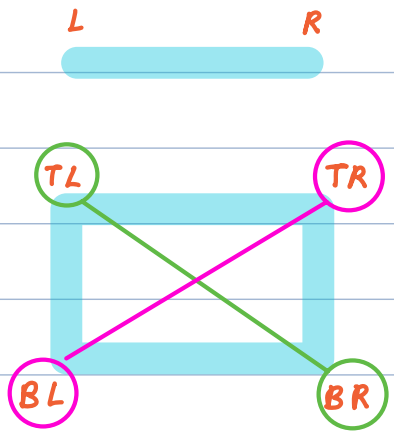
```
r = 0    c = 0
while ( N > 1 ) {
    for (i → 1 to (N-1)) {          →
        print ( A[r][c] )
        c++
    }
    for (i → 1 to (N-1)) {          ↓
        print ( A[r][c] )
        r++
    }
    for (i → 1 to (N-1)) {          ←
        print ( A[r][c] )
        c--
    }
    for (i → 1 to (N-1)) {          ↑
        print ( A[r][c] )
        r--
    }
    r++
    c++
    N -= 2
}
if (N == 1)  print ( A[r][c] )
```

TC = $O(N^2)$ SC = $O(1)$

Sub - Matrices

- Contiguous part of a matrix





< *Question* > : Given mat[N] [M]. Find sum of all sub-matrix sums.

	0	1	2
0	4	9	6
1	5	-1	2

2 * 3

[4] 4 [4 9] 13 [4 / 5] 9

[5] 5 [9 6] 15

[9] 9 [5 -1] 4 [9 / -1] 8

[-1] -1 [-1 2] 1

[6] 6 [4 9 6] 19 [6 / 2] 8

[2] 2 [5 -1 2] 6

subarray = $\frac{N*(N+1)}{2}$

[4 9 / 5 -1] 17 [9 6 / -1 2] 16 [4 9 6 / 5 -1 2] 25

submatrix = $\frac{N*(N+1)}{2} * \frac{M*(M+1)}{2}$

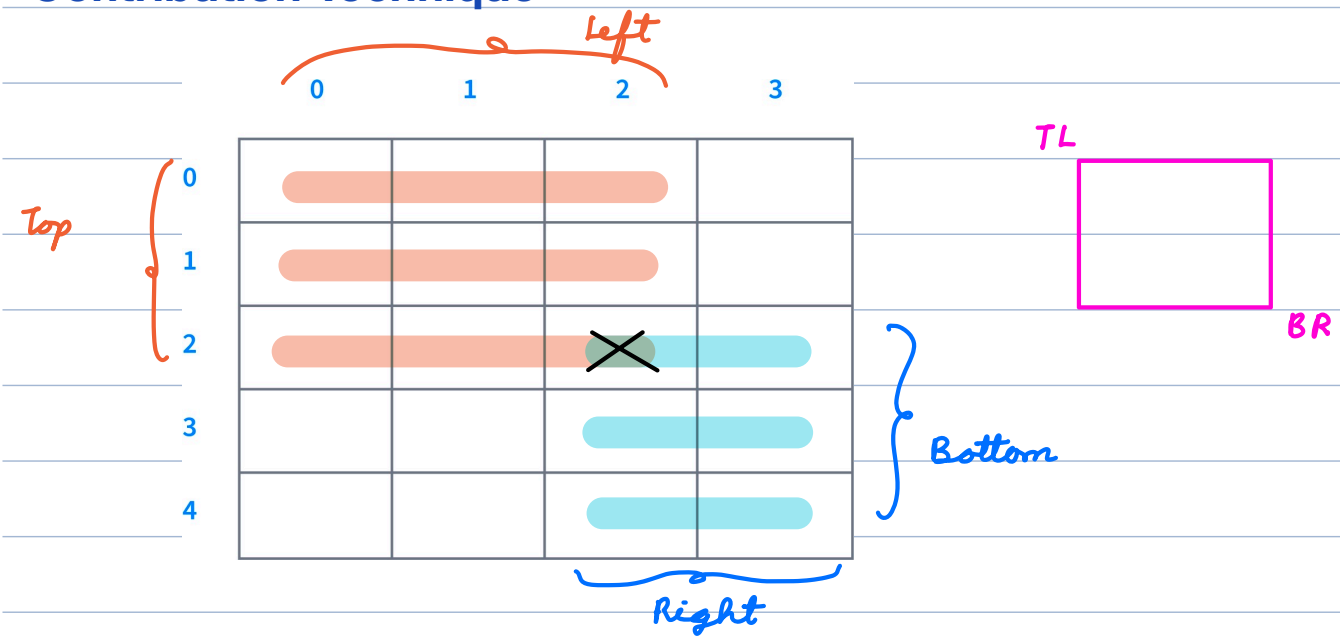
Ans = 166

2 * 3 ⟶ 3 * 6 = 18

Contribution Technique → Ans = $\sum_{i,j}$ contribution of A[i][j]

A[i][j] * (# submatrix A[i][j] is part of)

Contribution Technique



- In how many sub - matrices (2 , 2) will be present?

9 * 6 = 54

i , j

Top → [0 i] → (i + 1)

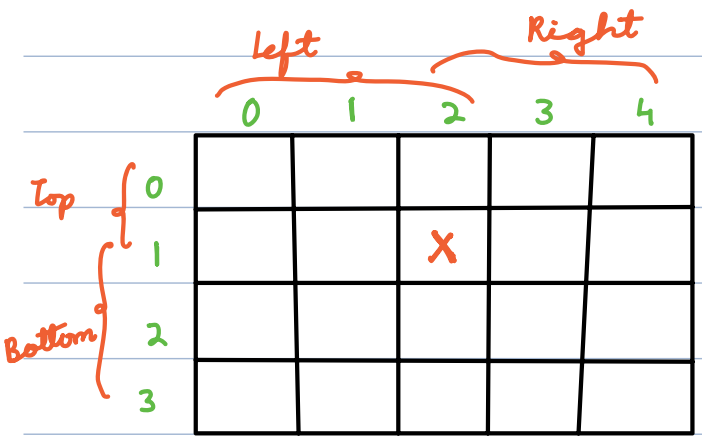
Left → [0 j] → (j + 1)

Bottom → [i N-1] → (N - i)

Right → [j M-1] → (M - j)

```
ans = 0
for i → 0 to (N-1) {
    for j → 0 to (M-1) {
        ans += A[i][j] * (i+1) * (j+1) * (N-i) * (M-j)
    }
}
return ans
```

TC = O(N * M) SC = O(1)



Top → 2

Left → 3

Bottom → 3

Right → 3

2 * 3 * 3 * 3

= 6 * 9

= 54

Today's Agenda

- merge Intervals → (2)
- find first +ve Integer
  - missing

Intro

Microsoft → 260

Pepcoding → Cofounder

Scaler → 2 years.

↳ Certification.

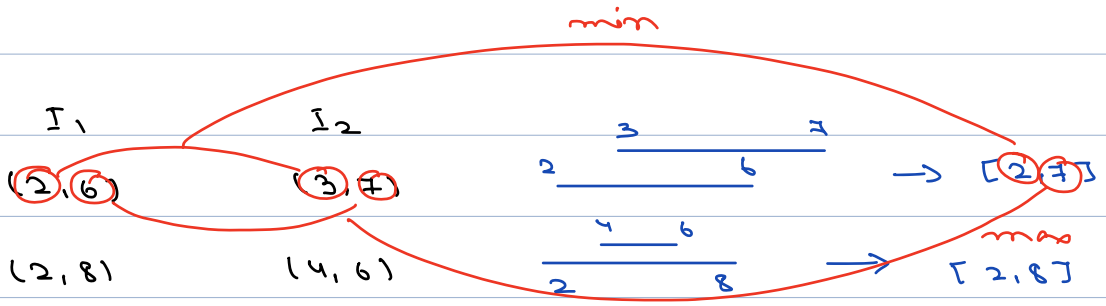
LLD, Database, HLD

Project

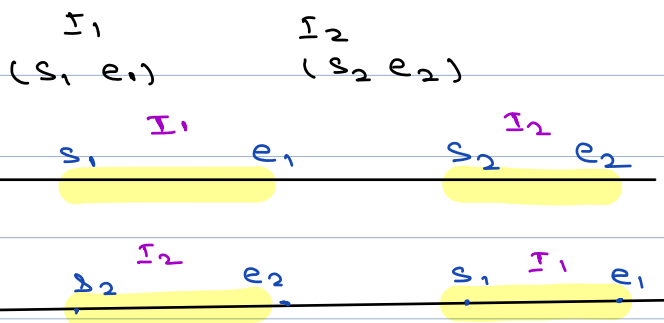
Merge Interval

Interval [s e] → [2, 5]

[1 5]



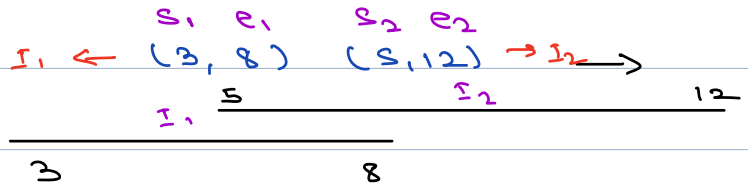
I_1	I_2		
(2, 6)	(3, 7)	2 6 / 3 7	→ [2, 7]
(2, 8)	(4, 6)	2 8 / 4 6	→ [2, 8]
(3, 7)	(4, 10)	3 7 / 4 10	→ [3, 10]
(3, 6)	(6, 10)	3 6 / 6 10	→ [3, 10]
(2 5)	(8, 10)	I_1 2 5 / I_2 8 10	→ no overlap
(5 8)	(1 3)	I_2 1 3 / I_1 5 8	→ no overlap



if ($s_2 > e_1$ || $s_1 > e_2$) → no overlap

if overlap → merged interval

[min(s_1, s_2) , max(e_1, e_2)]

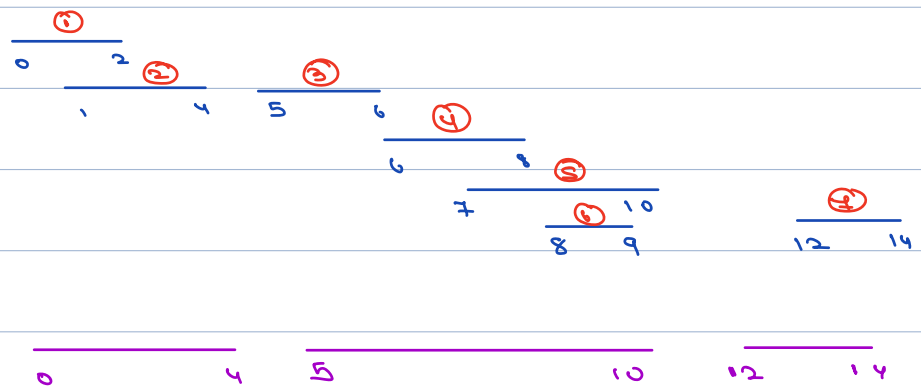


Ques

You are given a collection of intervals A in a 2-D array format, where each interval is represented by a pair of integers `[start, end]`. The intervals are sorted based on their start values.

Your task is to merge all overlapping intervals and return the resulting set of non-overlapping intervals.

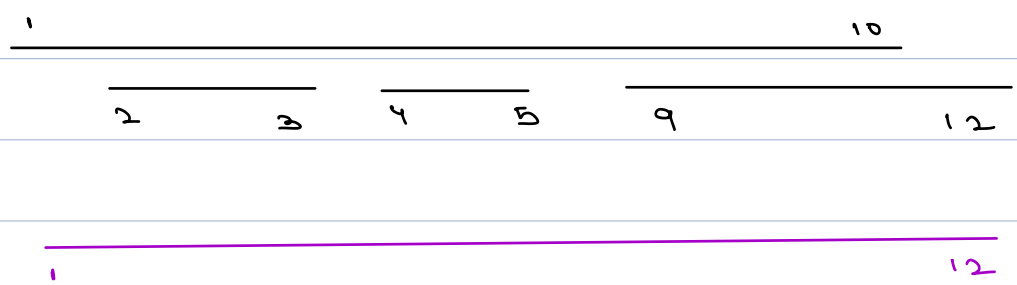
Intervals[] = {(0,2), (1,4), (5,6), (6,8), (7,10), (8,9), (12,14)}



Ans = {(0,4), (5,10), (12,14)}

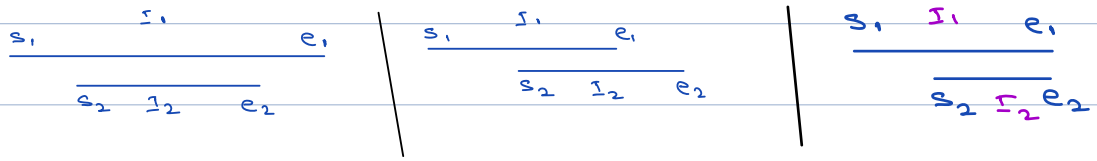
Quiz

(1,10), (2,3), (4,5), (9,12)

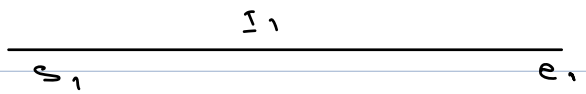


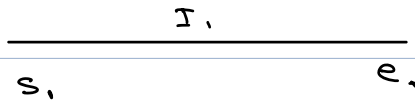
I_1 I_2

(s_1, e_1) (s_2, e_2)



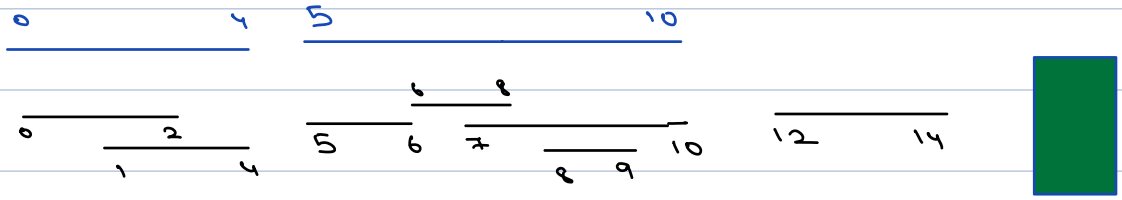
overlap $s_2 <= e_1$





Overlap → $s_2 <= e_1$

arr[0][0]
arr[0][1]



Intervals[] = {(0,2), (1,4), (5,6), (6,8), (7,10), (8,9), (12,14)}

Curr	most Available	After merged	final ans
(0-2)	(1-4)	(0-4)	
(0-4)	(5-6)	—	(0-4)
(5-6)	(6-8)	(5-8)	(0-4)
(5-8)	(7-10)	(5-10)	(0-4)
(5-10)	(8,9)	(5-10)	(0-4)
(5-10)	(12-14)		[(0-4), (5-10)]
(12-14)			[(0-4), (5-10), (12-14)]

int[] arr → int.

Interval[] arr → interval

arr[0].s , arr[0].e

Interval [] arr; // Given.

List<Interval> ans;

curr_start = arr[0].s, curr_end = arr[0].e

for (i=1; i < n; i++) {

if (arr[i].s <= curr_end) { // overlap

// merge

curr_end = max(curr_end, arr[i].e);

}

else {

print (curr_start, curr_end); // store it to ans;

curr_start = arr[i].s;

curr_end = arr[i].e;

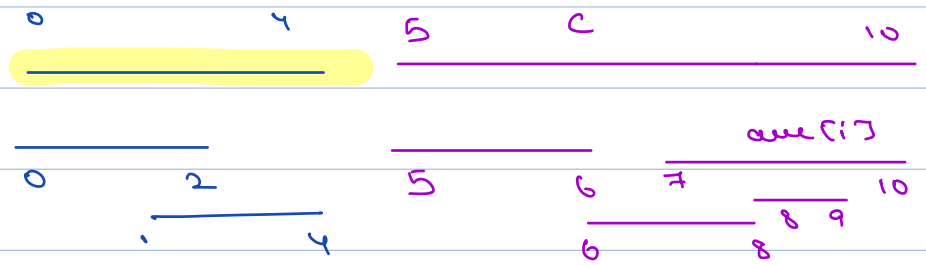
}

}

print (curr_start, curr_end);

T.C -> O(n)

S.C -> O(1)



i = 2

Intervals[] = {(0,2), (1,4), (5,6), (6,8), (7,10), (8,9)};

curr	next Available	After merged	final ans
(0-2)	(1,4)	(0-4)	
(0-4)	(5,6)		(0-4)
(5,6)	(6,8)	(5,8)	
(5,8)	(7,10)	(5,10)	
(5,10)	(8,9)	(5,10)	

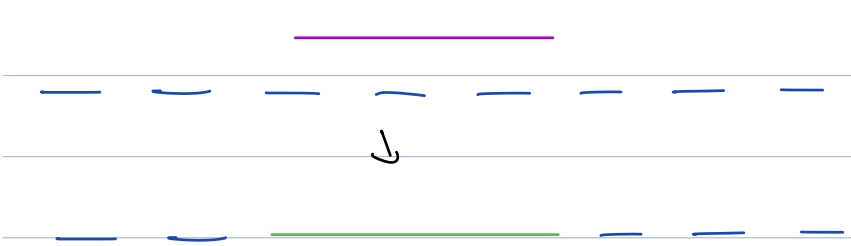
Ques

You have a set of non-overlapping intervals. You are given a new interval [start, end], insert this new interval into the set of intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

N = 9

(1,3)
(4,7)
(10,14)
(16,19)
(21,24)
(27,30)
(32,35)
(38,41)
(43,50)



new interval = [12, 22]

N = 9

(1,3) (12, 22) → (1, 3)

(4,7) (12, 22) → (4, 7)

(10,14) (12, 22) : (10, 22)

(16,19) (10, 22) : (10, 22)

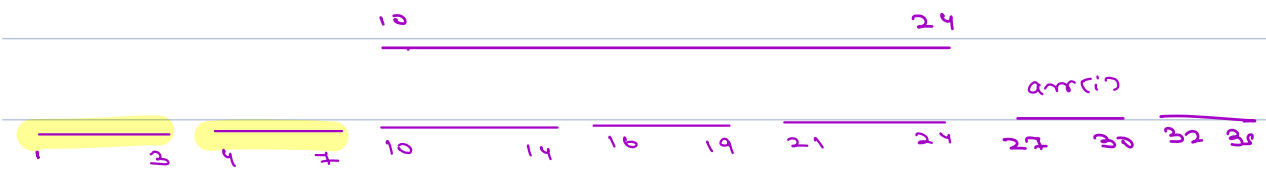
(21,24) (10, 22) : (10, 24)

(27,30) (10, 24) → (10, 24)

(32,35)

(38,41)

(43,50)



Ex 2

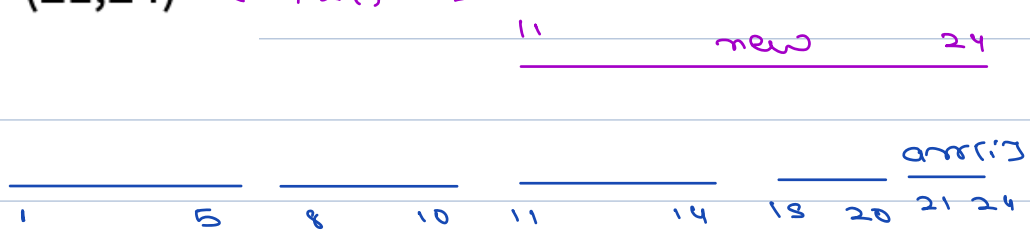
(1,5) (12, 24) → (1, 5)

(8,10) (12, 24) → (8, 10)

(11,14) (12, 24) : (11, 24)

(15,20) (11, 24) : (11, 24)

(21,24) (11, 24) → (11, 24)



ns new ne

arr[i]

cint

s e

T.C → O(n)

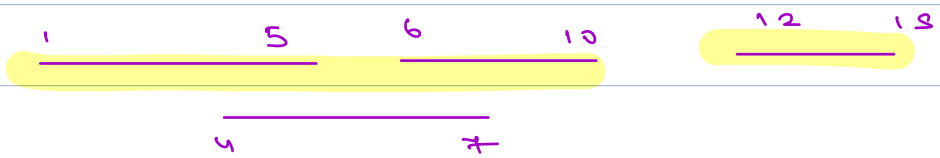
S.C → O(1).

(ns, ne) → new interval.

```
for (i=0; i<n; i++) {
    cInterval = arr[i];
    if (ns > cInterval.e) {
        print (cInterval);
    }
    else if (cInterval.s > ne) {
        print (ns, ne);
        for (j=i; j<n; j++) {
            print (arr[j]);
        }
        return;
    }
    else {
        ns = Min (cInterval.s, ns);
        ne = Max (cInterval.e, ne);
    }
}
print (ns, ne);
```

ns

cI



8:27 am → 8:35 am

Ques

+ve Integers

Given an unsorted array of integers, Find first missing Natural Number.

arr[5] = {3, -2, 1, 2, 7} → 4

arr[7] = {-9, 2, 6, 4, -8, 1, 3} → 5

{-2, 4, -1, -6, 3, 7, 8, 4, -3} → 1

{1, 0, -5, -6, 4, 2} → 3

{1, 2, 5, 6, 4, 3} → 7

{-4, 8, 3, -1, 0} → 1

-4, -1, 0, 3, 8

{4, 2, 1, 3} → 5

[7, 8, 3, 4, 6] → 1

Observation → n → 1 to n+1

n → 5 → 1 → [1, 2, 3, 4, 5] ↳ 6

→ 6

Search → (1 ... 5) [1, 6, 3, 2, 4] ↳ 5

Soln 1:- check all the numbers from 1 to n+1

```
for (i=1; i<=n; i++) {
    → check i in array
      ↳ i is missing → return n.
}
return n+1
```

T.C → O(n²)
S.C → O(1)

Soln 2:-

Add all elements to the set and check if element(1 to N) is present or not.

T.C → O(n)
S.C → O(n)

Soln 3:- Sort & check

T.C → O(n log n)

Soln 4:- T.C → O(n), S.C → O(1).

N = 5

val	idx
1	0
2	1
3	2
4	3
x	x-1

n → 0 → 1
→ n-1 → n

valid 1 to 8

arr[8] =

0	1	2	3	4	5	6	7
4	2	7	6	9	1	8	3
6		8	4		6	7	8
1		3					

→ 5 is missing

i	arr[i]		
0	4	δ(0,3)	δ(0,5)
1	2	set	
2	7	δ(2,6)	δ(2,7)
3	4		
4	9	ignore	
5	set		
6	set		
7	set		

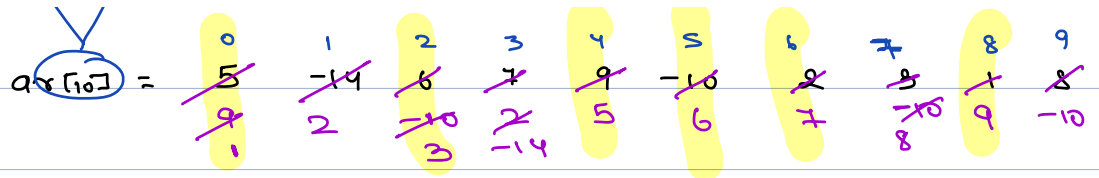
Ans → 5

n → 10 → 0 → 1
→ 9 → 10

n → 0 → 1
→ n-1 → n

10 → 0 → 1
→ 9 → 10

1 to 10



i	arr[i] = val
0	5
1	ignore
2	
3	

Ans → 4

f(0, 4)

f(i, val-1);

```
for (i=0; i<n; i++) {                          → valid
    while (arr[i] > 0 && arr[i] <= n &&
           arr[i] != i+1) {                    → not set
        val = arr[i];
        if (arr[i] == arr[val-1]) { break; }
        f(i, val-1);
    }
}
```

Iterate & find missing no'.

```
for (i=0; i<n; i++) {
    if (arr[i] != i+1)
        return i+1;
}
return n+1;
```

Duplicate issue.

0	1	2	3	4	5
~~4~~, ~~1~~, ~~3~~, ~~2~~, ~~2~~, 3					
~~3~~	~~3~~	3	4	3	
~~3~~	2				
1					

Ans → 2.

T.C → O(n)

S.C → O(1).

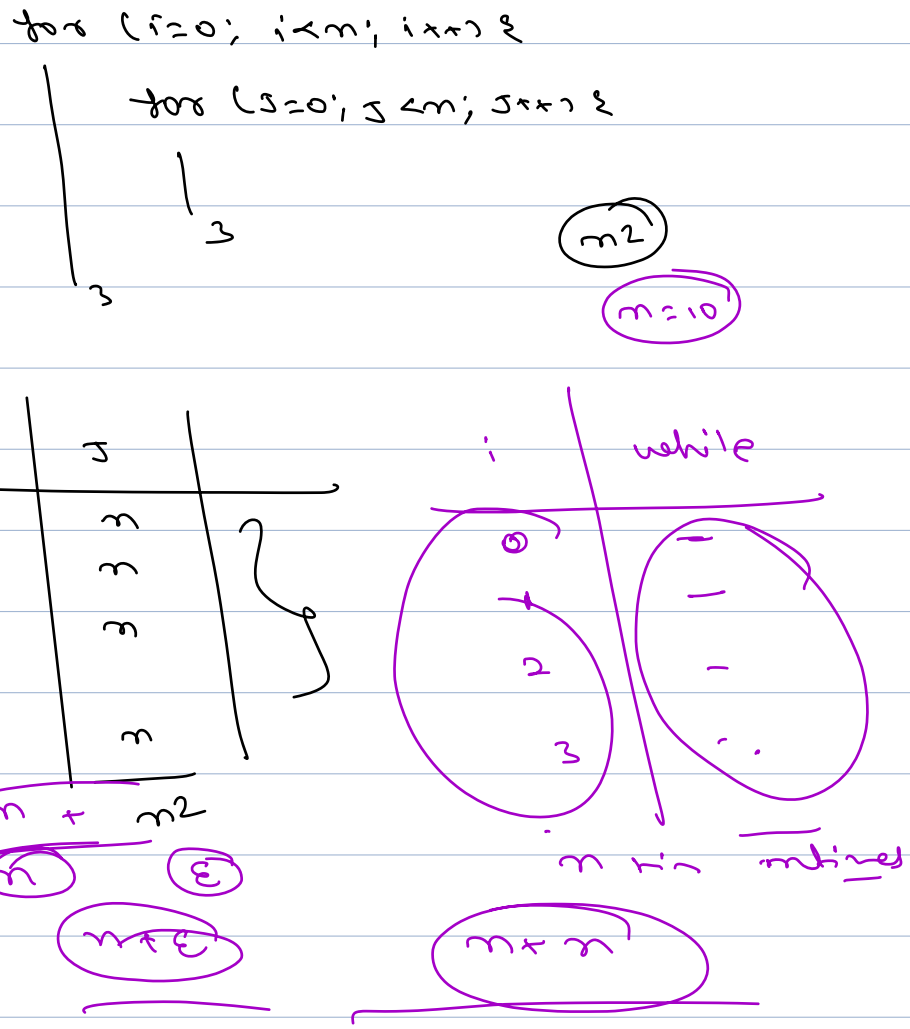
i	arr[i]
0	4

n = 10

0 - 1, 2, 3, ... 9

```
for (i=0; i<n; i++) {
    while (arr[i] > 0 && arr[i] <= n &&
           arr[i] != i+1) {
        val = arr[i];
        if (arr[i] == arr[val-1]) { break }
        swap(i, val-1);
    }
}
```

3 3


for (i=0; i<n; i++) {
for (j=0; j<n; j++) {
n
n
n2
n=10
i
j
0
1
2
3
n
n2
n + n2
while
n + n