Arrays :

	1st	2nd	3rd	4th	5th	6th
Index :	0	1	2	3	4	5
int ar[6] =	-3	2	5	6	7	8

ar[N] → first element : ar[0]

last element : ar[N-1]

[0, N-1] = N Elements

// To access index i^{th} Element : ar[i] ⇒ O(1)

// Print an arr[N]

```
for(i=0; i<N; i++){
    print(ar[i])
}
```

TC : O(N)

Q) Given N array elements, count no. of Elements having atleast 1 Element greater than itself. [No External Libraries]

ar[7] : { -3, -2, 6, 8, 4, 8, 5 }

output : 5

ar[8] = { 2, 3, 10, 7, 3, 10, 2, 10 }

output : 5

Observations

1) For max element There wont be an element greater than That.

2) Calculate No if Max Val = (x)

3) Final : N - x

Steps

1) Find Max ✓

2) Calculate No: of Max = Count

3) Ans: N - Count

// Find max of array

int max = 0 / arr[0] / INT_MIN → if array contains negative.

```
for(i = 0; i < N; i++){
    if(arr[i] > max){
        max = arr[i]
    }
}
```

N

```
int cnt = 0
for(i = 0; i < N; i++){
    if(max == arr[i]){
        cnt++
    }
}
```

N

TC: O(N)

SC: O(1)

```
return N - x
```

TODO HW

Trying it using a single loop.

// 2Q)

Given N array elements, check if there exists a pair (i, j) such that $ar[i] + ar[j] = k$ eq $i \neq j$ } i & j must be

↳ return true/false

↳ Input

[No Extra space / No External Limitation]

	0	1	2	3	4	5	6
ar[] :	3	-2	1	4	3	6	8

k = 10 : i = 3, j = 5 } ar[3] + ar[5] = 10 = ar[5] + ar[3]

k = 8 : No pair

Idea: Generate all pairs sum eq == k

ar[4] : 0 1 2 3

```
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        if(i==j){continue}
        if(ar[i]+ar[j]==k){
            return true
        }
    }
}
return False
```

i==j

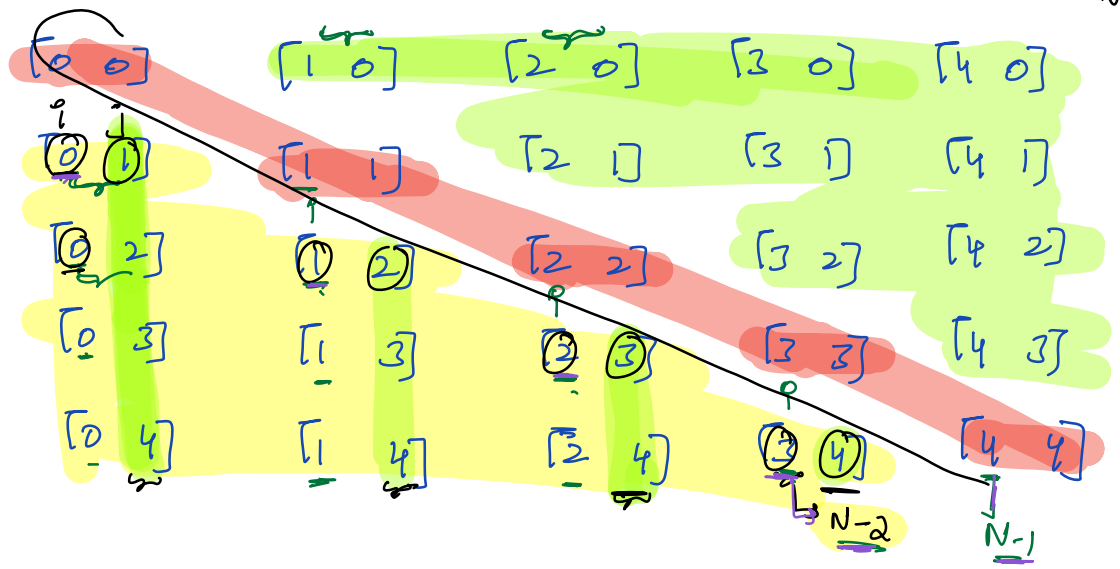
Iterations : N^2

TC : $O(N^2)$

SC : $O(1)$

ar[5] : 0 1 2 3 4] ar[i] + ar[j] == ar[j] + ar[i]

N = 5



// Iteration in lower triangular pairs.

```
for(i = 0; i < N-1; i++){
    for(j = i+1; j < N; j++){
        if(ar[i] + ar[j] == k){
            return True
        }
    }
}
return False
```

// Sum of N Natural

→ $\frac{(N)(N+1)}{2}$

Iterations :

i	j [i+1, N-1]	
0	[1, N-1]	N-1
1	[2, N-1]	N-2
2	[3, N-1]	N-3
⋮	⋮	⋮
N-2	[N-1, N-1]	1

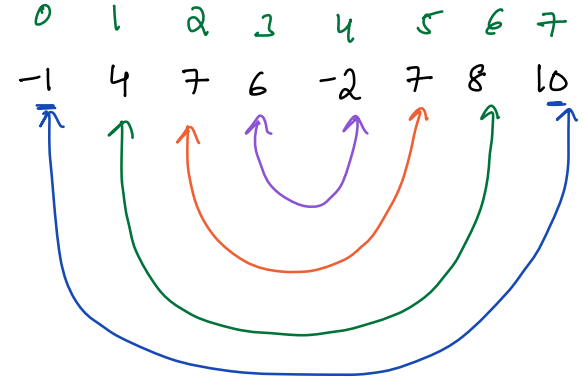
// Sum of N-1 Natural Natural

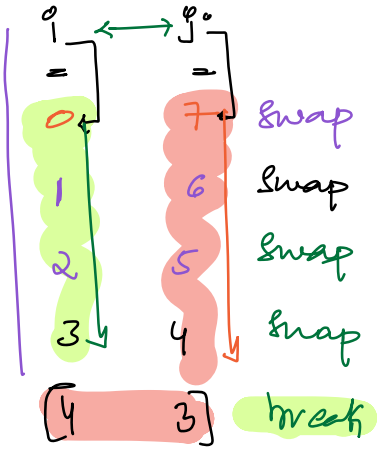
Replace N with N-1

$\frac{(N-1)(N)}{2}$ → $\frac{N^2-N}{2}$: $O(N^2)$

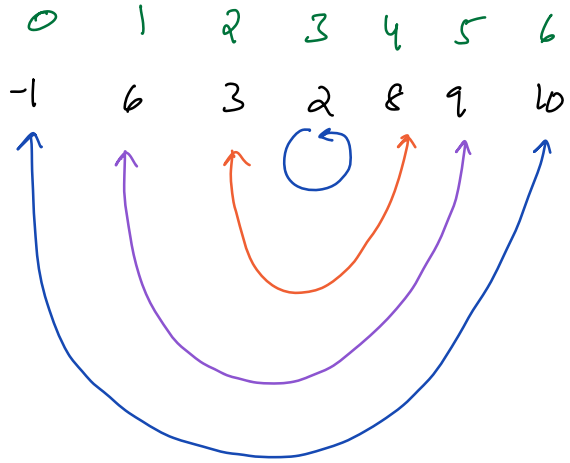
308) Given an array, Reverse entire array[] → No Extra Space

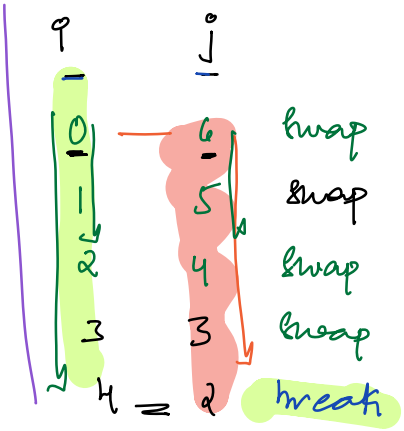
ar[8]:





ar[7]:





```
void reverse(int ar[], int N){
    i = 0, j = N-1        → i < N/2      Iterations : N/2
    while( i <= j ){
        swap ar[i] & ar[j]
        i++, j--;
    }
}
```

TC : O(N)

SC : O(1)

→ on your own un temp variable a Not

Swap a & b

temp = a

a = b

b = temp

a	b	temp
20	10	a

// Subarray : Continous sequence of an array is called Subarray

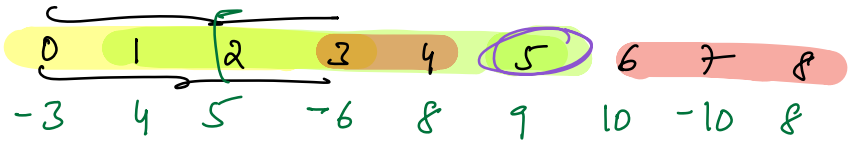
[si ⟷ ei] : All Elements from si to ei is subar

si <= ei

Doubts : 1) Yes, full array is subarray

2) Yes, Single Element is subarray

3) No, Subarray cannot be empty



[1 -5] : Yes

[5 -5] : Is Subarray Yes

[3 4 6 7 8] : NO

5 is missing

[0 -8] : Yes

[2 6] : {5, -6, 8, 9, 10}

si ei

[0 3] : Yes

	0	1	2	3	4	5	6
ar[7]:	3	4	5	6	-2	8	10

// Can subarray start at index = 0

i=0

[0 0]
[0 1]
[0 2]
[0 3]
[0, 6]

// Generalize

[0 0]
[0 1]
[0 2]
:
[0, N-1]

// count = N

Start at index 0

// Can subarrays start at index 1

[1 1]
[1 2]
[1 3]
[1 6]

Generalize

[1 1]
[1 2]
[1 3]
|
[1, N-1]

// start at 1 → N-1

// Generalized start = 2

[2 2]
[2 3]
[2 4]
[2 N-1]

N-2

start = 3

[3 3]
[3 4]
[3 5]
[3 N-1]

N-3

— start = N-1

[N-1, N-1] : 1

Total Subarrays

N + N-1 + N-2 + -- 1

= { N(N+1)/2 }

// Generate all Subarrays

// Generating all subarrays

```
for(i=0; i< N; i++) { // start of subarray
    for(j=i; j< N; j++){ // end of subarray
        // subarray [i...j]
        for(k=i; k<=j; k++) { print(ar[k]) }
        println()
    }
}
```

TC: O(N3)

SC: O(1)

//

	0	1	2	3
ar[4]:	-3	4	5	-6

[0 0] : {-3}

[0 1] : {-3, 4}

[0 2] : {-3, 4, 5}

[0 3] : {-3, 4, 5, -6}

[1 1] : {4}

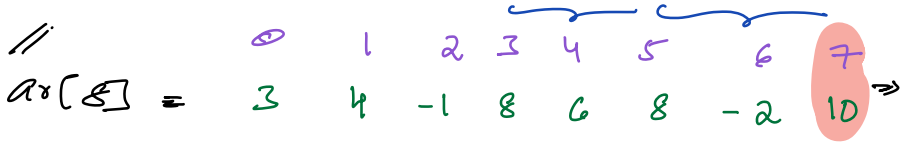
[1 2] : {4, 5}

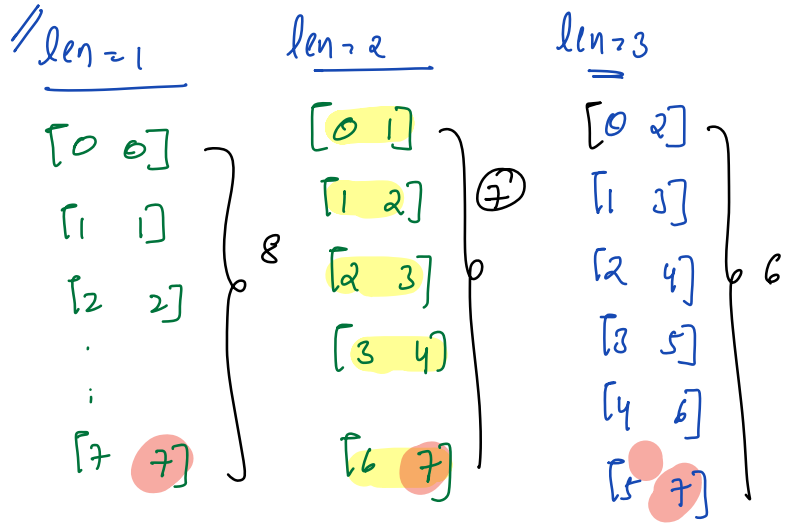
[1 3] : {4, 5, -6}

[2 2] : {5}

[2 3] : {5, -6}

[3 3] : {-6}





// Generalize : N

count len=1 : N

count len=2 : N-1

// Generalize len=3

[0 2]
[1 3]
[2 4]
[3 5]
[N-3 N-1] :

[# count = N-2

// General len=4

[0 3] [K-1]
[1 4]
[2 5]
[N-4 N-1]

count = N-3

Generalize len=k

x [0, k] → k+1 elements

[0, k-1]
[1, k]
[2 k+1]
[N-k, N-1]

[k-1, N-1]
[N-1 -[k-1]]+1
= N-1-k+1+1
= N-k+1

count = [N-k+1]

// Disadvantage of arrays

int ar[10] → 10 Elements

int n;
Read n
int ar[N] → N Elements

→ We cannot increase or decrease size

Dynamic Arrays:
↳ Size not fixed

C++	Java	Python	C#	JavaScript	C
vector	ArrayList	list	ArrayList	array	—

list → Dynamic Array

list<int> a : → a.size() == 0

Dynamic int array

a.insert(20)
a.insert(30)
a.insert(40)
a.insert(50)

0	1	2	3	4
20	30	40	50	-60

→ insert(): This will add an Element at back.
↳ a single insert : O(1)

a.size() == 4

→ size(): → O(1)

a.insert(-60) → delete(): Delete element from back.
↳ O(1)

```
— func (arr[], N) {
    | lprr<int> dlc;
    | for(i=0; i<N; i++){
    |     dlc.insert(arr[i])
    | }
}
```

```
// Given an lprr print
— print(lprr<int> dlc){
    | int N = dlc.size();
    | for(i=0; i<N; i++){
    |     print(dlc[i])
    | }
}
```

// Doubts:

ar[] = { -3, 8, 4, 7, 4, 8, 9, 10 } → N = 1 10^3

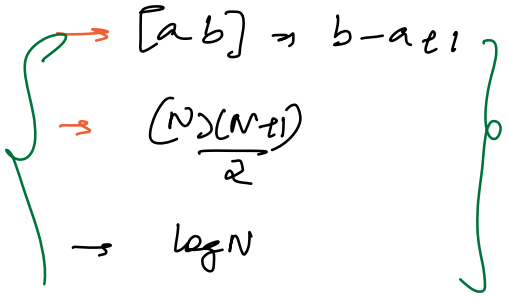
→ TC:

O(1) O(N²) N = 10^3

O(N) O(N)

//

// Doubts :



: Pattern