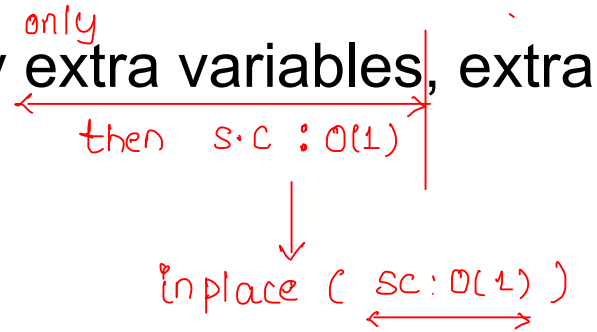
Space Complexity

To find the s.c of any program you need check

-> Apart from the given input, to solve the problem, see what is the extra space that you are using

-> that means, any extra variables, extra array, map, set etc...



1. Constant Space Complexity - $O(1)$: in-place ✓

Example 1: Finding the Maximum Element

python

```python
def find_max(arr):
    max_val = arr[0]  # constant space
    for val in arr[1:]:
        if val > max_val:
            max_val = val
    return max_val
```

i/p

S.C: O(1)

only 2 extra variable.

2. Linear Space Complexity - $O(n)$

Example 2: Copying an Array

```python
def copy_array(arr):
    new_array = arr[:]  # linear space
    return new_array

# Example usage:
arr = [1, 2, 3, 4, 5]
copied_arr = copy_array(arr)
print(copied_arr)  # Output: [1, 2, 3, 4, 5]
```

All array ele's stored in new_array
size: n

n → i/p

Another example:

Q) Find the frequency of each ele in the array of size(n)

Sol) dictionary = { y →

size of array = n.

Worst case, # of entries in dict ?

n entries (if All ele's are diff)

✓Best case -

1 entry (" " same)

∴ SC: O(n)

Quadratic Space Complexity - $O(n^2)$

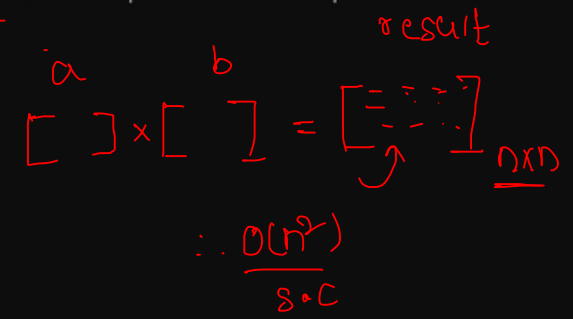
Example 4: 2D Array (Matrix) Multiplication

python

Copy code

```python
def multiply_matrices(a, b):
    n = len(a)
    result = [[0 for _ in range(n)] for _ in range(n)]  # quadratic space
    for i in range(n):
        for j in range(n):
            for k in range(n):
                result[i][j] += a[i][k] * b[k][j]
    return result

# Example usage:
a = [[1, 2], [3, 4]]
b = [[5, 6], [7, 8]]
result = multiply_matrices(a, b)
print(result)  # Output: [[19, 22], [43, 50]]
```



O(logn) and O(2^n) space , you will see in "recursion chapter" ✓

S.C

* → O(1) : Use only extra variables
* → O(n) : New array, dict, new string.
* → O(n²) : New matrix.

2 qs →

S.C created.

↑ new matrix

given i/p ✓

→ ~~O(1)~~

→ O(n²)

~~Logic~~

Why TLE comes?

- **Online Judge Restrictions:** TLE comes because the Online judge has some restriction that it will not allow to process the instruction after a certain Time limit given by Problem setter the problem(1 sec).
- **Server Configuration:** The exact time taken by the code depends on the speed of the server, the architecture of the server, OS, and certainly on the complexity of the algorithm. So different servers like practice, CodeChef, SPOJ, etc., may have different execution speeds. By estimating the maximum value of N (N is the total number of instructions of your whole code), you can roughly estimate the TLE would occur or not in 1 sec.

MAX value of N	Time complexity
10^8	O(N) Border case
10^7	O(N) Might be accepted
10^6	O(N) Perfect
10^5	O(N * logN)
10^4	O(N ^ 2)
10^2	O(N ^ 3)
10^9	O(logN) or Sqrt(N)

- So after analyzing this chart you can roughly estimate your Time complexity and make your code within the upper bound limit.
- **Method of reading input and writing output is too slow:** Sometimes, the methods used by a programmer for input-output may cause TLE.

take

Don't by-heart table.

Reference : ✓

✓MAX value of N (↑ size of i/p.)	Time complexity
10^8	O(N) Border case
10^7	O(N) Might be accepted ↳ some test case may fail.
10^6	O(N) Perfect
10^5	O(N * logN)
10^4	O(N ^ 2)
10^2	O(N ^ 3)
N: 10^9	O(logN) or Sqrt(N)

✓**Constraints**

0 <=**n**<= 1 000 000 000 → $0 \leq n \leq 10^9$ → O(log n) or $O(\sqrt{n})$

↳ 10^9

max
$n = 10^8$ → $O(n)$ may fail for some case.. ✓

then wat might be accepted.

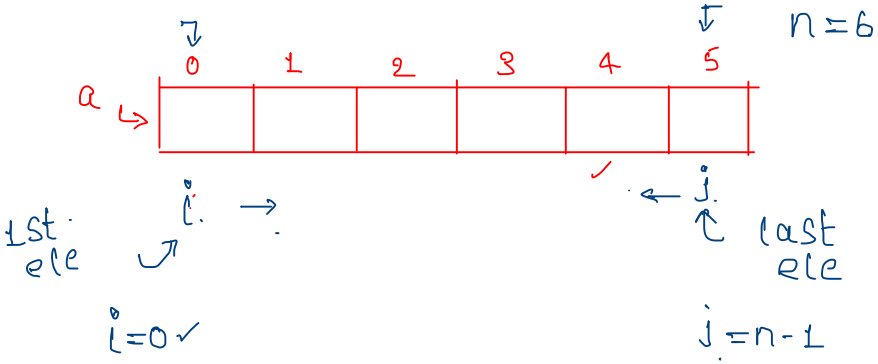
{ $O(\sqrt{n})$ or $O(\log n)$ or $O(1)$ . . . . }

↦ variable

Two - Pointer Technique (one of the optimisation technique).

* Type-1 (2 ptr's, moves in opposite direction)

i → ← j



→ depends on condition.
either i++ (or) j-- happen

→ How long? As long as they are not crossing each other.

i = 1, j = 4 ✓

i = 4, j = 2 ✗

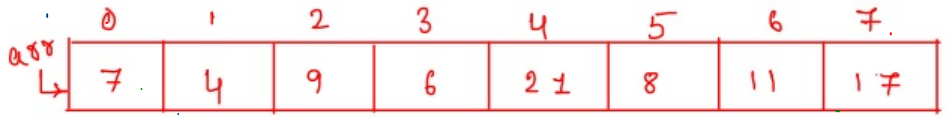
Type-2 (same direction)
i →
j →
later

Remember, in the problem statement they never say apply two pointer of particular type like that.

You need to identify, will I apply two pointer or not for optimization ?

1) Find a pair whose sum is equal to k [a+b=k]

i/p



n = 8 ✓

K = 16 ✓

o/p

T/F

2 diff eles in array

s.t a+b = K

7+9 = 16 ✓

Brute-force soln

```
function findSum(arr[ ], n, k)
{
        for( i=0;i<n;i++)
        {
                for(j=i+1;j<n-1;j++)
                {
                        if(arr[i]+arr[j]==k)
                                return true;
                }
        }
        return false;
}
```

i/p

→ T.C : $O(n^2)$ ✓

S.C : $O(1)$

→ may give TLE.

for $O(n^2)$: TLE,

to remove,

T.C must be less than $O(n^2)$

To remove the TLE, which of the following t.c's are valid? [MSQ]

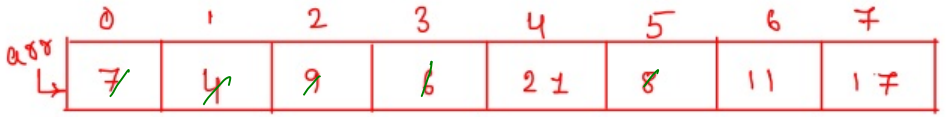
✗ A) O(n^2)

✗ B) O(n^3) → TLE

C) O(n) → ✓

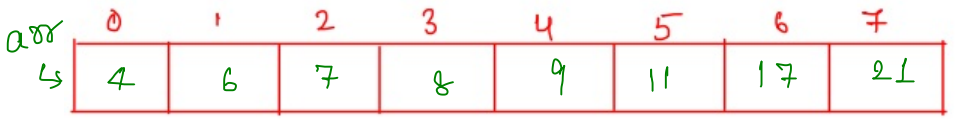
D) O(nlogn) → ✓

} Both are less than $O(n^2)$

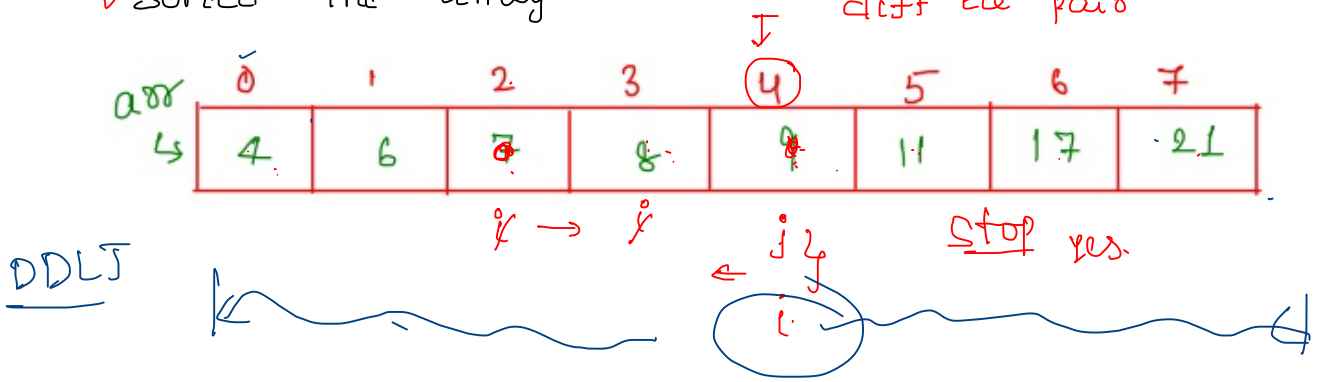


→ Before Applying 2ptr, need to sort the array.

5-10 min



✓ sorted the array.




```
function findSum(arr,n,k)
{
    arr.sort() // python pls check syntax
    → i=0, j=n-1
    while(i<j)
    {
        if(arr[i]+arr[j]==k)
            return true
        if(arr[i]+arr[j]<k) // c2
            i=i+1
        else // c3
            j=j-1
    }
    return false

}
```

a[i]+a[j] v/s k

c1	c2	c3
a[i]+a[j]==k	a[i]+a[j] < k	a[i]+a[j] > k
stop ✓	i++	j--

k

25 16 → c3 → j--

21 > 16 → c3

15 < 16 → c2 → i++ ✓

17 > 16 → c3

15 < 16 → c2

16 == 16 → c1 stop

→ Inplace soln: O(1) SC

Total T.C : nlogn + n = O(nlogn) ✓

↳ nlogn : maximum

```
function findSum(arr,n,k)
{
    1. arr.sort() // python pls check syntax
    + i=0, j=n-1
    ↑ while(i<j)   → O(n)
    ↑ {
    ↑     ✓if(arr[i]+arr[j]==k)
    ↑         return true
    ↑     ✓if(arr[i]+arr[j]<k) // c2
    ↑         i=i+1
    ↑     ✓else // c3
    ↑         j=j-1
    ↓ }
      return false

}
```

Remember for now, O(nlogn)

in Unit-3, you will study why?

S.C : O(1)

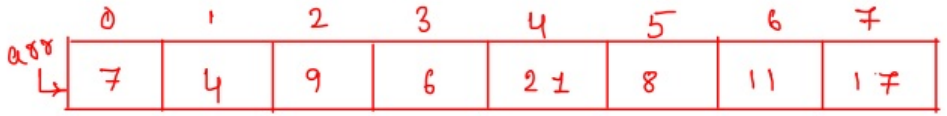
H/W

2) Find a triplet whose sum is equal to k [a+b+c=k]

↓ given i/p

a+b=k ✓

a+b+C=k (triplet)



T.C ✓

AP1: BF → $O(n^3)$

AP2: try using 2ptr ← let me know in slack

3) Seperate 0's and 1's

i/p → one scan

n=13

a

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	1	1	1

o/p

0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	1	1	1	1	1	1	1

freq_count

zero_count = 0 6 (++)

one_count = 0 7 (++)

L_1 → used n times

L_2 → only for zero_count times and set arr values to 0

L_3 → run for one_count times

start: zero_count arr values to 1

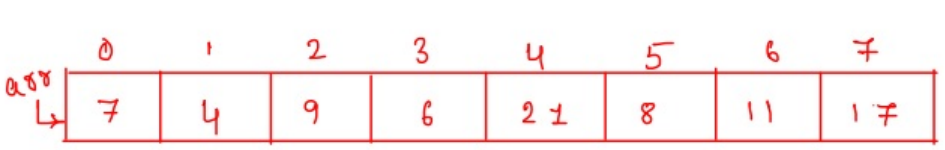
O(n) + ZC + OC

n + n = 2n

O(n)

SC: O(1) → should not use any extra array.

4. Reverse the array [in-place]



2ptr + swaps.