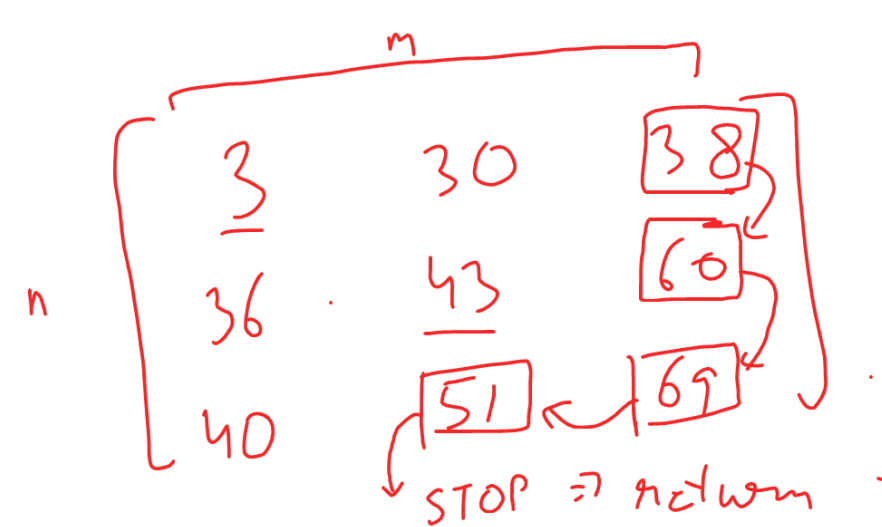
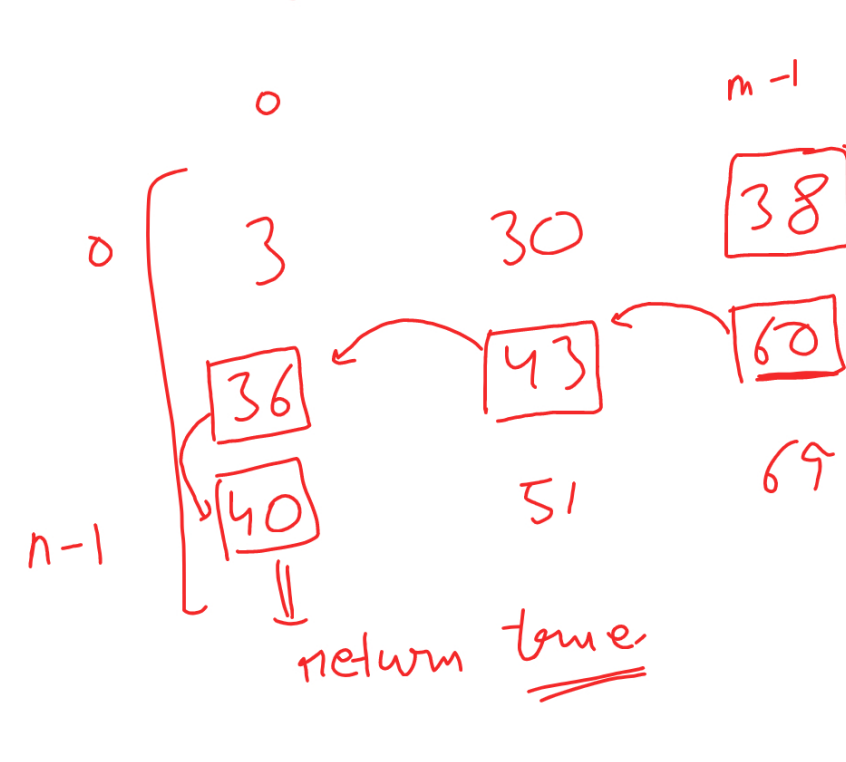


x = 62

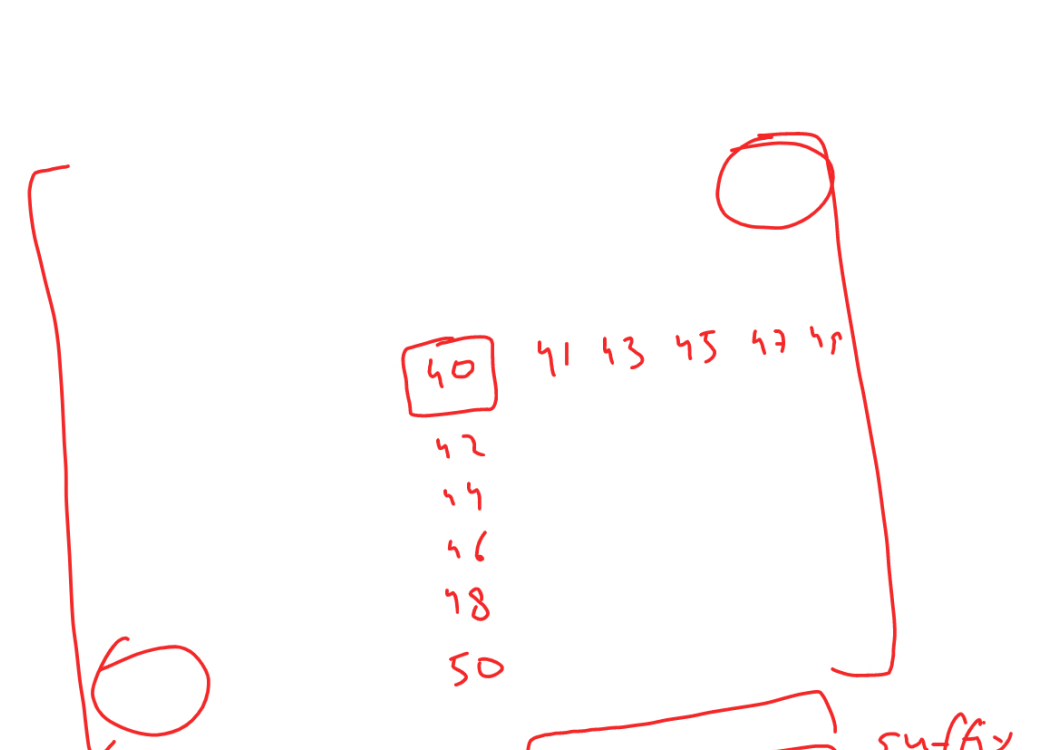
O(n*m)

BS-Based: O(n* log(m))



x = 40

TC: O(n+m)



10×10

48.

(Prefix → Any subarray starting from the beginning.)

[Suffix → Any subarray ending at the end]

$L[\,] = [2 \quad 1 \quad 3]$

$R[\,] = [5 \quad 3 \quad 9]$

1, 9

[2, 5] : [2, 3, 4, 5]

[1, 3] : [1, 2, 3]

[3, 9] : [3, 4, 5, 6, 7, 8, 9]

element	freq
1	1
2	2
⇒ 3	3
4	2
5	~~3~~ 2
6	1
7	1
8	1
9	1

[2, 5] : [2, 3, 4, 5]

[1, 4] : [1, 2, 3, 4]

[3, 9] : [3, 4, 5, 6, 7, 8, 9]

elt	freq
3	3
4	3

[0 , 0 , 0]

[10^5 , 10^5 , 10^5]

Hashmap based approach.

TC: O(n * max_range-length)

wc: n = 10^6 , L[] = [0, 0, 0 . . . . 0]

R[] = [10^6, 10^6, 10^6 . . . . 10^6]

[max no. of computations = $10^6 \times 10^6$ = $\boxed{10^{12}}$]

$\boxed{10^8}$ ⇒ max LIMIT

n = 10^5 $\boxed{O(n^2)}$

[L[] = [1, 2, 3, 100]

R[] = [5, 5, 5, 100]]

[0, 10^6]

$L[\,] = [2 \quad 1 \quad 3]$

$R[\,] = [5 \quad 3 \quad 9]$

	0	1	2	3	4	5	6	7	8	9	10	11
freq:	0	1	1	1	1	-1	0	-1	0	0	0	-1	0
cum. freq:	0	1	2	3	2	2	1	1	1	1	0	0

[for L[i] → increment 1 at index L[i]

R[i] → decrement 1 at index R[i]+1]

L[i] → R[i]

cum-freq [i] = freq [0] + freq [1] - - - - + freq [i]

[cum-freq [i] == frequency of element 'i' in all of the ranges.]

$L[\,] = [3]$

$R[\,] = [7]$

8 ⑦ 8.

size = 9.

	0	1	2	3	4	5	6	7	8
freq:	0	0	0	~~0~~ +1	0	0	0	0	~~0~~ -1
cum-freq:	0	0	0	1	1	1	1	1	0

TC: O(n + max_range)

Aux.space: O(max_range)

Space Comp: O(n + max_range)

[Prefix_sum [i] = prefix_sum [i-1] + current_element]

	0	1	2	3	4	5	6	7	8	9	10	11
freq:	0	1	1	1	1	-1	0	-1	0	0	0	-1	0
cum. freq:	0	1	2	3	2	2	1	1	1	1	0	0

result = ~~2~~ ~~2~~ 3

---

nums: [1, 2, 3, 4] Product of all = 24

ans: $[\frac{24}{1}, \frac{24}{2}, \frac{24}{3}, \frac{24}{4}]$

⇒ [24, 12, 8, 6]

nums: [1, 2, 0, 4] Product of all = ~~0~~ 8

ans: ~~$[\frac{0}{1}, \frac{0}{2}, \frac{0}{0}, \frac{0}{4}]$~~

ans: [0, 0, 8, 0]

[nums: [0, 2, 0, 4]

ans: [0, 0, 0, 0]]

~ 2.147×10^9

nums: [2, 10^9, 2]

ans: [2×10^9, 4, 2×10^9]

[Product of all = 4×10^9]

(64-bit int to store the product)

	0	1	2	3	4
nums:	4	5	1	8	2
left_product:	1	4	20	20	160
right_product:	80	16	16	2	1
answer:	80	64	320	40	160

TC: O(n)

Aux-space: O(n)

left_product [i] = nums [0] * num[1] - - - - * nums [i-1]

right_product [i] = nums [i+1] * nums (i+2) - - - - * nums [n-1]

[

```
[ i: 0 ⟶ n-1
      j: i+1 ⟶ n-1
          if (A[i] + A[j] == target)
              return true; ]
```

$O(n^2)$

```
[ i: 0 ⟶ n-1
      Binary search on arr[i+1 - - - n-1] to
      search for (target - arr[i]) ]
```

TC: O(n * log(n))

[1, 2, 5, 6, 10] target = 8

sum = 11 > 8

1+6 = 7 < 8

2+6 == 8 ⇒ return true.

```
[ i = 0,  j = n-1
    if arr[i] + arr[j] == target:
        return true
    if arr[i] + arr[j] > target:
        j--;
    else:
        i++; ]
```

TC: O(n)

Aux.space: O(1)

[1, 2, 5, 6, 10] target = 9

1+10 = 11 > 9

1+6 = 7 < 9

2+6 = 8 < 9

5+6 = 11 > 9

[1, 4, 6] , target = 8

O/P = false.

[1, 4, 4, 6] target = 8

O/P = True.

TRIPLET SUM

Brute-force: $O(n^3)$

[1, 4, 45, 6, 10, 8] target = 13

↓ sort

fixed

step-1: [1, 4, 6, 8, 10, 45]

check pair with sum = (target - 1)

↳ 2-pointer

fixed

step-2: [1, 4, 6, 8, 10, 45]

check pair with sum = (target - 1)

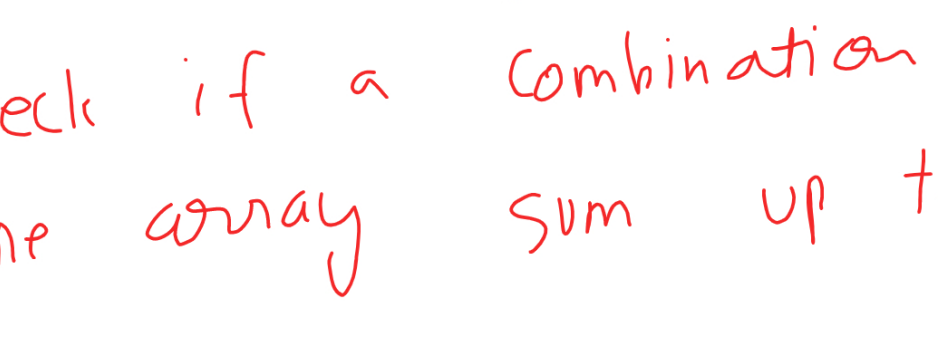
How many runs of 2-pointer? ⇒ n-2 = O(n)

worst-case TC of 2-pointer = O(n)

TC of the algo = O(n log n + n^2)

= O(n^2)

	K	TC (2-ptr)	Brute-force
Pair - Sum:	2	O(n)	O(n^2)
Triplet - sum:	3	O(n^2)	O(n^3)
Quadruple - sum:	4	O(n^3)	O(n^4)
⋮	⋮		⋮
K-sum :	K	O(n^{K-1})	O(n^K)



⇒ Given an array, K, target

check if a combination of any K-elements is the array sum up to target.

Soln: fix (K-2) elements

Search the final 2 elements using 2-ptr approach.

[0, 1, 2 - - - - - - n-1], K, target

[1, 2, 3 - - - - - n-1], K-1, target - arr[0]

K-2.

⋮

K = 2.