=> $O(n \log m)$

x = 62

$$\begin{bmatrix} 3 & 30 & 38 \\ 36 & 43 & 60 \\ 40 & 51 & 69 \end{bmatrix}$$

=> false.

x = 36

$$\begin{bmatrix} 3 & 30 & 38 \\ 36 & 43 & 60 \\ 40 & 51 & 69 \end{bmatrix}$$

38 > 36 => left] j--

30 < 36 => down] i++

43 > 36 => left

36 == 36 -> return true.

i = 0   j = 2

(i < n && j >= 0)

TC: $O(n+m)$

Aux. space: $O(1)$

L[] = [1, 4, 3, 1]

R[] = [15, 8, 5, 4]

(1, 15): [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

(4, 8): [4, 5, 6, 7, 8]

(3, 5): [3, 4, 5]

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

OR = 4.

(1-2 sec.)

($n = 10^6$) $[0, 10^6]$ $10^6 \times 10^6 = 10^{12}$ $\sim 10^8$

10^6

[2, 5] [1, 3] [3, 9]   (9+2)   (R[i]+1) 10^6 index

0	1	2	3	4	5	6	7	8	9	10
0	~~0~~	~~0~~	~~0~~	~~0~~	0	~~0~~	0	0	0	~~0~~
	+1	+1	+1	-1		-1				-1
0	1	2	3	2	2	1	1	1	1	0

result = ~~0~~ ~~1~~ ~~2~~ 3

=> O/P = 3.

n = 2.

$(0, 10^6)$

$(0, 10^6)$

I/P: [1, 2, 3, 4]   [-1, 1, 0, -3, 3]

O/P: [24, 12, 8, 6]   [0, 0, 9, 0, 0]

answer[i] = (Prefix prod of i * suffix prod of i)

	0	1	2	3
	1	2	3	4
Pref:	1	1	2	6
Suff:	24	12	4	1
ans:	24	12	8	6

-> 0

$$Pref[i] = \begin{cases} 1 & \text{for } i = 0 \\ nums[0] \times nums[1] \dots \times nums[i-1] & \forall\ i \geq 1 \end{cases}$$

$$Suff[i] = \begin{cases} 1 & \text{for } i = n-1 \\ nums[i+1] \times nums[i+2] \dots \times nums[n-1] & \forall\ i < n-1 \end{cases}$$

✓ Pref[i] = Pref[i-1] x nums[i-1]   ∀ i > 0

✓ Suff[i] = Suff[i+1] x nums[i+1]   ∀ i < n-1

Pair with a given Sum

#1

```
i: 0 -> n-2
    j: i+1 -> n-1
        if arr[i] + arr[j] == target:
            return true.
```

{[1, 4, 5] target = 8} -> False

[1, 4, 4, 5] target = 8 -> True

TC: $O(n^2)$

Aux. space: $O(1)$

#2 -> Keep storing all elements in a hash-table while iterating through the array & if at any step, (target - arr[i]) is present in the hash-table, return true.

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

hash-table

1 -> True

=> 2 -> True

5 -> True

=> return true

TC: O(n)

Aux. space: O(n)

#3 Two-pointer approach:

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

1+10 = 11 > 8 => j--

1+6 = 7 < 8 => i++

2+6 = 8 == 8 => return true.

TC: O(n)

Aux. space: O(1)

Triplet Sum Problem:

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

=> 1+4+8 = 13 => True.

#1

```
i: 0 -> n-1
    j: i+1 -> n-1
        k: j+1 -> n-1
            if arr[i] + arr[j] + arr[k] == target:
                return true.
```

TC: $O(n^3)$

Aux sp: $O(1)$

#2: Fix two elements & search for the 3rd using a hash-table.

TC: $O(n^2)$

Aux. space: $O(n)$

#3: -> Sort the array

-> Keep fixing one of the elements of the triplet. (Let it be at index i)

-> Perform a 2-pointer based pair sum technique on arr[i+1 ..... n-1] to find (target - arr[i]).

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

[1, 4, 6, 8, 10, 45]   (13-1) = 12

TC: $O(n \log n + n^2) = O(n^2)$

Aux. space: O(1)

NOTE		
Sum of 2	$O(n)$	$O(n^2)$
Sum of 3	$O(n^2)$	$O(n^3)$
Sum of 4	$O(n^3)$	$O(n^4)$
⋮	⋮	⋮
Sum of K	$O(n^{K-1})$	$O(n^K)$
	2-pointer	Brute-force

Follow-up

[3 inputs: arr, target, k]

=> HINT: Recursion

Keep dec k by 1 till k becomes equal to 2

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

[1, 2, 3] => (size = 3)   3

[1, 2, 2, 3, 3]

[1, 2, 3, x, y] (return 3)

#1 Hashset based approach: {1, 2, 3}   O(1)

TC: $O(n * \log n)$

Aux. Space: O(n)

#2 2-Pointer based approach: [1, 1, 2, 3]

0 1 2 3 4

[1, 2, 2, 3, 3]

(ii) Second pointer: Gets me the distinct elements present.

(i) first Pointer: Update arr[i] with the current distinct element (i.e., arr[j])

0 1 2 3 4 5 6

[1, ~~1~~ 2, ~~2~~ 3, ~~3~~ 4, 3, 3, 4]   => return i.

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

TC: O(n)

Aux. space: O(1)

0 1 2 3 4 5 6

[1, 2, ~~2~~ 3, ~~3~~ 4, 3, 3, 4]

i = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 7

new_size = ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

j = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ 7

[1,

[a a a b b c c c d]