Date

5-11-2025

# Basic Array Questions

## Q1 → Find sum of all array elements.

arr = [8, 4, 2, 3, 1]

If you know, how to traverse through an array then ~~t~~this is a piece of cake for you.

How would I know if i need to use for loop or while loop?

For Loop → Where I know how many times it will run as in above case we know it will run till end of array.

While Loop → Loop runs based on some condition and we don't know upfront that how many times it will run.
e.g → while (start <= end) [Will discuss later]

arr = [8, 4, 2, 3, 1]
i ↗

sum ⇒ ~~0~~ ~~8~~ ~~12~~ ~~14~~ ~~17~~ 18

Result = 18

Take one variable & keep on updating the sum in it ~~of~~ while traversing through complete array.

TC → O(N) [To sum every element of an array we need to traverse complete arr, no shortcut]

SC → O(1) [We used only one variable to store the sum which we will eventually return so only 1 variable, size of array does not matter]

Date

## Q2 → Find count of odd numbers in an array.

arr = [4, 5, 8, 9, 6]

Same as last problem, we should know how to traverse through an array and also how can we check if a number is odd or not.

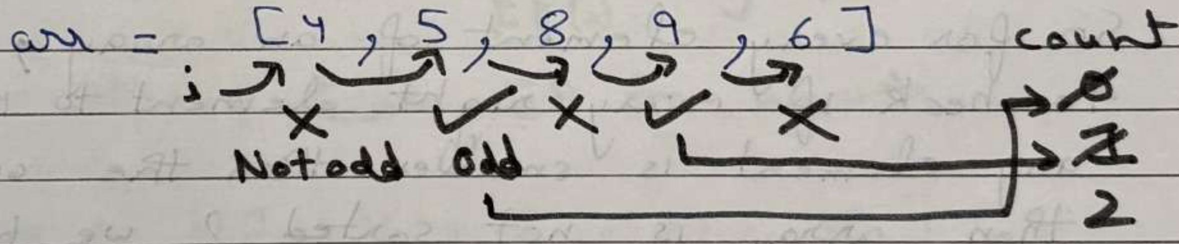
**Simple way to check if num is odd or not →**

- Divide it by 2 and check if remainder is equal to 1 then it is an odd number.

$n \% 2 == 1$

(%) ← This sign is modulo which will return remainder after dividing it with the num.

Note → ( We will check other way to verify odd number in BIT MANIPULATION )



Result = 2

Took one count variable & keep on updating it if the number of array is odd otherwise no need

TC → O(N) [ You should know abhi tak

SC → O(1)

Date

## Q3 → Check if Array is sorted or not (Asc order)

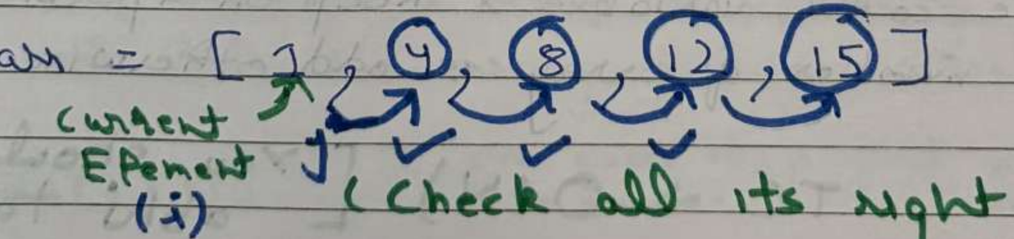
arr = [1, 2, 3, 4, 5]

Here we need to check if array is sorted in ascending order which will mean that if we take any element, all elements to its right will be greater than or equal to it.

arr = [1, 2, 3, 4, 5]
↳ All elements to its right must be >= to it if it is sorted. [For Asc order]

## Brute Force →

So for every element of an array, we need to check if every right element to it & if any element is smaller than the current element then array is not sorted & we break then and there and return false.



A There is no element which is smaller than 1 to its right.

Date

Then we move current element (1) to next position & check for any element which is smaller to its right. If there is none, we move on to next otherwise we break.

$$arr = [1, 4, \boxed{8}, \boxed{12}, \boxed{15}]$$

Curr El (1) → 4; j → 8 ✓ 12 ✓ 15 ✓ checked all its right

Move curr El to next

$$arr = [1, 4, 8, \boxed{12}, \boxed{15}]$$

Curr El (1) → 8; j → 12 ✓ 15 ✓

Move curr El to next

$$arr = [1, 4, 8, 12, \boxed{15}]$$

Curr El (1) → 12; j → 15 ✓

Move curr El to next

$$arr = [1, 4, 8, 12, 15]$$

Curr El (1) → 15

There is no next so we return true

Hence, Array is sorted

TC → $O(N^2)$ SC → $O(1)$

If you see, for size = 5 of array, we made 15 comparison which is not exactly $(N^2)$ but nearly

Date

**Optimal** → Space complexity was good for last one, but we definitely need to improve time complexity.

arr = [1, 4, 8, 12, 15]

So instead of checking every element to its right, we only check next element of current element if it is not smaller to it then we are good.

arr = [1, 4, 8, 12, 15]

**But here is the catch →**

If we start checking from zero index and keep & on checking next element to it then we need to put extra condition of checking if next element exists or not.

**So to avoid this, we start current element from 1. We keep checking previous element if it is greater then it is not sorted.**

$$\begin{tikzcd} \text{arr} = [1, & \boxed{4} \arrow[r, bend right] & 8 \arrow[r, bend right] & 12 \arrow[r, bend right] & 15] \end{tikzcd}$$

curr

1 > 4 ✗
4 > 8 ✗
8 > 12 ✗
12 > 15 ✗

Return True, Array is sorted

Date

arr = [1, 4, 8, 2, 15]

curr

1 > 4 ✗
4 > 8 ✗
8 > 2 ✓ [Return false]
Array Not Sorted

TC → O(N)    SC → O(1)

For size = 5 of an array, we made maximum 5 comparison in worst case where array is sorted.

**Q4 → Reverse the given array**

arr[] = [1, 4, 2, 6, 0]

Result = [0, 6, 2, 4, 1] (Reversed)

**Brute Force →** Create a temp array and place all elements of the given array in the reverse order then replace the given array with elements of temp array.

arr = [1, 4, 2, 6, 0]

Created temp array with same size → temp[]=

Now, I need to fill the temp array in reverse order of elements. There are two ways I can do it →

**1**

$$\begin{array}{rccccc} & 0 & 1 & 2 & 3 & 4 \\ \text{arr} = [ & 1, & 4, & 2, & 6, & 0 ] \\ \text{temp} = [ & 0, & 6, & 2, & 4, & 1 ] \end{array}$$

i

Start loop from back & keep placing element in temp

| | Goes to | |
|---|---|---|
| arr [4] | → | temp[0] |
| arr [3] | → | temp[1] |
| arr [2] | → | temp[2] |
| arr [1] | → | temp[3] |
| arr [0] | → | temp[4] |

Now see relation

n = 5

arr[ i ] → temp[n-i-1]

**2**

$$\begin{array}{rccccc} & 0 & 1 & 2 & 3 & 4 \\ \text{arr} = [ & 1, & 4, & 2, & 6, & 0 ] \\ \text{temp} = [ & 0 & 6 & 2 & 4 & 1 ] \\ & 0 & 1 & 2 & 3 & 4 \end{array}$$

i

Start loop from front & keep placing in temp from last

| | | |
|---|---|---|
| arr [0] | → | temp[4] |
| arr [1] | → | temp[3] |
| arr [2] | → | temp[2] |
| arr [3] | → | temp[1] |
| arr [4] | → | temp[0] |

Now see relation

arr [i] → temp[n-i-1]

Once we have the temp array ready, now we can start loop again and keep replacing elements of temp into given array.

$$\text{arr} = [\overset{0}{\cancel{1}}, \overset{6}{\cancel{4}}, \overset{2}{\cancel{2}}, \overset{4}{\cancel{6}}, \overset{1}{\cancel{0}}]$$

$$\uparrow \quad \uparrow \quad \uparrow \quad \uparrow \quad \uparrow$$

temp = [0, 6, 2, 4, 1]

arr [i] = temp [i]

Date

With this, we have reversed the given array.

$$TC \rightarrow \underbrace{O(N)}_{\text{To fill temp arr}} + \underbrace{O(N)}_{\text{To replace element in the given array}} \Rightarrow O(2N)$$

$SC \rightarrow O(N) \Rightarrow$ (Temp array we created)

Optimal → We should not create extra array, instead reverse in place.

arr = [1, 4, 2, 6, 0]

Result = [0, 6, 2, 4, 1]

We know one thing for sure →

Last element goes to first & first goes to last.

We will take advantage of this, inorder to swap elements I need two pointer left & right.

$$\text{arr} = [\underset{\overset{\uparrow}{\text{Left}}}{1}, 4, 2, 6, \underset{\overset{\uparrow}{\text{Right}}}{0}]$$

Swap → Increase Left → Decrease Right

Here we need to run loop until <u>left < right</u>

Once left & right are equal, we are at middle element which we don't need to swap & our array is sorted.

So we have condition to run loop,

So we will use while Loop.

$$\begin{array}{lccccccc} & & 0 & 1 & 2 & 3 & 4 & \\ arr & = [ & 1, & 4, & 2, & 6, & 0 & ] \\ & & \uparrow & & & & \uparrow & \\ & & \text{Left} & & & & \text{Right} & \end{array}$$

At start
left → 0 , Right → n-1

Swap

$$\begin{array}{lccccccc} & & 0 & 1 & 2 & 3 & 4 & \\ arr & = [ & 0, & 4, & 2, & 6, & 1 & ] \\ & & \uparrow & & & & \uparrow & \\ & & \text{Left} & & & & \text{Right} & \end{array}$$

Increase Left by 1
Decrease Right by 1

$$\begin{array}{lccccccc} & & 0 & 1 & 2 & 3 & 4 & \\ arr & = [ & 0, & 4, & 2, & 6, & 1 & ] \\ & & & \uparrow & & \uparrow & \cancel{\uparrow} & \\ & & & \text{Left} & & \text{Right} & & \end{array}$$

Swap & Move Left Right

Date

$$arr = \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ [0, & 6, & 2, & 4, & 1] \\ & & \uparrow & & \\ & & \text{Left} & & \\ & & \text{Right} & & \end{array}$$

Left Right at same position (middle element) → No swap, Array is reversed.

arr = [0, 6, 2, 4, 1]

TC → O(N)    SC → O(1)
No Extra Array

## Q5) Richest Customer Wealth

LC → 1672

$$\begin{bmatrix} [1, & 2, & 3] \\ [3, & 2, & 1] \end{bmatrix}$$

John wealth → 1+2+3=6
Ram wealth → 3+2+1=6

So maximum wealth is 6, so return 6.

Here you should know how to traverse 2d array, then you are good to go. In each row, there is one person wealth so keep on adding each row & check for max sum of row.

Date

$$arr = \begin{bmatrix} [2, 8, 7], \\ [7, 1, 3], \\ [1, 9, 5] \end{bmatrix}$$

- Take a variable and initialize it with zero which will be our max wealth.
- For each row sum, also we need one variable which will be initialised for each row and we will put each row sum there.

$$\begin{bmatrix} [2, 8, 7] \\ [7, 1, 3] \\ [1, 9, 5] \end{bmatrix}$$

Wealth Max

~~0~~ ~~0~~

~~2~~ ↗ 17

~~10~~

Row 0 → 17 ⇒ 17 > 0 ✓

~~0~~

~~7~~

~~8~~

11

Row 1 → 11 → 11 > 0 ✗

~~0~~

~~1~~

~~10~~

Row 2 → 15 → 15 > 0 ✗

So result is max = 17

$TC \rightarrow O(\overset{Row}{m} \times \overset{Col}{n})$

As you know, matrix traversal takes $O(m \times n)$

$SC \rightarrow O(1)$