First Missing Positive Integer

Ex:-

arr[5] → (3, -2, 1, 2, 7) → ans = 4 ← is missing +ve

arr(7) → (-8, 2, 6, 4, -7, 1, 3) → ans = 5

arr(6) → (2, 1, 6, 4, 3, 5) → ans = 7

arr(5) → (-4, 8, 3, -1, 0) → ans = 1

// Basic - idea

Sort the array & traverse upto last Element

T.C : $O(n \log n) + O(n) \approx O(n \log n)$

Interviewer :- I'm not ok with this Time Complexity, can you optimize it further ?

Me : No, I'm happy with this 😭

Anyway

In Order to think of an Optimal approach, let's look some Observations

① arr[7] :

0	1	2	3	4	5	6
1	2	3	4	5	6	7

Possible min Ans → 1

Possible max ans → 8

So, for array size = 7 → Ans Range (1, 8)

What if it is ∋ n → Ans Range (1, n+1)

Answer range (1, n+1)

{1, n} — we can see them in our array

{n+1} — we can't see this at all,

So, when to return n+1 as answer?

⟹ If each corresponding index has its (+1) value, then we can return (n+1)

* If our given array is sorted, let's see ?

	0	1	2	3	4	5	6	7
arr[8]:	1	2	3	4	5	6	7	8

Each index has its (+1) value at place

*Our main aim is to convert given array into above pattern, from that we can easily get the first missing value by O(n)

	0	1	2	3	4	5	6	7
arr[8]:	1	2	3	9	5	6	7	8

if after all possible arrangement we set all indexes perfectly and still some indexes are not in perfect place it means some positive value is missing.

In above example, at 3rd index → 4 needs to be there

But 4 not present → so first missing element is 4

if before 3rd index, any index faces same issue, then that will be our ans.

as they are acting first missing

So, how to map like →

	0	1	2	3	4	5	6	7
arr[8] :	1	2	3	4	5	6	7	8

By Simple Swapping

	0	1	2	3	4	5	6	7
arr [8] :	4	2	-7	6	9	1	-8	3

arr [0] = 4 , but 4 need to be at index = 3

So, Swap 0th idx & 3rd idx

	0	1	2	3	4	5	6	7
arr [8] :	6	2	-7	4	9	1	-8	3

arr [0] = 6 , but 6 need to be at index = 5

So, Swap 0th idx & 5th idx

	0	1	2	3	4	5	6	7
arr [8] :	1	2	-7	4	9	6	-8	3

Now idx 0 is perfect with its (i+1) value So, increment i

Dry run

arr (0) = 4 ⟶ Actual index for 4 = 3 (Swap (0, 3))

arr (0) = 6 ⟶ Actual index for 6 = 5 (Swap (0, 5))

arr (0) = 1 ⟶ Actual index for 1 = 0 perfect → i++

arr (1) = 2 ⟶ Actual index for 2 = 1 i++

arr (2) = -7 ⟶ Irrelevant ✗ i++

arr (3) = 4 ⟶ Actual index for 4 = 3 i++

arr ( ) = 9 ⟶ Irrelevant i++

arr (7) = 3 ⟶ Actual index for 3 = 2 (Swap (7, 2))

Another key Observation

arr[] : (4, 1, 3, 3, 2) — indices 0 1 2 3 4

arr(0) = 4 → Swap (0, 3)

arr[] : (~~4~~, 1, 3, ~~3~~, 2) — indices 0 1 2 3 4

3 4

arr(0) = 3 → Swap (0, 2)

arr[] : (~~4~~, 1, ~~3~~, ~~3~~, 2) — indices 0 1 2 3 4

~~3~~ 3 4

3

↓

now → Same loop runs again n again

We will go to a infinite loop

So, if you about to Swap → check if Swapping values are Same or not

* if Same → ignore (i++)

* else → do Swap

```java
public int firstMissingPositive(int[] nums) {
    int n = nums.length;

    int i = 0;
    while(i<n){
        if(nums[i]<1 || nums[i]>n || nums[i] == i+1){
            i++;
        }
        else{
            int idx = nums[i]-1;
            if(nums[i] == nums[idx])i++;
            else{
                int temp = nums[i];
                nums[i] = nums[idx];
                nums[idx] = temp;
            }
        }

    }
    for(i = 0;i<n;i++){
        if(nums[i]!=i+1)return i+1;
    }
    return n+1;
}
```

already set (Perfect placed)

Out of range

→ if Swapping index values are Same, don't Swap

} else, Swap

} traverse through the array, & check which index is not perfectly placed

if all are perfect, our answer is (n+1)