LeetCode: 41

✅ Problem Statement (short & crisp)

Mujhe ek **unsorted array** `nums` diya hai. Mujhe **smallest positive integer** (≥ 1) dhoondhna hai jo array me **present nahin** hai — aur ye kaam **O(n)** time me, **O(1)** extra space ke saath karna hai.

🧠 Intuition

- Answer hamesha range **[1 … n+1]** me hota hai (n = array size).
- Ideal placement: value `x` ko index `x-1` par hona chahiye.
- Isliye main **in-place re-arrangement** (cycle-sort style) karta hoon: valid positive aur range ke andar wale elements ko unke **correct index** par swap karta hoon.
- End me linear scan se pehla index `i` jahan `arr[i] != i+1` milta hai, wahi **answer = i+1** hai. Agar sab sahi baith gaya, answer **n+1**.

🚀 Mera Approach (Cycle-Sort style)

- ❌ Negatives/zero/`> n` → ignore (move on).
- 🔁 Agar `arr[i]` ka correct index `arr[i]-1` hai:
  - Agar already sahi jagah par hai **ya** waha duplicate pada hai → move on.
  - Warna **swap** karke `arr[i]` ko uski sahi jagah bhejta hoon.
- ✅ Last me pehla mismatch index de deta hai answer.

Why safe?

- `arr[i] <= 0` hone par main `i++` karta hoon; is branch ke baad hi `(arr[i]-1)` access hota hai, to negative index ka risk nahi.

- `(arr[i]-1) >= n` pehle hi check kar leta hoon; C++ me `||` short-circuit hota hai, to out-of-range index access nahi hota.
- Duplicate check `arr[i] == arr[arr[i]-1]` se infinite swaps avoid.

🧾 Mera Code (as-is, bina badle)

```
class Solution {
public:
    int firstMissingPositive(vector<int>& arr) {
        int i = 0;
        int n = arr.size();
        while(i < n){
            if(arr[i] <= 0) i++;
            else if((arr[i] - 1) >= n || arr[i] == i+1 || arr[i] == arr[arr[i] - 1])
                i++;
            else
                swap(arr[i], arr[arr[i] - 1]);
        }
        for(int i = 0; i < n; i++){
            if(arr[i] != i + 1) return i + 1;
        }
        return n + 1;
    }
};
```

🔍 Dry Run 1 (classic): `nums = [3,4,-1,1]`

Initial:

`i=0` → `arr[0]=3` → correct idx = 2 → 🔄 swap(0,2) → `[-1,4,3,1]`

`i=0` → `arr[0]=-1` → skip → `i=1`

`i=1` → `arr[1]=4` → correct idx = 3 → 🔄 swap(1,3) → `[-1,1,3,4]`

`i=1` → `arr[1]=1` → correct idx = 0 → 🔄 swap(1,0) → `[1,-1,3,4]` `i=1` → `arr[1]=-1` → skip → `i=2`

`i=2` → `arr[2]=3` → already ok → `i=3`

`i=3` → `arr[3]=4` → already ok → `i=4` (done)

Final array: `[1, -1, 3, 4]`

Scan:

- idx0 → 1 ✅
- idx1 → expected 2, found -1 ❌ → **answer = 2** ✅

🔍 Dry Run 2: `nums = [1,2,0]`

Rearrange phase:

- `i=0` → `1` already ok → `i=1`
- `i=1` → `2` already ok → `i=2`
- `i=2` → `0` (≤0) skip → `i=3` (done)

Scan:

- idx0 → 1 ✅
- idx1 → 2 ✅
- idx2 → expected 3, found 0 ❌ → **answer = 3** ✅

🔍 Dry Run 3: `nums = [7,8,9,11,12]`

Sab values `> n` (n=5), to sab skip hoga.

Scan me `idx0` par expected 1 hi missing milta hai → **answer = 1** ✅

⏱️ Time & 💾 Space

- **Time:** O(n) — har element apni jagah par max ek-do swaps me pahunchta hai; total linear.
- **Space:** O(1) — in-place swaps; extra arrays nahi.

🧩 Quick Visual (index vs value)

For `arr = [3,4,-1,1]`

```
Index:  0  1  2  3
Value:  3  4 -1  1
Target: 2  3  -  0   (x goes to x-1)
Moves: 0↔2, 1↔3, 1↔0
Final:  [1,-1,3,4] → first gap at index 1 ⇒ 2
```

Example: nums = [3, 4, -1, 1]

Goal: smallest missing positive integer ✅

Initial State

```
Index:   0   1   2   3
Value:   3   4  -1   1
```

Step-by-Step with Emojis & Arrows

🔄 Step 1: i = 0, arr[0] = 3

- ✅ Positive & in range → correct index = `3 - 1 = 2`
- `arr[0] != arr[2]` → SWAP(0, 2)

```
[ 3, 4, -1, 1 ]
   🔄 swap
[ -1, 4, 3, 1 ]
```

👉 Now i = 0 again

➡️ Step 2: i = 0, arr[0] = -1

- ❌ arr[0] <= 0 → skip → i++

```
i = 1
```

🔄 Step 3: i = 1, arr[1] = 4

- ✅ Positive & in range → correct index = `4 - 1 = 3`
- `arr[1] != arr[3]` → SWAP(1, 3)

```
[ -1, 4, 3, 1 ]
   🔄 swap
[ -1, 1, 3, 4 ]
```

👉 **Now i = 1 again**

🔄 Step 4: i = 1, arr[1] = 1

- correct index = `0`
- `arr[1] != arr[0]` → SWAP(1, 0)

```
[ -1, 1, 3, 4 ]
   🔄 swap
[  1, -1, 3, 4 ]
```

👉 **Now i = 1 again**

➡️ Step 5: i = 1, arr[1] = -1

- ❌ arr[1] <= 0 → skip → i++

```
i = 2
```

➡️ Step 6: i = 2, arr[2] = 3

- ✅ already at correct place → i++

```
i = 3
```

➡️ Step 7: i = 3, arr[3] = 4

- ✅ already at correct place → i++

```
i = 4 (done)
```

✅ Rearranged Array

```
[ 1, -1, 3, 4 ]
```

Final Check 🔍

```
Index 0 → 1 ✅
Index 1 → expected 2, got -1 ❌ → ANSWER = 2
```

🎯 **Output: 2** ✅

Visual Flow with Emoji

```
Start:    [3️⃣, 4️⃣, -1️⃣, 1️⃣]
Swap 0-2: [-1️⃣, 4️⃣, 3️⃣, 1️⃣]
Swap 1-3: [-1️⃣, 1️⃣, 3️⃣, 4️⃣]
Swap 1-0: [1️⃣, -1️⃣, 3️⃣, 4️⃣]
Check:    Missing → 2️⃣ ✅
```
