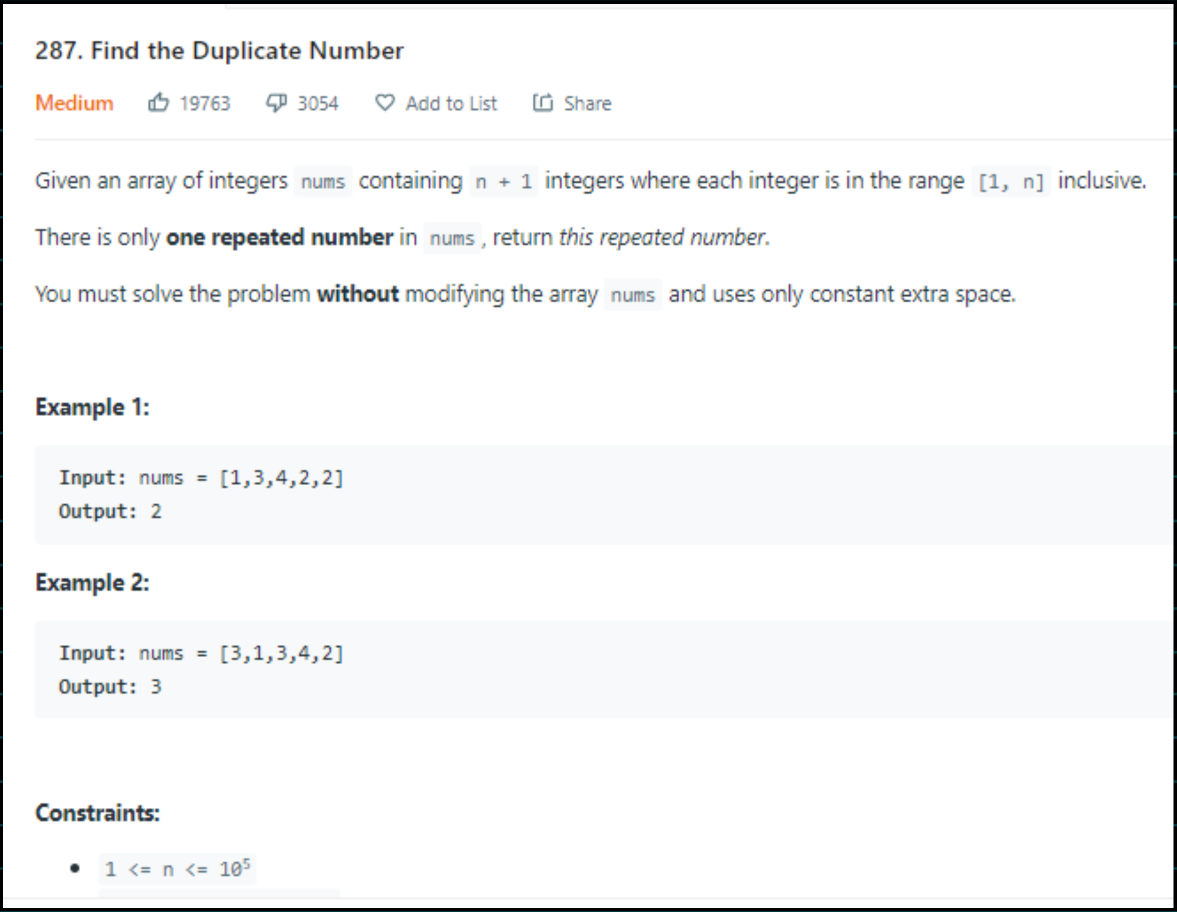
Day3 Arsh Goyal challenges

Find the duplicate number



287. Find the Duplicate Number

Medium 19763 3054 Add to List Share

Given an array of integers `nums` containing `n + 1` integers where each integer is in the range `[1, n]` inclusive.

There is only **one repeated number** in `nums`, return *this repeated number*.

You must solve the problem **without** modifying the array `nums` and uses only constant extra space.

Example 1:

```
Input: nums = [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: nums = [3,1,3,4,2]
Output: 3
```

Constraints:

- `1 <= n <= 10^5`

Brute force :

It is given in the question that there is only one element in Array which is repeated number, so we can take advantage and Sort the Array and find that element

nums = [1, 3, 4, 2, 2]

Sorted nums = [1, 2, 2, 3, 4]

$\longrightarrow$ traverse linearly and find out which element is repeated

[1, 2, 2, 3, 4]

i-1 ↗ ↗ i

T.C - O(nlogn) + O(n)

↓ sort ↓ linear traverse

S.C - O(1)

[1, 2, 2, 3, 4]

i-1 ↗ ↗ i

nums[i-1] = num[i] return num[i];

Better approch :- (unorderedmap)

To optimize our T.C we will be using unordered map (extra space)

Using unordered map we will be storing their frequency. and element who frequency more than 2 will be our repeated number

$nums = [1, 3, 4, 2, 2]$

1 :	1
3 :	1
4 :	1
2 :	2

$\rightarrow$ repeated number

T.C - $O(n)$

S.C - $O(n)$

Optimal approch : (Fast & Slow)

If we look at our constraint it is given that array element lie between 1 to <= n i.e $1 \leq num[i] \leq n$

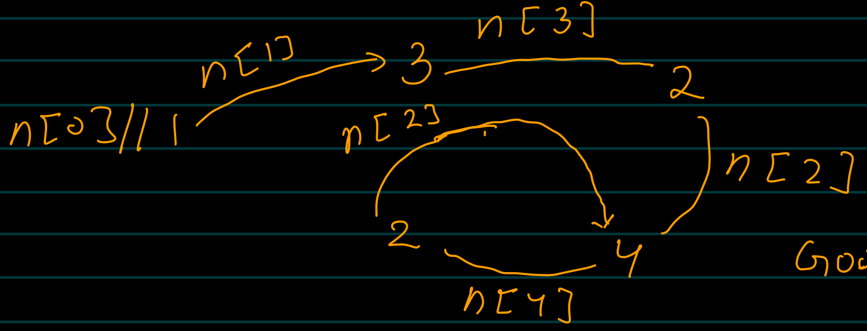
and there is alway a duplicate in Array.

so, we say that Array must be forming loop

How?

$$\begin{array}{cccccc} n = [& 1, & 3, & 4, & 2, & 2] \\ & 0 & 1 & 2 & 3 & 4 \end{array}$$

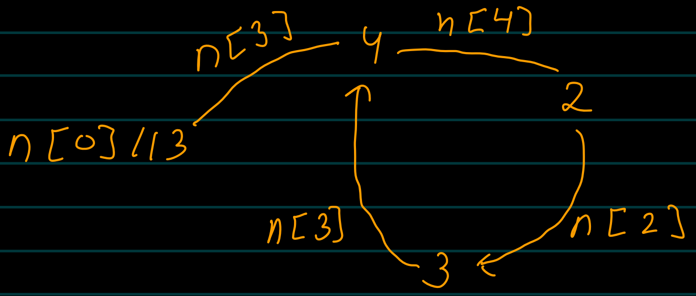
starting from 0



Got the cycle

Another example

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 \\ n = [& 3, & 1, & 3, & 4, & 2] \end{array}$$



Got cycle

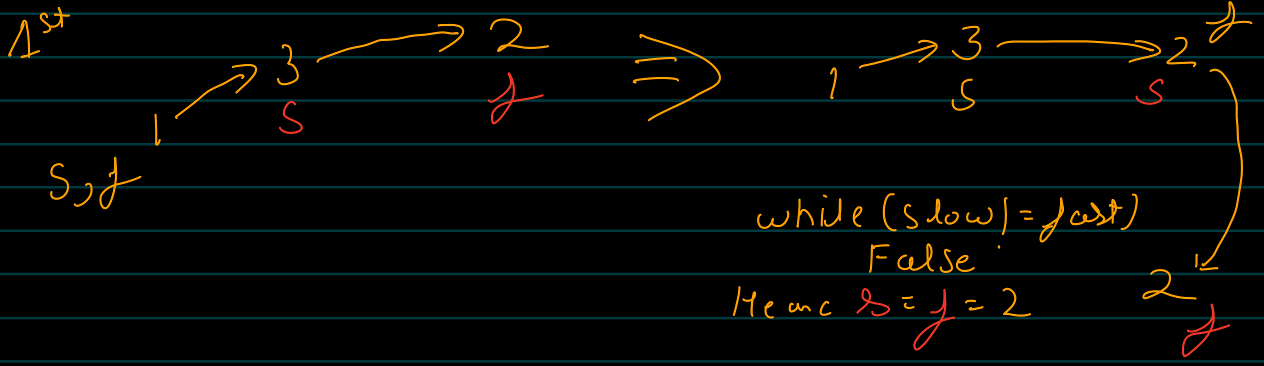
Now, there is cycle we can use fast and slow pointer at first they will start from 0 index and run a loop where until they collide (ie slow != fast)

After collision, move fast pointer to 0 index and move slow & fast with same speed of 1 (ie slow = nums[slow], fast = nums[fast])

$$\begin{array}{cccccc} n = [& 1, & 3, & 4, & 2, & 2] \\ & 0 & 1 & 2 & 3 & 4 \end{array}$$

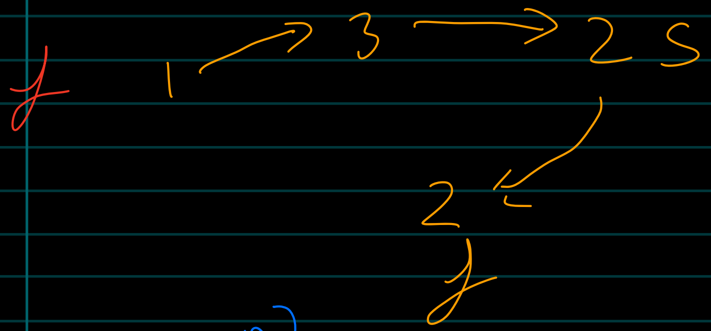
S = n[1], F = n[n[1]]

S = n[3], F = n[n[2]] // ~~n[4]~~ 2



while (slow != fast)
False
Hence S = f = 2

Shift fast to 0 index ie 1



move both ptr with Single step

S = n[S]

f = n[f]

T.C - O(n)
S.C - O(1)

where ever they collide again will be repeated element.

EXPERIMENT NO : 8

```
[1, 3, 4, 2, 2]
 0  1  2  3  4
slow = 1
Fast = 1

slow = nums [slow]
Fast = nums [ nums[Fast]]
```

```
slow = n[1] // 3
Fast = n [ n[1]]
         n [ 3] // 2

slow = n [3]
Fast = n [n [2]]
       n [4] // 2

slow = 2, Fast = 2
while (slow != Fast)
                False
```

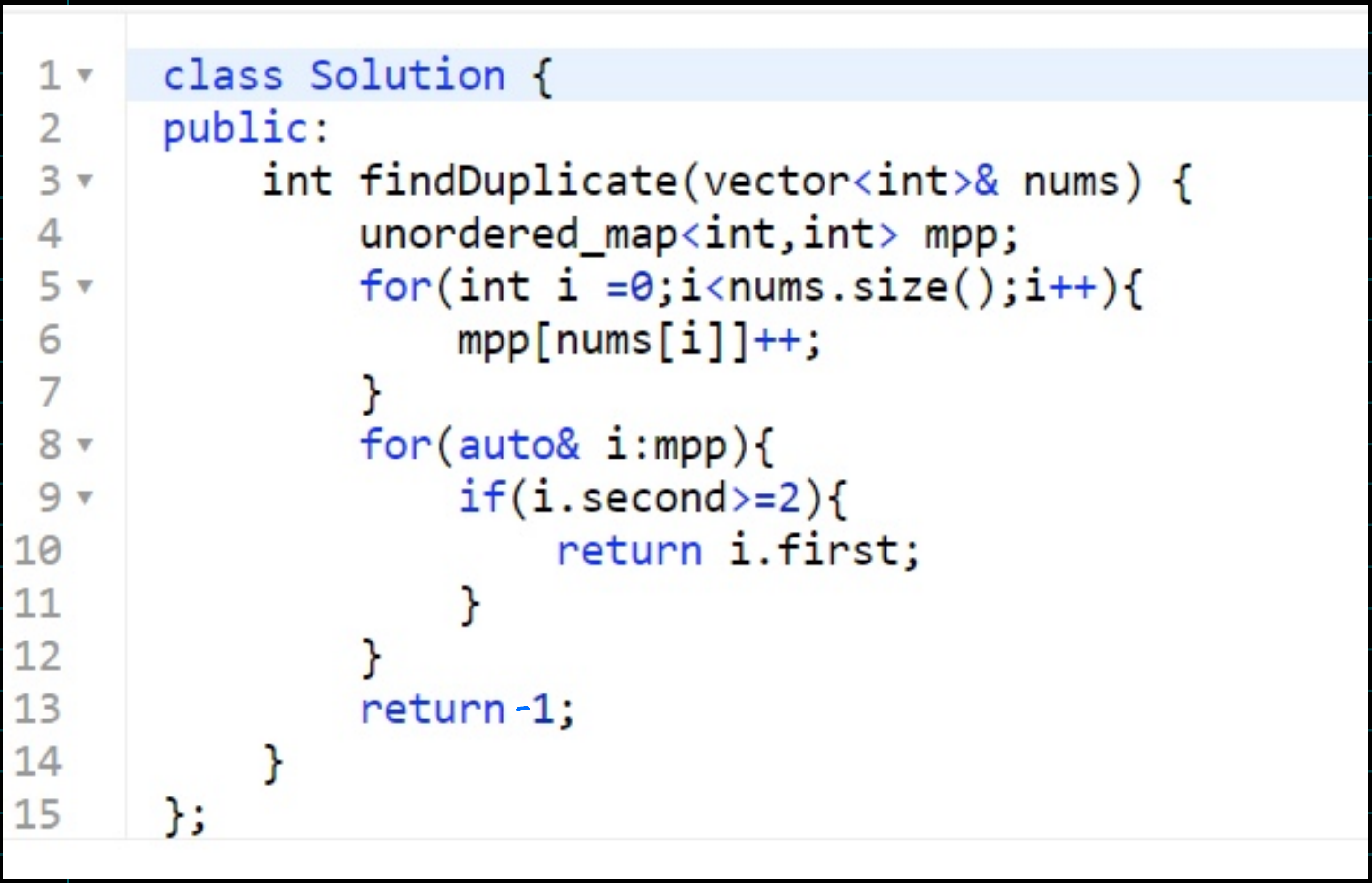
```
Fast = num [0] // 1     and slow = ~~7~~ 4
while( slow != Fast) {

   slow = num [slow]
   Fast = nums [Fast]
```

```
~~S = num~~ slow = n[2] // 4
      Fast = n[1] // 3
   slow = n[4] // 2
   Fast = n [3] // 2
~~while~~
   while(slow != Fast)
              False
```

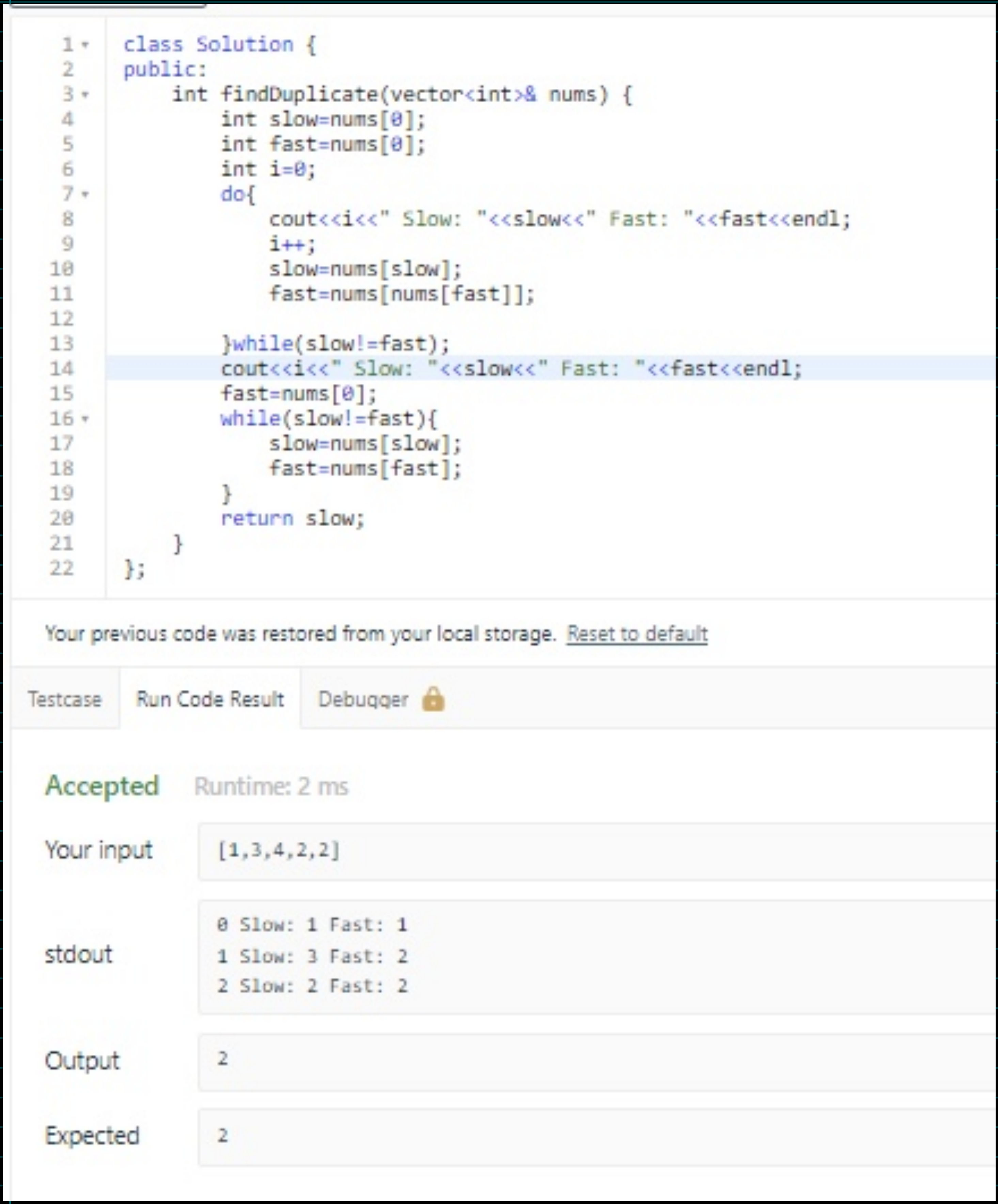
Return slow

Better approach code



```cpp
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        unordered_map<int,int> mpp;
        for(int i =0;i<nums.size();i++){
            mpp[nums[i]]++;
        }
        for(auto& i:mpp){
            if(i.second>=2){
                return i.first;
            }
        }
        return -1;
    }
};
```

Optimal approach code



```cpp
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        int slow=nums[0];
        int fast=nums[0];
        int i=0;
        do{
            cout<<i<<" Slow: "<<slow<<" Fast: "<<fast<<endl;
            i++;
            slow=nums[slow];
            fast=nums[nums[fast]];

        }while(slow!=fast);
        cout<<i<<" Slow: "<<slow<<" Fast: "<<fast<<endl;
        fast=nums[0];
        while(slow!=fast){
            slow=nums[slow];
            fast=nums[fast];
        }
        return slow;
    }
};
```

Your previous code was restored from your local storage. Reset to default

Accepted Runtime: 2 ms

Your input
```
[1,3,4,2,2]
```

stdout
```
0 Slow: 1 Fast: 1
1 Slow: 3 Fast: 2
2 Slow: 2 Fast: 2
```

Output
```
2
```

Expected
```
2
```