

Find the Duplicate Number (easy)

We'll cover the following ^

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity
- Similar Problems

Problem Statement

We are given an unsorted array containing 'n+1' numbers taken from the range 1 to 'n'. The array has only one duplicate but it can be repeated multiple times. **Find that duplicate number without using any extra space.** You are, however, allowed to modify the input array.

Example 1:

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

Example 2:


```
Input: [2, 1, 3, 3, 5, 4]
Output: 3
```


Example 3:


```
Input: [2, 4, 1, 4, 4]
Output: 4
```


Try it yourself

Try solving this question here:

 Java

 Python3

 JS


 C++

```
1 def find_duplicate(nums):
2     # TODO: Write your code here
3     return -1
4
```

Test

Save

Reset



Solution

This problem follows the **Cyclic Sort** pattern and shares similarities with [Find the Missing Number](#). Following a similar approach, we will try to place each number on its correct index. Since there is only one duplicate, if while swapping the number with its index both the numbers being swapped are same, we have found our duplicate!

Code

Here is what our algorithm will look like:

Java Python3 C++ JS

```
1
2 def find_duplicate(nums):
3     i = 0
4     while i < len(nums):
5         if nums[i] != i + 1:
6             j = nums[i] - 1
7             if nums[i] != nums[j]:
8                 nums[i], nums[j] = nums[j], nums[i] # swap
9             else: # we have found the duplicate
10                return nums[i]
11        else:
12            i += 1
13    return -1
14
15
16
17 def main():
18     print(find_duplicate([1, 4, 4, 3, 2]))
19     print(find_duplicate([2, 1, 3, 3, 5, 4]))
20     print(find_duplicate([2, 4, 1, 4, 4]))
21
22
23 main()
24
```

Run Save Reset

Time complexity

The time complexity of the above algorithm is $O(n)$.

Space complexity

The algorithm runs in constant space $O(1)$ but modifies the input array.

Similar Problems

Problem 1: Can we solve the above problem in $O(1)$ space and without modifying the input array?

Solution: While doing the cyclic sort, we realized that the array will have a cycle due to the duplicate number and that the start of the cycle will always point to the duplicate number. This means that we can use the fast & the slow pointer method to find the duplicate number or the start of the cycle similar to [Start of LinkedList Cycle](#).

Java Python3 C++ JS

```
1 def find_duplicate(arr):
2     slow, fast = arr[0], arr[arr[0]]
3     while slow != fast:
4         slow = arr[slow]
5         fast = arr[arr[fast]]
6
7     # find cycle length
8     current = arr[arr[slow]]
9     cycleLength = 1
10    while current != arr[slow]:
11        current = arr[current]
12        cycleLength += 1
13
14    return find_start(arr, cycleLength)
15
16
17 def find_start(arr, cycleLength):
18     pointer1, pointer2 = arr[0], arr[0]
19     # move pointer2 ahead 'cycleLength' steps
20     while cycleLength > 0:
21         pointer2 = arr[pointer2]
22         cycleLength -= 1
23
24     # increment both pointers until they meet at the start of the cycle
25     while pointer1 != pointer2:
```

```
25     while pointer1 != pointer2:
26         pointer1 = arr[pointer1]
27         pointer2 = arr[pointer2]
28
29     return pointer1
30
31
32 def main():
33     print(find_duplicate([1, 4, 4, 3, 2]))
34     print(find_duplicate([2, 1, 3, 3, 5, 4]))
35     print(find_duplicate([2, 4, 1, 4, 4]))
36
37
38 main()
39
```

Run

Save

Reset

The time complexity of the above algorithm is $O(n)$ and the space complexity is $O(1)$.


[← Back](#)

Find all Missing Numbers (easy)

[Next →](#)

Find all Duplicate Numbers (easy)

✓ Completed

 Report an Issue  Ask a Question