Find the Duplicate Number (easy)

We'll cover the following

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++

```
const find_duplicate = function(nums) {
  // TODO: Write your code here
  return -1;
};

```

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!

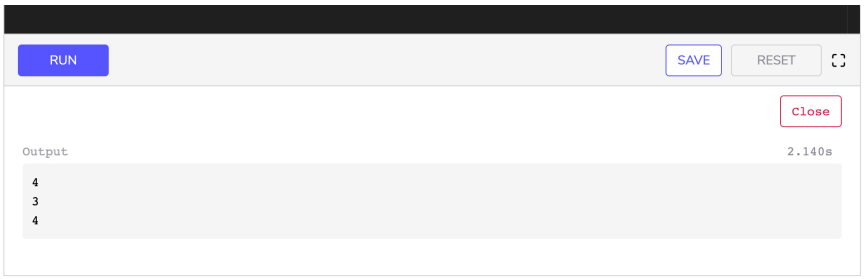
Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```
function find_duplicate(nums) {
  let i = 0;
  while (i < nums.length) {
    if (nums[i] !== i + 1) {
      j = nums[i] - 1;
      if (nums[i] !== nums[j]) {
        [nums[i], nums[j]] = [nums[j], nums[i]]; // swap
      } else { // we have found the duplicate
        return nums[i];
      }
    } else {
      i += 1;
    }
  }
  return -1;
}


console.log(find_duplicate([1, 4, 4, 3, 2]));
console.log(find_duplicate([2, 1, 3, 3, 5, 4]));
console.log(find_duplicate([2, 4, 1, 4, 4]));
```



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

```
function find_duplicate(arr) {
  let slow = arr[0];
  fast = arr[arr[0]];
  while (slow !== fast) {
    slow = arr[slow];
    fast = arr[arr[fast]];
  }
  // find cycle length
  let current = arr[arr[slow]];
  let cycleLength = 1;
  while (current !== arr[slow]) {
    current = arr[current];
    cycleLength += 1;
  }

  return find_start(arr, cycleLength);
}

function find_start(arr, cycleLength) {
  let pointer1 = arr[0];
  let pointer2 = arr[0];
  // move pointer2 ahead 'cycleLength' steps
  while (cycleLength > 0) {
    pointer2 = arr[pointer2];
    cycleLength -= 1;
  }
  // increment both pointers until they meet at the start of the cycle
  while (pointer1 !== pointer2) {
```

RUN SAVE RESET

Close

Output 1.661s

```
4
3
4
```

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