December 9, 2021 ▪ Arrays / Data Structure

Find the duplicate in an array of N+1 integers

Problem Statement: Given an array of N + 1 size, where each element is between 1 and N. Assuming there is only one duplicate number, your task is to find the duplicate number.

Examples:

```
Example 1:

Input: arr=[1,3,4,2,2]

Output: 2

Explanation: Since 2 is the duplicate number the answer will be 2.

Example 2:

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

Output:3

Explanation: Since 3 is the duplicate number the answer will be 3.
```

Solution

Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Solution 1:Using sorting

Approach: Sort the array. After that, if there is any duplicate number they will be adjacent.So we simply have to check if arr[i]==arr[i+1] and if it is true,arr[i] is the duplicate number.

Code:

C++Code

```cpp
#include<bits/stdc++.h>

using namespace std;
int findDuplicate(vector < int > & arr) {
  int n = arr.size();
  sort(arr.begin(), arr.end());
  for (int i = 0; i < n - 1; i++) {
    if (arr[i] == arr[i + 1]) {
      return arr[i];
    }
  }
}
int main() {
  vector < int > arr;
  arr = {1,3,4,2,2};
  cout << "The duplicate element is " << findDuplicate(arr) << endl;
}
```

Output: The duplicate element is 2

Time Complexity:O(Nlogn + N)

Reason: NlogN for sorting the array and O(N) for traversing through the array and checking if adjacent elements are equal or not. But this will distort the array.

Space Complexity:O(1)

Java Code

```java
import java.util.*;
class TUF {
    static int findDuplicate(int[] arr) {
        int n = arr.length;
        Arrays.sort(arr);
        for (int i = 0; i < n - 1; i++) {
            if (arr[i] == arr[i + 1]) {
                return arr[i];
```

```
            }
        }
        return 0;
    }
    public static void main(String args[]) {
        int arr[] = {1,3,4,2,2};
        System.out.println("The duplicate element is " + findDuplicate(arr));
    }
}
```

Output: The duplicate element is 2

Time Complexity:O(Nlogn + N)

Reason: NlogN for sorting the array and O(N) for traversing through the array and checking if adjacent elements are equal or not. But this will distort the array.

Space Complexity:O(1)

Python Code

```
from typing import List



def findDuplicate(arr: List[int]) -> int:
    n = len(arr)
    arr.sort()
    for i in range(n - 1):
        if arr[i] == arr[i + 1]:
            return arr[i]


if __name__ == "__main__":
    arr = [1, 3, 4, 2, 2]
    print("The duplicate element is ", findDuplicate(arr))
```

Output: The duplicate element is 2

Time Complexity:O(Nlogn + N)

Reason: NlogN for sorting the array and O(N) for traversing through the array and checking if adjacent elements are equal or not. But this will distort the array.

Space Complexity:O(1)

Solution 2:Using frequency array

Approach: Take a frequency array of size N+1 and initialize it to 0. Now traverse through the array and if the frequency of the element is 0 increase it by 1, else if the frequency is not 0 then that element is the required answer.

Code:

C++ Code

```
#include<bits/stdc++.h>

using namespace std;
int findDuplicate(vector < int > & arr) {
  int n = arr.size();
  int freq[n + 1] = {
    0
  };
  for (int i = 0; i < n; i++) {
    if (freq[arr[i]] == 0) {
      freq[arr[i]] += 1;
    } else {
      return arr[i];
    }
  }
  return 0;
}
int main() {
  vector < int > arr;
  arr = {1,3,4,2,3};
  cout << "The duplicate element is " << findDuplicate(arr) << endl;
}
```

Output: The duplicate element is 3

Time Complexity: O(N), as we are traversing through the array only once.

Space Complexity: O(N), as we are using extra space for frequency array.

Java Code

```java
import java.util.*;
class TUF {
    static int findDuplicate(int[] arr) {
        int n = arr.length;
        int freq[] = new int[n + 1];
        for (int i = 0; i < n; i++) {
            if (freq[arr[i]] == 0) {
                freq[arr[i]] += 1;
            } else {
                return arr[i];
            }
        }
        return 0;
    }
    public static void main(String args[]) {
        int arr[] = {1,3,4,2,3};
        System.out.println("The duplicate element is " + findDuplicate(arr));
    }
}
```

Output: The duplicate element is 3

Time Complexity: O(N), as we are traversing through the array only once.

Space Complexity: O(N), as we are using extra space for frequency array.

Python Code

```python
from typing import List
def findDuplicate(arr: List[int]) -> int:
    n = len(arr)
    freq = [0] * (n + 1)
    for i in range(n):
        if freq[arr[i]] == 0:
            freq[arr[i]] += 1
        else:
            return arr[i]
    return 0


if __name__ == "__main__":
    arr = [1, 3, 4, 2, 3]
    print("The duplicate element is ", findDuplicate(arr))
```

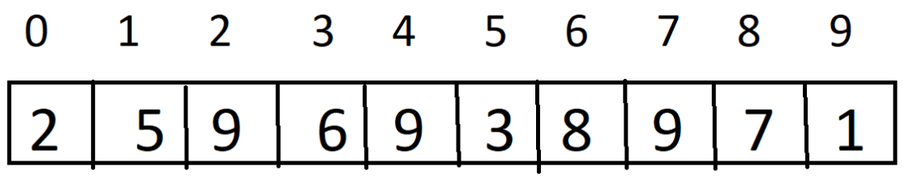
Output: The duplicate element is 3

Time Complexity: O(N), as we are traversing through the array only once.

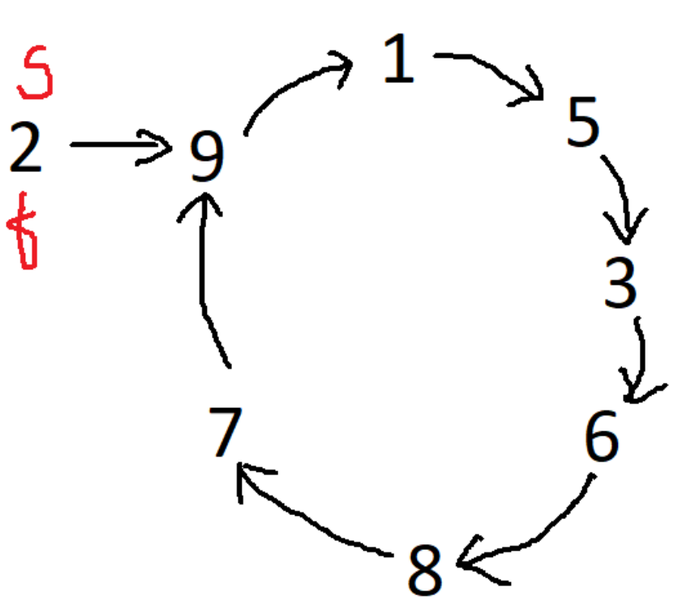
Space Complexity: O(N), as we are using extra space for frequency array.

Solution 3: [Linked List cycle method]

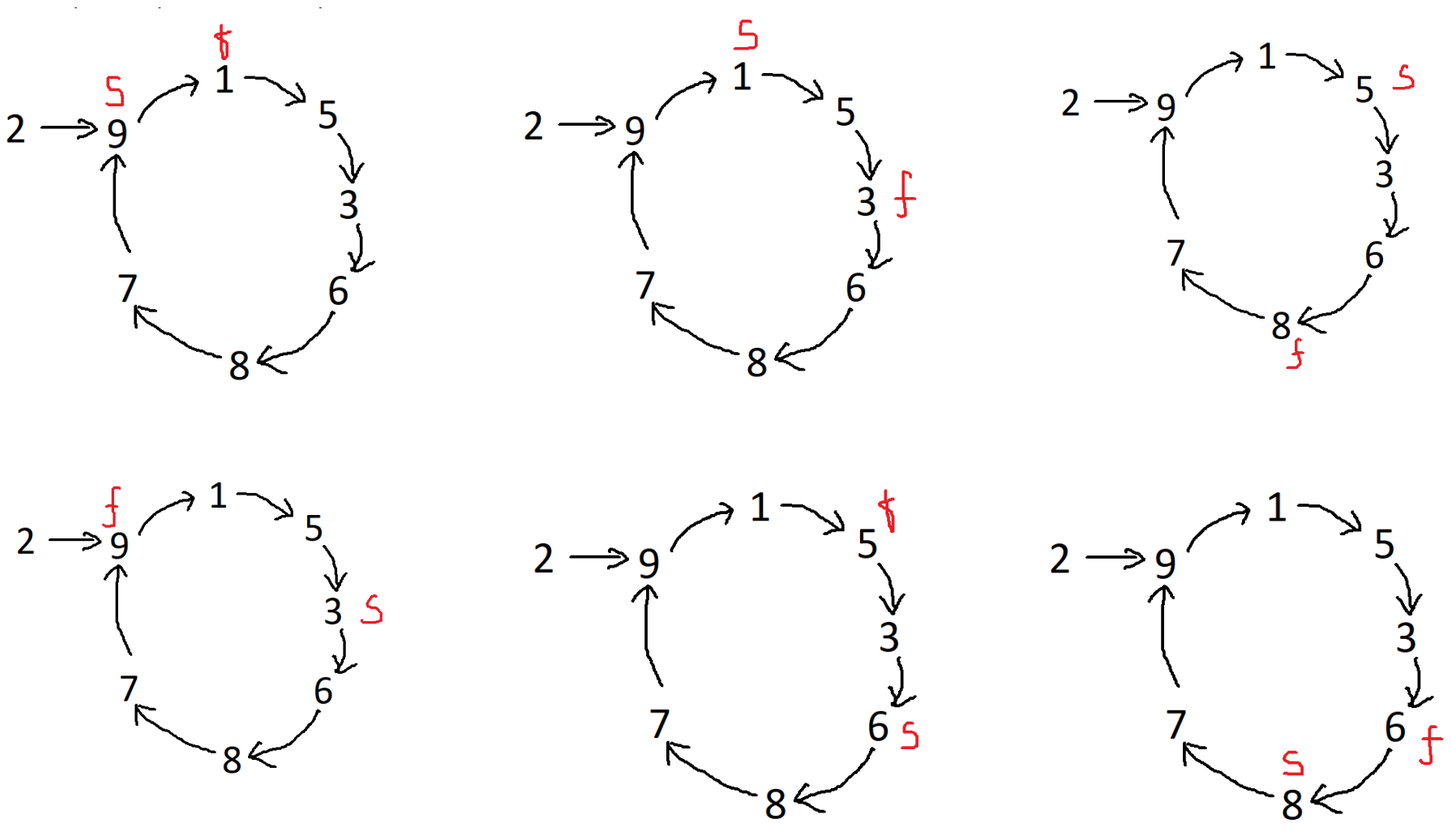
Approach: Let's take an example and dry run on it to understand.

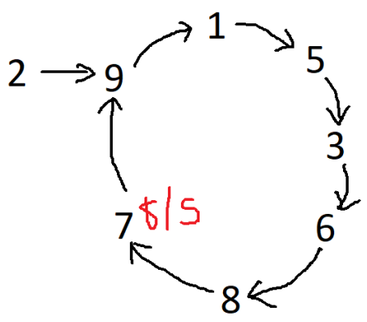


Initially, we have 2, then we got to the second index, we have 9, then we go to the 9th index, we have 1, then we go to the 1st index, we again have 5, then we go to the 5th index, we have 3, then we go to the 3rd index, we get 6, then we go to the 6th index, we get 8, then we go to the 8th index, we get 7, then we go to the 7th index and we get 9 and here cycle is formed.



Now initially, the slow and fast pointer is at the start, the slow pointer moves by one step, and the fast pointer moves by 2 steps.



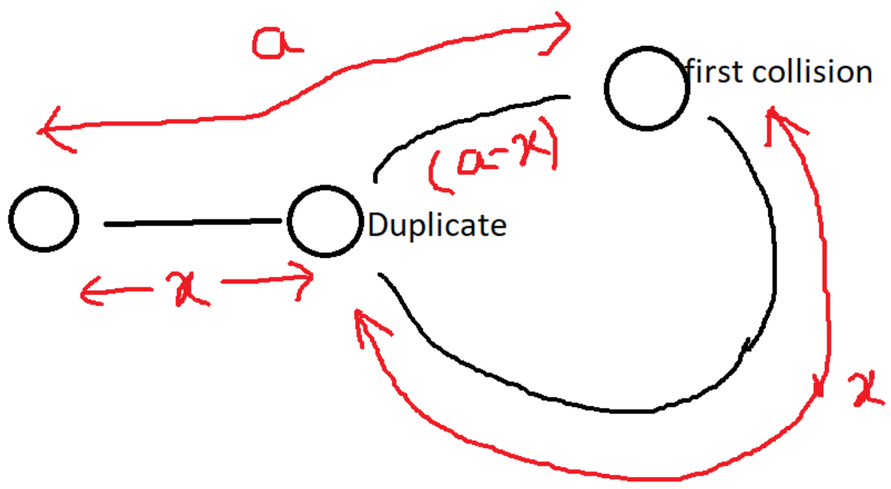


The slow and fast pointers meet at 7. Now take the fast pointer and place it at the first element i.e 2 and move the fast and slow pointer both by 1 step. The point where they collide will be the duplicate number.

So 9 is the duplicate number.

Intuition: Since there is a duplicate number, we can always say that cycle will be formed.

The slow pointer moves by one step and the fast pointer moves by 2 steps and there exists a cycle so the first collision is bound to happen.



Let's assume the distance between the first element and the first collision is a. So slow pointer has traveled a distance while fast(since moving 2 steps at a time) has traveled 2a distance. For slow and a fast pointer to collide 2a-a=a should be multiple of the length of cycle, Now we place a fast pointer to start. Assume the distance between the start and duplicate to be x. So now the distance between slow and duplicate shows also be x, as seen from the diagram, and so now fast and slow pointer both should move by one step.

Code:

C++ Code

```
#include<bits/stdc++.h>

using namespace std;
int findDuplicate(vector < int > & nums) {
  int slow = nums[0];
  int fast = nums[0];
  do {
    slow = nums[slow];
    fast = nums[nums[fast]];
  } while (slow != fast);
  fast = nums[0];
```

```
  while (slow != fast) {
    slow = nums[slow];
    fast = nums[fast];
  }
  return slow;
}
int main() {
  vector < int > arr;
  arr = {1,3,4,2,3};
  cout << "The duplicate element is " << findDuplicate(arr) << endl;
}
```

Output:

The duplicate element is 3

Time complexity: O(N). Since we traversed through the array only once.

Space complexity: O(1).

Java Code

```
import java.util.*;
class TUF {
    public static int findDuplicate(int[] nums) {
        int slow = nums[0];
        int fast = nums[0];
        do {
            slow = nums[slow];
            fast = nums[nums[fast]];
        } while (slow != fast);

        fast = nums[0];
        while (slow != fast) {
            slow = nums[slow];
            fast = nums[fast];
        }
        return slow;
    }
    public static void main(String args[]) {
        int arr[] = {1,3,4,2,3};
        System.out.println("The duplicate element is " + findDuplicate(arr));
    }
}
```

Output:

The duplicate element is 3

Time complexity: O(N). Since we traversed through the array only once.

Space complexity: O(1).

Python Code

```
from typing import List
def findDuplicate(nums: List[int]) -> int:
    slow = nums[0]
    fast = nums[0]
    while True:
        slow = nums[slow]
        fast = nums[nums[fast]]
        if slow == fast:
            break
    fast = nums[0]
    while slow != fast:
        slow = nums[slow]
        fast = nums[fast]
    return slow


if __name__ == "__main__":
    arr = [1, 3, 4, 2, 3]
    print("The duplicate element is ", findDuplicate(arr))
```

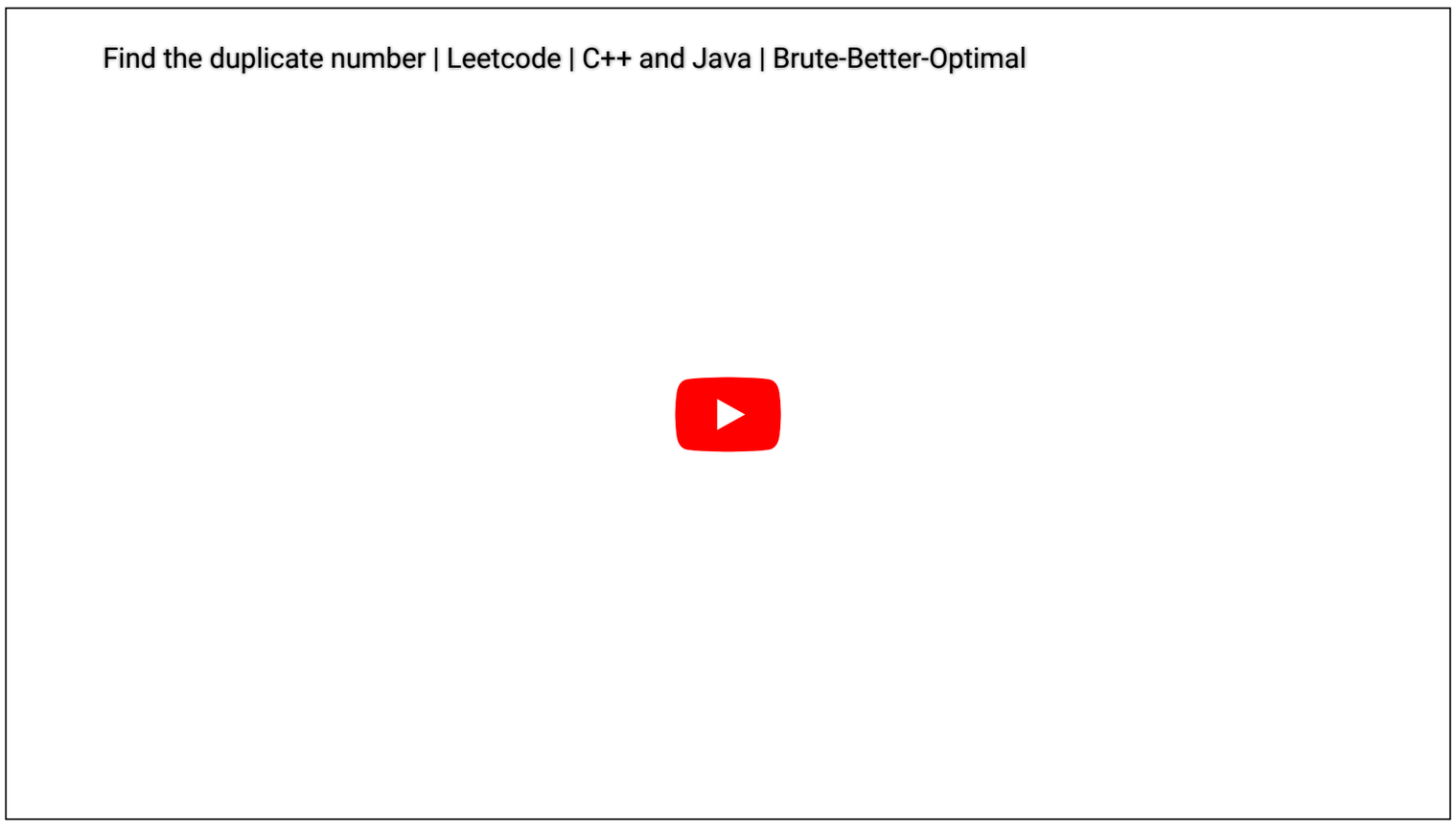
Output:

The duplicate element is 3

Time complexity: O(N). Since we traversed through the array only once.

Space complexity: O(1).

> Special thanks to **Pranav Padawe** and **Sudip Ghosh** for contributing to this article on takeUforward. If you also wish to share your knowledge with the takeUforward fam, please check out this article



« Previous Post
Find K-th Permutation Sequence

Next Post »
Trapping Rainwater

Load Comments