Find duplicate

Problem Description: You are given with an array of integers of size n which contains numbers from 0 to n - 2. Each number is present at least once. That is, if n = 5, numbers from 0 to 3 is present in the given array at least once and one number is present twice. You need to find and return that duplicate number present in the array.

How to approach?

Approach 1: In this question you need to run two loops, pick an element from the first loop and then in the inner loop check if the element appears once again or not, if yes then return that element, otherwise move to the next element.
This method doesn't use the other useful data provided in questions like range of numbers is between 0 to n-2 and hence, it is increasing the time complexity.

Pseudo Code for this approach:

```
Function Findduplicate:
    For i = 0 to i less than size:
        For j = 0 to j less than size:
            If i not equal to j and arr[i] is equal to arr[j]:
                Return arr[i]
    return minus infinity
```

Time Complexity for this approach: Time complexity for this approach is O(n^2), which is not good, hence we move to the next approach.

Approach 2: A better solution for this problem can be by using XOR operator. Using XOR operator, we can solve this problem in one traversal only. The following facts about XOR operation will be useful for this question:

1. If we XOR a number by itself, even number of times then it will give you 0.
2. If we XOR a number with itself, odd number of times, then it will give you the number itself.
3. Also XOR of a number with 0 gives you that number again.

So, if we take XOR of all the elements present in the array with every element in the range 0 to n-2, then all the elements of that array except the duplicate element are XORed 2 times and

hence, their resultant is 0. But the duplicate element is XORed 3 times, hence, its resultant is the number itself. Hence, you will get you answer as the duplicate number present in the array.

For example, if you are given with n=5 and let us say array is 0 1 3 2 2, then according to this approach, we have to XOR all elements present in the array with every element in the range 0 to 3.

Answer = (0^1^3^2^2)^(0^1^2^3)
As XOR operation is associative and commutative, so, by rearranging
Answer = (0^0) ^ (1^1) ^ (2^2^2) ^ (3^3)
= 0 ^ 0 ^ 2 ^ 0
= 2

Pseudo Code for this approach:

```
Function Findduplicate:
    answer=0
    For i =0 to i less than n:
        answer=answer xor arr[i]
    For i=0 to i less than or equal to n-2:
        answer=answer xor i
    Return answer
```

Time Complexity for this approach: Time complexity for this approach is O(n) as you are traversing the array only once for XORing.

Approach 3: Another approach is to make use of the condition that all elements lies between 0 and n-2. So first calculate the sum of all natural numbers between 0 to n-2 by using the direct formula *((n - 1) * (n - 2)) / 2* and sum of all elements of the array. Now, subtract the sum of all natural numbers between 0 to n-2 from sum of all elements of the array. This will give you the duplicate element present in the array.

Pseudo Code for this approach:

```
Function findduplicate:
    sum=0
   For i = 0 to i less than size:
        sum = sum + input[i];
  n = size
 sumOfNaturalNumbers = ((n - 1) * (n - 2)) / 2
```

return sum - sumOfNaturalNumbers

Time Complexity for this approach: Time complexity for this approach is O(n) as you are traversing the array only once to calculate the sum of all elements present in the array.

- Let us dry run the code for the N= 9

  arr[]= 0 7 2 5 4 7 1 3 6

  Sum = 0+7+2+5+4+7+1+3+6 =35
  sumOfNaturalNumbers=8*7/2=28
  Output = 35-28 =7
  So 7 should get printed.