

## Find the repeating and the missing | Added 3 new methods

Given an unsorted array of size n. Array elements are in range from 1 to n. One number from set {1, 2, ...n} is missing and one number occurs twice in array. Find these two numbers.

Examples:

```
arr[] = {3, 1, 3}
Output: 2, 3 // 2 is missing and 3 occurs twice
```

```
arr[] = {4, 3, 6, 2, 1, 1}
Output: 1, 5 // 5 is missing and 1 occurs twice
```

### Method 1 (Use Sorting)

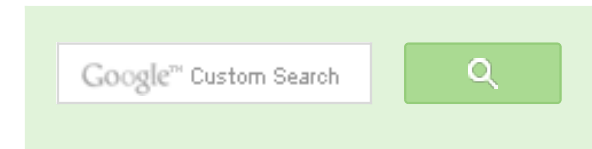
- 1) Sort the input array.
- 2) Traverse the array and check for missing and repeating.

Time Complexity:  $O(n \log n)$

Thanks to LoneShadow for suggesting this method.

### Method 2 (Use count array)

- 1) Create a temp array temp[] of size n with all initial values as 0.
- 2) Traverse the input array arr[], and do following for each arr[i]
  - .....a) if(temp[arr[i]] == 0) temp[arr[i]] = 1;
  - .....b) if(temp[arr[i]] == 1) output "arr[i]" //repeating
- 3) Traverse temp[] and output the array element having value as 0 (This is the missing element)



GeeksforGeeks



53,520 people like GeeksforGeeks.



Interview Experiences

Advanced Data Structures

Dynamic Programming

Greedy Algorithms

Backtracking

Pattern Searching

Divide & Conquer

Mathematical Algorithms

Recursion

Geometric Algorithms

Time Complexity:  $O(n)$ Auxiliary Space:  $O(n)$ **Method 3 (Use elements as Index and mark the visited places)**

Traverse the array. While traversing, use absolute value of every element as index and make the value at this index as negative to mark it visited. If something is already marked negative then this is the repeating element. To find missing, traverse the array again and look for a positive value.

```
#include<stdio.h>
#include<stdlib.h>

void printTwoElements(int arr[], int size)
{
    int i;
    printf("\n The repeating element is");

    for(i = 0; i < size; i++)
    {
        if(arr[abs(arr[i])-1] > 0)
            arr[abs(arr[i])-1] = -arr[abs(arr[i])-1];
        else
            printf(" %d ", abs(arr[i]));
    }

    printf("\nand the missing element is ");
    for(i=0; i<size; i++)
    {
        if(arr[i]>0)
            printf("%d", i+1);
    }
}

/* Driver program to test above function */
int main()
{
    int arr[] = {7, 3, 4, 5, 5, 6, 2};
    int n = sizeof(arr)/sizeof(arr[0]);
    printTwoElements(arr, n);
    return 0;
}
```

Time Complexity:  $O(n)$ 

Thanks to Manish Mishra for suggesting this method.

# ITT Tech - Official Site

[itt-tech.edu](http://itt-tech.edu)

Associate, Bachelor Degree  
Programs Browse Programs Now &  
Learn More.

## Popular Posts

[All permutations of a given string](#)
[Memory Layout of C Programs](#)
[Understanding "extern" keyword in C](#)
[Median of two sorted arrays](#)
[Tree traversal without recursion and without stack!](#)
[Structure Member Alignment, Padding and](#)
[Data Packing](#)
[Intersection point of two Linked Lists](#)
[Lowest Common Ancestor in a BST.](#)
[Check if a binary tree is BST or not](#)
[Sorted Linked List to Balanced BST](#)

#### Method 4 (Make two equations)

Let x be the missing and y be the repeating element.

1) Get sum of all numbers.

Sum of array computed  $S = n(n+1)/2 - x + y$

2) Get product of all numbers.

Product of array computed  $P = 1*2*3*...n * y / x$

3) The above two steps give us two equations, we can solve the equations and get the values of x and y.

Time Complexity:  $O(n)$

Thanks to disappearedng for suggesting this solution.

This method can cause arithmetic overflow as we calculate product and sum of all array elements. See [this](#) for changes suggested by [john](#) to reduce the chances of overflow.

#### Method 5 (Use XOR)

Let x and y be the desired output elements.

Calculate XOR of all the array elements.

```
xor1 = arr[0]^arr[1]^arr[2].....arr[n-1]
```

XOR the result with all numbers from 1 to n

```
xor1 = xor1^1^2^.....^n
```

In the result *xor1*, all elements would nullify each other except x and y. All the bits that are set in *xor1* will be set in either x or y. So if we take any set bit (We have chosen the rightmost set bit in code) of *xor1* and divide the elements of the array in two sets – one set of elements with same bit set and other set with same bit not set. By doing so, we will get x in one set and y in another set. Now if we do XOR of all the elements in first set, we will get x, and by doing same in other set we will get y.

```
#include <stdio.h>
#include <stdlib.h>
```

```
/* The output of this function is stored at *x and *y */
void getTwoElements(int arr[], int n, int *x, int *y)
```



## Recent Comments

**Aman** Hi, Why arent we checking for conditions...

Write a C program to Delete a Tree. · 13 minutes ago

kzs please provide solution for the problem...

Backtracking | Set 2 (Rat in a Maze) · 17 minutes ago

**Sanjay Agarwal** bool

tree::Root\_to\_leaf\_path\_given\_sum(tree...

Root to leaf path sum equal to a given number · 42 minutes ago

**GOPI GOPINATH** @admin Highlight this sentence "We can easily...


Count trailing zeroes in factorial of a number · 44 minutes ago

**newCoder3006** If the array contains negative numbers also. We...

Find subarray with given sum · 1 hour ago

**newCoder3006** Code without using while loop. We can do it..

Find subarray with given sum · 1 hour ago

AdChoices 

[▶ Java Source Code](#)

[▶ Java Array](#)

[▶ Missing Numbers](#)

```

{
    int xor1;    /* Will hold xor of all elements and numbers from 1 to n */
    int set_bit_no; /* Will have only single set bit of xor1 */
    int i;
    *x = 0;
    *y = 0;

    xor1 = arr[0];

    /* Get the xor of all array elements */
    for(i = 1; i < n; i++)
        xor1 = xor1^arr[i];

    /* XOR the previous result with numbers from 1 to n*/
    for(i = 1; i <= n; i++)
        xor1 = xor1^i;

    /* Get the rightmost set bit in set_bit_no */
    set_bit_no = xor1 & ~(xor1-1);

    /* Now divide elements in two sets by comparing rightmost set
    bit of xor1 with bit at same position in each element. Also, get XO
    of two sets. The two XORs are the output elements.
    The following two for loops serve the purpose */
    for(i = 0; i < n; i++)
    {
        if(arr[i] & set_bit_no)
            *x = *x ^ arr[i]; /* arr[i] belongs to first set */
        else
            *y = *y ^ arr[i]; /* arr[i] belongs to second set*/
    }
    for(i = 1; i <= n; i++)
    {
        if(i & set_bit_no)
            *x = *x ^ i; /* i belongs to first set */
        else
            *y = *y ^ i; /* i belongs to second set*/
    }

    /* Now *x and *y hold the desired output elements */
}

/* Driver program to test above function */
int main()
{
    int arr[] = {1, 3, 4, 5, 5, 6, 2};
    int *x = (int *)malloc(sizeof(int));

```

```

int *y = (int *)malloc(sizeof(int));
int n = sizeof(arr)/sizeof(arr[0]);
getTwoElements(arr, n, x, y);
printf(" The two elements are %d and %d", *x, *y);
getchar();
}

```

Time Complexity:  $O(n)$

This method doesn't cause overflow, but it doesn't tell which one occurs twice and which one is missing. We can add one more step that checks which one is missing and which one is repeating. This can be easily done in  $O(n)$  time.

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

AdChoices 

[► The Missing](#)

[► Repeating](#)

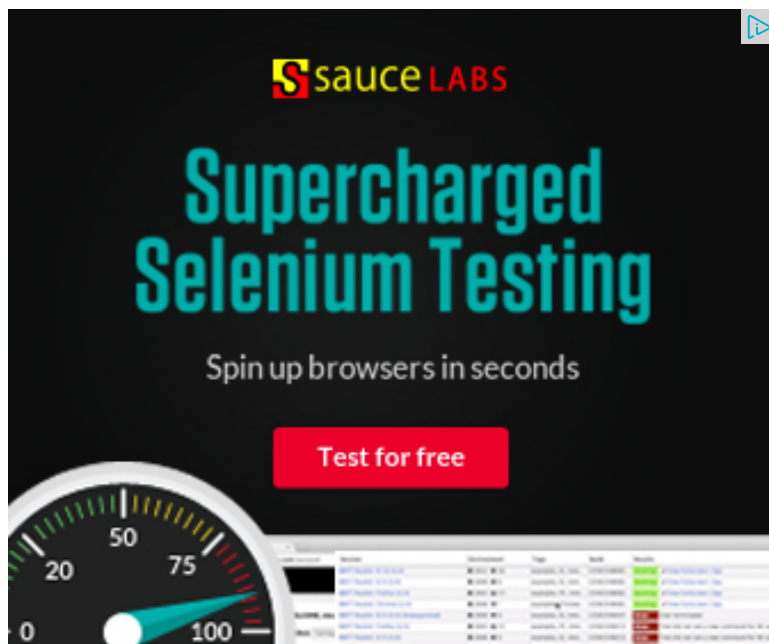
[► Memory Array](#)

AdChoices 

[► Missing Data](#)

[► Numbers Number](#)

[► Log Me Int](#)



Related Tpoics:

- Remove minimum elements from either side such that  $2 \times \min$  becomes more than max
- Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
- Bucket Sort
- Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1

- Find the number of zeroes
- Find if there is a subarray with 0 sum
- Divide and Conquer | Set 5 (Strassen's Matrix Multiplication)
- Count all possible groups of size 2 or 3 that have sum as multiple of 3



1

Tweet

0



1

Writing code in comment? Please use [ideone.com](https://ideone.com) and share the link here.

49 Comments

GeeksforGeeks

Sort by Newest ▼



Join the discussion...



**Guest** • 6 months ago

for i=0 to n-1  
arr[ (arr[i]-1)%n] + = n;

for i=0 to n-1  
if(arr[i]>2n)  
(i+1) is the repeated no  
if(arr[i] < n)  
(i+1) is the missing no

^ | v • Reply • Share ›



**Robert** • 7 months ago

let x1 = xor all the elements in array.

let x2 = xor elements 1 to n

let xr = x1 xor x2.

repeated element = xr xor x2.

missing element =  $xr \oplus x1$

1 ^ | v • Reply • Share ›



**wasseypuriyan** → Robert • 7 months ago

According to your formula

repeated elem =  $xr \oplus x2 = (x1 \oplus x2) \oplus x2 = x1$

which is xor of all elements in array not the repeated element.

On very similar lines I can say that your formula won't work for missing

^ | v • Reply • Share ›



**srinivas** • 8 months ago

```
#include<stdio.h>
```

```
int main()
```

```
{
```

```
int i,n,sum=0,actual_sum,twise;
```

```
int a[]={1,2,5,5,4};
```

```
n=sizeof(a)/sizeof(int);
```

```
actual_sum=(n*(n+1))/2;
```

```
for(i=0;i<n;i++) {="" a[(a[i]%(n+1)-1)]+="n+1;" sum+="(a[i]%(n+1));//finding" th  
1]="">(2*n+2))
```

```
twise=a[i]%(n+1);
```

```
}
```

```
printf("twise=%d,mis=%d",twise,twise+(actual_sum-sum));
```

```
return 0;
```

```
}
```

1 ^ | v • Reply • Share ›



**Ravish** • 9 months ago

I think there is no need for the last for loop in 5th method.\*x and \*y already hav

/\* Paste your code here (You may **delete** these lines **if not** writing c

^ | v • Reply • Share ›



**Ganesh** • a year ago

You can find java code here:

```
[sourcecode language="JAVA"]
```

```
/**
```

```
* Given an unsorted array of size n. Array elements are in range from 1 to n.
```

```
* One number from set {1, 2, ...n} is missing and one number occurs twice in
```

```
* Example: arr[] = {3, 1, 3}
```

```
* Output: 2, 3 // 2 is missing and 3 occurs twice
```

```
*
```

```
* @author GAPIITD
```

```
*
```

```
*/
```

```
public class FindTheRepeatingAndTheMissing {
```

```
    public static void main(String[] args) {
```

```
        int arr[] = {1, 3, 4, 5, 5, 6, 2};
```

```
        FindRepeatingAndMissing(arr);
```

```
    }
```

---

[see more](#)

1 ^ | v • Reply • Share ›



**ramdas** • a year ago

hey geeks here is a problem statment:-

search a particular no. in a given array ( return with how many times and with the no. inside array also any.....

^ | v • Reply • Share ›





nit\_d → ramdas · a year ago

Are elements sorted?

```
/* Paste your code here (You may delete these lines if not wri
```

^ | v · Reply · Share ›



Vikky · 2 years ago

Code for the logic given by me :

<http://www.geeksforgeeks.org/a...>

```
#include<stdio.h>

/*
This program assumes that array elements are given according to prob:
* ie. from 1..n
*/

int main(){

    int arr[]={1,2,3,4,5,3};
    int len=sizeof(arr)/sizeof(arr[0]);

    int sum1=0,sum3=0;
    int sum=((len+1)*len)/2;
    for(int i=0;i<len;i++)
```

see more

^ | v · Reply · Share ›



cracker · 2 years ago

How could method 5 be extended if there are more than one repeating and mi:  
post your ideas on this

post your code on them.

^ | v • Reply • Share ›



**BlackMath** • 2 years ago

This is a java code and is inspired by a similar post, but there the method did r  
This is a nice solution with Time complexity  $O(n)$ , but we traverse the array thi

```
/* Paste your code here (You may delete these lines if not writing c  
public class FindRepeatingAndMissing  
{  
    public static void main (String args[])  
    {  
        int arr[] = new int[] {4,2,1,6,5,6};  
        int size = 6;  
  
        for (int i = 0; i < size; i++)  
            if (arr[i] != i+1)  
            {  
                int tmp = arr[arr[i]-1];  
                arr[arr[i]-1] = arr[i];  
                arr[i] = tmp;  
            }  
    }  
}
```

see more

1 ^ | v • Reply • Share ›



**Nages** • 2 years ago

```
public class ArrayDuplicate {  
  
    public static void main(String[] args) {  
  
        int a[] = new int[] { 3, 5, 1, 6, 4, 7, 5 };  
    }  
}
```

```

int len = a.length;

int sum = 0;
int sum2 = 0;

for (int i = 0; i < len; ++i) {
    sum += a[i];
    sum2 += a[i] * a[i];
}

int asum = (len * (len + 1)) / 2;
int asum2 = (len * (len + 1) * (2 * len + 1)) / 6;

```

[see more](#)

^ | v • Reply • Share ›



**PsychoCoder** • 2 years ago

In method 5 : we can detect two desired results. But can we find which one is mean with traversing the array for another time to detect which one is missing

^ | v • Reply • Share ›



**GeeksforGeeks** → PsychoCoder • 2 years ago

Yes, we can add a postprocessing step to figure out which one is miss have added a note for the same.

^ | v • Reply • Share ›



**Venkatesh** • 2 years ago

correct me if i am wrong. seems like method 3 overruns array.

for ex: my input is 5 4 3 2.  $\text{arr}[\text{abs}(\text{arr}[i]) - 1] = \text{arr}[4]$  which is overrun. isn't it?

for(i = 0; i 0)

```
arr[abs(arr[i])-1] = -arr[abs(arr[i])-1];  
else  
printf(" %d ", abs(arr[i]));  
}
```

^ | v • Reply • Share ›



**Venkatesh** • 2 years ago

<http://www.c-sharpcorner.com/B...>

^ | v • Reply • Share ›



**Venkatesh** • 2 years ago

what does following means

```
#include&lt;stdio.h&gt;  
#include&lt;stdlib.h&gt;
```

^ | v • Reply • Share ›



**GeeksforGeeks** → Venkatesh • 2 years ago

This has been Fixed. Please check now.

^ | v • Reply • Share ›



**GeekyPortal.com** • 3 years ago

I think 2nd one is the easiest among all.

^ | v • Reply • Share ›



**Nitin Kumar** → GeekyPortal.com • 3 years ago

But in case n is too large... then creating an array of that size is not po:

and How come the complexity of method 5 is just  $O(n)$  ??

^ | v • Reply • Share ›



**kartik** → Nitin Kumar • 3 years ago

Time complexity of Method 5 is  $O(n)$  only. There are no nested

Time complexity of method 3 is  $O(n)$  only. There are no nested loops than  $O(n)$ ?

^ | v • Reply • Share ›



**Nitin Kumar** → kartik • 3 years ago

Ok I got it..  $O(n)$ !!

^ | v • Reply • Share ›



**Satyarth** • 3 years ago

I have another approach to solve this problem.

```
#include<stdio.h>
#include<stdlib.h>

void swap(int *x, int *y)
{
    int t;
    t = *x;
    *x = *y;
    *y = t;
}

void find_missing(int A[],int n){
    int i;
    for(i=1;i<=n;++i){
        while(A[i]!=i && A[i]!=A[A[i]]){
            swap(&A[i],&A[A[i]]);
        }
    }
}
```

see more

^ | v • Reply • Share ›



**Mritiunjay** • 3 years ago

Easiest and fastest way using sum and sum of squares formula



Easiest and fastest way using sum and sum of squares formula..

Assume the array is 1 to n and sum of the numbers in shuffled array(with one S.

Let a be the repeated number and b be the missing number.

so,

$$s(\text{actual sum of numbers}) = n*(n+1)/2 + a - b \dots\dots(1)$$

$$s'(\text{actual sum of squares of numbers}) = \{n*(n+1)*(2n+1)/6\} + a^2 - b^2 \dots\dots(2)$$

by (2)/(1) get value of a+b ...(3)

Now, solve for a and b from equation (1) and (3)

^ | v • Reply • Share ›



**CaesiumX** • 3 years ago

how about this...

```
void tofind(int a[],int size)
{
    int i,j,cnt;
    for(i=1;i<=size;i++)
    {
        for(cnt=0,j=0;j<size;j++)
        {
            if(i==a[j])
                cnt++;
        }
        if(cnt>1)
            printf("Repeating:%d\n",i);
        else if(cnt==0)
            printf("Missing:%d\n",i);
    }
}
```

}

not sure if this is covered in the methods posted.  
Kindly let me know if something fails or is missing

Regards

^ | v • Reply • Share ›



**Faisy** • 3 years ago

I think Method 5 is the easiest.

^ | v • Reply • Share ›



**Sandeep** • 3 years ago

@Manish Mishram, @disappearedng, @john and @LoneShadow:

Thanks for your contribution. We have added the suggested methods to the or

^ | v • Reply • Share ›



**LoneShadow** • 3 years ago

Another obvious solution I didnt see, but nlogn. Sort and walk the array.

^ | v • Reply • Share ›



**asd** • 3 years ago

This gives the rightmost set bit

$\text{set\_no\_bits} = (\text{xor1} \& -\text{xor1}) + 1$

The one given in the code doesn't work...

^ | v • Reply • Share ›



**Sandeep** → asd • 3 years ago



@asd: could you please provide an example for which the logic given i

^ | v • Reply • Share ›



raman • 3 years ago

We can find the repeating element in  $O(n)$  by negating each element method.

then compute the sum of given array and compare with  $n*(n+1)$  find the difference the  $\text{abs}(\text{difference})$  will be the missing element.

^ | v • Reply • Share ›



Manish Mishra • 3 years ago

how about this..

```
void printRepeating(int arr[], int size)
{
    int i;

    printf("\n The repeating element is");

    for(i = 0; i < size; i++)
    {
        if(arr[abs(arr[i])-1] > 0)
            arr[abs(arr[i])-1] = -arr[abs(arr[i])-1];
        else
            printf(" %d ", abs(arr[i]));
    }
    printf("\nand the missing element is ");
    for(i=0;i<size;i++)
    {
        if(arr[i]>0)
            printf("%d", i+1);
    }
}
```



```
}
```

This is same as the logic used in two of the previous questions..  
Correct me if the above code fails in any of the test cases..

^ | v • Reply • Share ›



**Sandeep** → Manish Mishra • 3 years ago

@Manish Mishra: Thanks for suggesting a new method. The method s  
and test it further and add to the original post.

^ | v • Reply • Share ›



**Gaurav** • 3 years ago

```
/* Get the rightmost set bit in set_bit_no */  
set_bit_no = xor1 & ~(xor1-1);
```

Please explain this line.

^ | v • Reply • Share ›



**john** • 3 years ago

disappearedng's method can be improved to reduce the chance of overflow by  
numbers with  $1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$ .

So we have

A.  $\frac{n(n+1)}{2} - \text{duplicate} + \text{missing} = \text{sum\_of\_the\_array}$

B.  $\frac{n(n+1)(2n+1)}{6} - \text{duplicate}^2 + \text{missing}^2 = \text{sum\_of\_array\_square}$

We can solve the system to find the duplicate and missing in  $O(n)$  time and O  
be reduced further in coding. Below is the Java implementation:

```
public void findMissionAndDuplicate(int[] arr) {  
    long sum1 = 0;  
    long sum2 = 0;  
    for (int i = 0; i < arr.length; i++) {  
        sum1 += i + 1 - arr[i];  
        sum2 += (i + 1) * (i + 1) - arr[i] * arr[i];  
    }  
}
```

```

        sum2 += (i + 1 - arr[i]) * (i + 1 + arr[i]);
    }
    long d_minus_m = sum1;
    long d_plus_m = sum2 / d_minus_m;
    long m = (d_plus_m + d_minus_m) / 2;

```

[see more](#)

3 ^ | v • Reply • Share ›



**atul** → john • 2 years ago

how did you solve these equations using these technique..i didnt get th

```

/* Paste your code here (You may delete these lines if not wr

```

^ | v • Reply • Share ›



**disappearedng** • 3 years ago

Actually,

I think rather than writing code, let's step back and do some math.

Given 1...n, 1 missing and 1 repeated,  
let x be missing and y be repeated.

We have:

$$n(n+1)/2 - x + y = \{ \text{sum of array computed} \}$$

$$1*2*3*...*n * y / x = \{ \text{multiplication of all elements} \}$$

Then you have two equations and you can solve.

You can code this up, but coding a solution for this might be a little tricky.

time: O(n)

space:  $O(1)$

This is easier to understand

^ | v • Reply • Share ›



**GeeksforGeeks** → disappearedng • 3 years ago

@disappearedng: Thanks for suggesting a new method. This is another and  $O(1)$  extra space.

Following are the points to note when this method is compared with the

(a) This method also tells which one is repeating and which one is missing info.

(b) This method can cause arithmetic overflow which is not possible in

We will analyze this further and add to the original post.

^ | v • Reply • Share ›



**Vikky** → GeeksforGeeks • 2 years ago

Hi,

Rather than multiplying the elements we can use sum of squares follows:

Lets say missing number is  $x$  and repeating number is  $y$ .

1) First get the sum of all the numbers, let's say it sum and get sum1, now subtract from (sum1-sum) we will get  $(x-y)$  - equation

2) Now using sum of squares get another result  $n(n+1)(2n+1)$ , the numbers and add it let's say it sum3, now getting (sum2-sum

Now from equation 1 and 2 we can easily get  $x$  and  $y$  as equations using value from equation 1 we get value of  $(x+y)$ , now using the again get the value of  $x$  and similarly value for  $y$ .

Hope this helps.!

^ | v • Reply • Share ›



**Eatesh** → GeeksforGeeks • 3 years ago

i guess time complexity of the suggested code is  $O(1)$  as we n  
Correct me if i am wrong.

^ | v • Reply • Share ›



**GeeksforGeeks** → Eatesh • 3 years ago

@Eatesh:  $O(n)$  is the correct time complexity. We need  
numbers and multiply all numbers.

^ | v • Reply • Share ›



**Eatesh** → GeeksforGeeks • 3 years ago

Thanks, interpreted it wrongly.

^ | v • Reply • Share ›



**aman** • 3 years ago

i did not understand the logic of XOR method. can anyone please explain?

^ | v • Reply • Share ›



**Sandeep** → aman • 3 years ago

@aman:

The approach used here is similar to the method 4 of [this](#) post and me

^ | v • Reply • Share ›



**Uttam** • 3 years ago

The logic given for XOR will not work for the set  $\{1,2,3\}$   
and input array 3,1,3.. it will give output 2 and 2.

But Actual output is 2 and 3.

^ | v • Reply • Share ›



**GeeksforGeeks** → Uttam · 3 years ago

The code works fine for {3, 1, 3} and prints 2 and 3. Could you please p

^ | v · Reply · Share ›



**diver99** → GeeksforGeeks · 3 years ago

"So if we take any set bit (We have chosen the rightmost set bit) and partition the elements of the array in two sets – one set of elements with that bit set and one set of elements with that bit not set. By doing so, we will get x in one set and y in another set.

Let's say 1 was missing and 3 was repeated.

Input = {2,3,3}

When you partition the elements wouldn't 1 and 3 end up in the same set?

^ | v · Reply · Share ›



**Sandeep** → diver99 · 3 years ago

@diver99: Please take a closer look. When we do XOR from 1 to n, we get XOR of the two required numbers. For example, if the missing number is 1 and the repeated number is 3, the XOR will be 2 (0010). Now we take a set bit from this XOR (the rightmost set bit, which is 2) and partition the elements into two sets: one with that bit set and one without. This ensures the missing number and the repeated number end up in different sets.

^ | v · Reply · Share ›



**slimshady** · 3 years ago

this is cool..thanks for posting

^ | v · Reply · Share ›



Subscribe



Add Disqus to your site

@geeksforgeeks, **Some rights reserved**

**Contact Us!**

Powered by **WordPress** & **MooTools**, customized by geeksforgeeks team