Find the element that appears once

Given an array where every element occurs three times, except one element which occurs only once. Find the element that occurs once. Expected time complexity is O(n) and O(1) extra space.

Examples :

```
Input: arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 3}
Output: 2
```

Recommended: Please solve it on “*PRACTICE* ” first, before moving on to the solution.

We can use sorting to do it in O(nLogn) time. We can also use hashing, it has the worst case time complexity of O(n), but requires extra space.

The idea is to use bitwise operators for a solution that is O(n) time and uses O(1) extra space. The solution is not easy like other XOR based solutions, because all elements appear odd number of times here. The idea is taken from here.

Run a loop for all elements in array. At the end of every iteration, maintain following two values.

ones: The bits that have appeared 1st time or 4th time or 7th time .. etc.

twos: The bits that have appeared 2nd time or 5th time or 8th time .. etc.

Finally, we return the value of ‘ones’

How to maintain the values of ‘ones’ and ‘twos’?
‘ones’ and ‘twos’ are initialized as 0. For every new element in array, find out the common set bits in the new element and previous value of ‘ones’. These common set bits are actually the bits that should be added to ‘twos’. So do bitwise OR of the common set bits with ‘twos’. ‘twos’ also gets some extra bits that appear third time. These extra bits are removed later.
Update ‘ones’ by doing XOR of new element with previous value of ‘ones’. There may be some bits which appear 3rd time. These extra bits are also removed later.

Both ‘ones’ and ‘twos’ contain those extra bits which appear 3rd time. Remove these extra bits by finding out common set bits in ‘ones’ and ‘twos’.

C/C++

```
#include <stdio.h>

int getSingle(int arr[], int n)
{
    int ones = 0, twos = 0 ;

    int common_bit_mask;

    // Let us take the example of {3, 3, 2, 3} to understand this
    for( int i=0; i< n; i++ )
    {
        /* The expression "one & arr[i]" gives the bits that are
           there in both 'ones' and new element from arr[].  We
           add these bits to 'twos' using bitwise OR

           Value of 'twos' will be set as 0, 3, 3 and 1 after 1st,
           2nd, 3rd and 4th iterations respectively */
        twos  = twos | (ones & arr[i]);


        /* XOR the new bits with previous 'ones' to get all bits
           appearing odd number of times

           Value of 'ones' will be set as 3, 0, 2 and 3 after 1st,
           2nd, 3rd and 4th iterations respectively */
        ones  = ones ^ arr[i];


        /* The common bits are those bits which appear third time
           So these bits should not be there in both 'ones' and 'twos'.
           common_bit_mask contains all these bits as 0, so that the bits can
           be removed from 'ones' and 'twos'

           Value of 'common_bit_mask' will be set as 00, 00, 01 and 10
           after 1st, 2nd, 3rd and 4th iterations respectively */
        common_bit_mask = ~(ones & twos);


        /* Remove common bits (the bits that appear third time) from 'ones'

           Value of 'ones' will be set as 3, 0, 0 and 2 after 1st,
           2nd, 3rd and 4th iterations respectively */
        ones &= common_bit_mask;


        /* Remove common bits (the bits that appear third time) from 'twos'

           Value of 'twos' will be set as 0, 3, 1 and 0 after 1st,
           2nd, 3rd and 4th itearations respectively */
        twos &= common_bit_mask;

        // uncomment this code to see intermediate values
        //printf (" %d %d n", ones, twos);
    }

    return ones;
}

int main()
{
    int arr[] = {3, 3, 2, 3};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("The element with single occurrence is %d ",
            getSingle(arr, n));
    return 0;
}
```

Run on IDE

Java

```
//JAVA code to find the element
//that occur only once

class GFG
{
    // Method to find the element that occur only once
    static int getSingle(int arr[], int n)
    {
        int ones = 0, twos = 0;
        int common_bit_mask;

        for(int i=0; i<n; i++ )
        {
            /*"one & arr[i]" gives the bits that are there in
            both 'ones' and new element from arr[]. We
            add these bits to 'twos' using bitwise OR*/
            twos = twos | (ones & arr[i]);

            /*"one & arr[i]" gives the bits that are
            there in both 'ones' and new element from arr[].
            We add these bits to 'twos' using bitwise OR*/
            ones = ones ^ arr[i];

            /* The common bits are those bits which appear third time
            So these bits should not be there in both 'ones' and 'twos'.
            common_bit_mask contains all these bits as 0, so that the bits can
            be removed from 'ones' and 'twos'*/
            common_bit_mask = ~(ones & twos);

            /*Remove common bits (the bits that appear third time) from 'ones'*/
            ones &= common_bit_mask;

            /*Remove common bits (the bits that appear third time) from 'twos'*/
            twos &= common_bit_mask;
        }
        return ones;
    }

    //  Driver method
    public static void main(String args[])
    {
        int arr[] = {3, 3, 2, 3};
        int n = arr.length;
        System.out.println("The element with single occurrence is " + getSingle(arr, n));
    }
}
// Code contributed by Rishab Jain
```

Run on IDE

Python3

```
# Python3 code to find the element that
# appears once

def getSingle(arr, n):
    ones = 0
    twos = 0
    for i in range(n):
        # one & arr[i]" gives the bits that
        # are there in both 'ones' and new
        # element from arr[]. We add these
        # bits to 'twos' using bitwise OR
        twos = twos | (ones & arr[i])

        # one & arr[i]" gives the bits that
        # are there in both 'ones' and new
        # element from arr[]. We add these
        # bits to 'twos' using bitwise OR
        ones = ones ^ arr[i]

        # The common bits are those bits
        # which appear third time. So these
        # bits should not be there in both
        # 'ones' and 'twos'. common_bit_mask
        # contains all these bits as 0, so
        # that the bits can be removed from
        # 'ones' and 'twos'
        common_bit_mask = ~(ones & twos)

        # Remove common bits (the bits that
        # appear third time) from 'ones'
        ones &= common_bit_mask

        # Remove common bits (the bits that
        # appear third time) from 'twos'
        twos &= common_bit_mask
    return ones

# driver code
arr = [3, 3, 2, 3]
n = len(arr)
print("The element with single occurrence is ",
        getSingle(arr, n))

# This code is contributed by "Abhishek Sharma 44"
```

Run on IDE

C#

```
// C# code to find the element
// that occur only once
using System;
class GFG
{
    // Method to find the element
    // that occur only once
    static int getSingle(int []arr, int n)
    {
        int ones = 0, twos = 0;
        int common_bit_mask;

        for(int i = 0; i < n; i++ )
        {
            // "one & arr[i]" gives the bits
            // that are there in both 'ones'
            // and new element from arr[].
            // We add these bits to 'twos'
            // using bitwise OR
            twos = twos | (ones & arr[i]);

            // "one & arr[i]" gives the bits
            // that are there in both 'ones'
            // and new element from arr[].
            // We add these bits to 'twos'
            // using bitwise OR
            ones = ones ^ arr[i];
            // The common bits are those bits
            // which appear third time So these
            // bits should not be there in both
            // 'ones' and 'twos'. common_bit_mask
            // contains all these bits as 0,
            // so that the bits can be removed
            // from 'ones' and 'twos'
            common_bit_mask = ~(ones & twos);

            // Remove common bits (the bits that
            // appear third time) from 'ones'
            ones &= common_bit_mask;

            // Remove common bits (the bits that
            // appear third time) from 'twos'
            twos &= common_bit_mask;
        }
        return ones;
    }

    // Driver code
    public static void Main()
    {
        int []arr = {3, 3, 2, 3};
        int n = arr.Length;
        Console.WriteLine("The element with single" +
            "occurrence is " + getSingle(arr, n));
    }
}

// This Code is contributed by vt_m.
```

Run on IDE

PHP

```
<?php

function getSingle($arr, $n)
{
    $ones = 0; $twos = 0 ;

    $common_bit_mask;

    // Let us take the example of
    // {3, 3, 2, 3} to understand this
    for($i = 0; $i < $n; $i++ )
    {
        /* The expression "one & arr[i]"
        gives the bits that are there in
        both 'ones' and new element from
        arr[]. We add these bits to 'twos'
        using bitwise OR
        Value of 'twos' will be set as 0,
        3, 3 and 1 after 1st, 2nd, 3rd
        and 4th iterations respectively */
        $twos = $twos | ($ones & $arr[$i]);


        /* XOR the new bits with previous
        'ones' to get all bits appearing
        odd number of times

        Value of 'ones' will be set as 3,
        0, 2 and 3 after 1st, 2nd, 3rd and
        4th iterations respectively */
        $ones = $ones ^ $arr[$i];

        /* The common bits are those bits
        which appear third time. So these
        bits should not be there in both
        'ones' and 'twos'. common_bit_mask
        contains all these bits as 0, so
        that the bits can be removed from
        'ones' and 'twos'

        Value of 'common_bit_mask' will be
        set as 00, 00, 01 and 10 after 1st,
        2nd, 3rd and 4th iterations respectively */
        $common_bit_mask = ~($ones & $twos);


        /* Remove common bits (the bits
        that appear third time) from 'ones'

        Value of 'ones' will be set as 3,
        0, 0 and 2 after 1st, 2nd, 3rd
        and 4th iterations respectively */
        $ones &= $common_bit_mask;


        /* Remove common bits (the bits
        that appear third time) from 'twos'

        Value of 'twos' will be set as 0, 3,
        1 and 0 after 1st, 2nd, 3rd and 4th
        itearations respectively */
        $twos &= $common_bit_mask;

        // uncomment this code to see
        // intermediate values
        // printf (" %d %d n", ones, twos);
    }

    return $ones;
}

// Driver Code
$arr = array(3, 3, 2, 3);
$n = sizeof($arr);
echo "The element with single " .
                "occurrence is ",
             getSingle($arr, $n);

// This code is contributed by m_kit
?>
```

Run on IDE

Output :

```
The element with single occurrence is 2
```

Time Complexity : O(n)

Auxiliary Space : O(1)

Following is another O(n) time complexity and O(1) extra space method suggested by *aj*. We can sum the bits in same positions for all the numbers and take modulo with 3. The bits for which sum is not multiple of 3, are the bits of number with single occurrence.

Let us consider the example array {5, 5, 5, 8}. The 101, 101, 101, 1000

Sum of first bits%3 = (1 + 1 + 1 + 0)%3 = 0;

Sum of second bits%3 = (0 + 0 + 0 + 0)%0 = 0;

Sum of third bits%3 = (1 + 1 + 1 + 0)%3 = 0;

Sum of fourth bits%3 = (1)%3 = 1;

Hence number which appears once is 1000

C/C++

```
#include <stdio.h>
#define INT_SIZE 32

int getSingle(int arr[], int n)
{
    // Initialize result
    int result = 0;

    int x, sum;

    // Iterate through every bit
    for (int i = 0; i < INT_SIZE; i++)
    {
      // Find sum of set bits at ith position in all
      // array elements
      sum = 0;
      x = (1 << i);
      for (int j=0; j< n; j++ )
      {
          if (arr[j] & x)
            sum++;
      }

      // The bits with sum not multiple of 3, are the
      // bits of element with single occurrence.
      if (sum % 3)
        result |= x;
    }

    return result;
}

// Driver program to test above function
int main()
{
    int arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("The element with single occurrence is %d ",
            getSingle(arr, n));
    return 0;
}
```

Run on IDE

Java

```
// JAVA code to find the element
// that occur only once

class GFG
{
    static final int INT_SIZE = 32;

    // Method to find the element that occur only once
    static int getSingle(int arr[], int n)
    {
        int result = 0;
        int x, sum;

        // Iterate through every bit
        for(int i=0; i<INT_SIZE; i++)
        {
            // Find sum of set bits at ith position in all
            // array elements
            sum = 0;
            x = (1 << i);
            for(int j=0; j<n; j++)
            {
                if((arr[j] & x) == 0)
                sum++;
            }
            // The bits with sum not multiple of 3, are the
            // bits of element with single occurrence.
            if ((sum % 3) == 0)
            result |= x;
        }
        return result;
    }

    // Driver method
    public static void main(String args[])
    {
        int arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7};
        int n = arr.length;
        System.out.println("The element with single occurrence is " + getSingle(arr, n));
    }

}
// Code contributed by Rishab Jain
```

Run on IDE

Python 3

```
# Python3 code to find the element
# that occur only once
INT_SIZE = 32

def getSingle(arr, n) :

    # Initialize result
    result = 0

    # Iterate through every bit
    for i in range(0, INT_SIZE) :

        # Find sum of set bits
        # at ith position in all
        # array elements
        sm = 0
        x = (1 << i)
        for j in range(0, n) :
            if (arr[j] & x) :
                sm = sm + 1

        # The bits with sum not
        # multiple of 3, are the
        # bits of element with
        # single occurrence.
        if (sm % 3) :
            result = result | x

    return result

# Driver program
arr = [12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7]
n = len(arr)
print("The element with single occurrence is "
      ,getSingle(arr, n))
```

```
# This code is contributed
# by Nikita Tiwari.
```

Run on IDE

C#

```
// C# code to find the element
// that occur only once
using System;

class GFG
{
    static int INT_SIZE = 32;

    // Method to find the element
    // that occur only once
    static int getSingle(int []arr, int n)
    {
        int result = 0;
        int x, sum;

        // Iterate through every bit
        for(int i = 0; i < INT_SIZE; i++)
        {
            // Find sum of set bits at ith
            // position in all array elements
            sum = 0;
            x = (1 << i);
            for(int j = 0; j < n; j++)
            {
                if((arr[j] & x) == 0)
                sum++;
            }

            // The bits with sum not multiple
            // of 3, are the bits of element
            // with single occurrence.
            if ((sum % 3) == 0)
            result |= x;
        }
        return result;
    }

    // Driver Code
    public static void Main()
    {
        int []arr = {12, 1, 12, 3, 12, 1,
                     1, 2, 3, 2, 2, 3, 7};
        int n = arr.Length;
        Console.WriteLine("The element with single " +
                "occurrence is " + getSingle(arr, n));
    }

}

// This code is contributed ny vt_m.
```

Run on IDE

PHP

```
<?php
// PHP code to find the element
// that occur only once
$INT_SIZE= 32;

function getSingle($arr, $n)
```

```
{
    global $INT_SIZE;

    // Initialize result
    $result = 0;

    $x; $sum;

    // Iterate through every bit
    for ($i = 0; $i < $INT_SIZE; $i++)
    {
    // Find sum of set bits at ith
    // position in all array elements
    $sum = 0;
    $x = (1 << $i);
    for ($j = 0; $j < $n; $j++ )
    {
        if ($arr[$j] & $x)
            $sum++;
    }

    // The bits with sum not multiple
    // of 3, are the bits of element
    // with single occurrence.
    if ($sum % 3)
        $result |= $x;
    }

    return $result;
}

// Driver Code
$arr = array (12, 1, 12, 3, 12, 1,
              1, 2, 3, 2, 2, 3, 7);
$n = sizeof($arr);
echo "The element with single occurrence is ",
                          getSingle($arr, $n);

// This code is contributed by ajit
?>
```

Run on IDE

Output :

```
The element with single occurrence is 7
```

Another approach suggested by *Abhishek Sharma 44*. Add each number once and multiply the sum by 3, we will get thrice the sum of each element of the array. Store it as thrice_sum. Subtract the sum of the whole array from the thrice_sum and divide the result by 2. The number we get is the required number (which appears once in the array).

Array [] : [a, a, a, b, b, b, c, c, c, d]

Mathematical Equation = (3*(a+b+c+d) – (a + a + a + b + b + b + c + c + c + d)) / 2

In more simple words: (3*(sum_of_array_without_duplicates) – (sum_of_array)) / 2

```
let arr[] = {12, 1, 12, 3, 12, 1, 1, 2, 3, 3}
Required no = ( 3*(sum_of_array_without_duplicates) - (sum_of_array) ) / 2
            = ( 3*(12 + 1 + 3 + 2) - (12 + 1 + 12 + 3 + 12 + 1 + 1 + 2 + 3 + 3))/2
            = ( 3*     18          -      50) / 2
```

```
              = (54 - 50) / 2
              = 2 (required answer)
```

As we know that **set** does not contain any duplicate element, we will be using set here.

Below is the implementation of above approach:

Python3

```python
# Python program to find the element
# that occur only once

# function which find number
def singleNumber(nums):

    # applying the formula.
    return (3 * sum(set(nums)) - sum(nums)) / 2

# driver function.
a = [12, 1, 12, 3, 12, 1, 1, 2, 3, 2, 2, 3, 7]
print ("The element with single occurrence is ",
                          int(singleNumber(a)))
```

Run on IDE

Output :

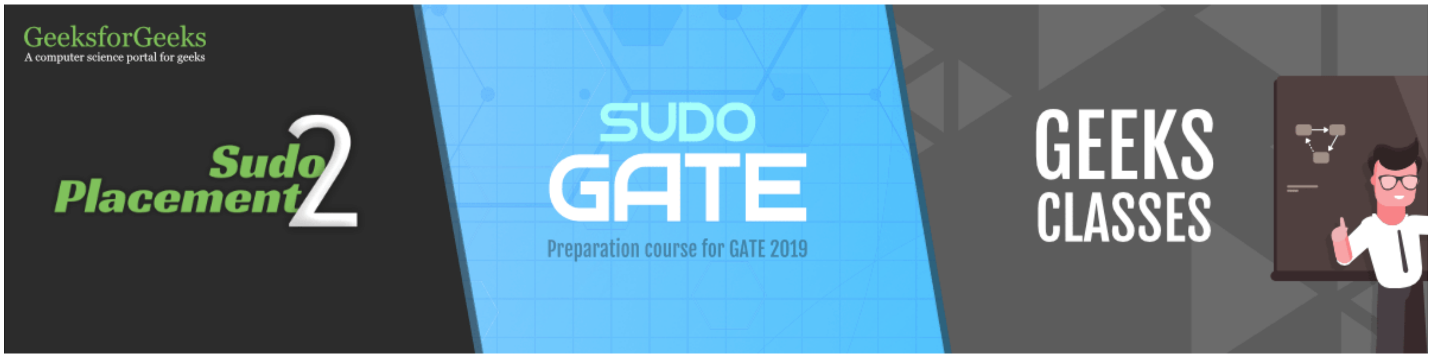
```
The element with single occurrence is 7
```

This article is compiled by **Sumit Jain** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Improved By : jit_t

Practice Tags : Microsoft Amazon Ola Cabs Arrays Mathematical Bit Magic

Article Tags : Arrays Bit Magic Mathematical Amazon Microsoft Ola Cabs XOR


GeeksforGeeks
A computer science portal for geeks
Sudo Placement 2
SUDO GATE
Preparation course for GATE 2019
GEEKS CLASSES

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content. Login to Improve this Article

Recommended Posts:

Detect if two integers have opposite signs
Binary representation of a given number
Count total set bits in all numbers from 1 to n
Find the two non-repeating elements in an array of repeating elements
Add 1 to a given number
Convert a String to Integer Array in C/C++
Minimum number of items to be delivered
Check if subarray with given product exists in an array
Replace every element of the array by sum of all other elements
Calculate the total fine to be collected

(Login to Rate)

4.2 Average Difficulty : **4.2/5.0**
Based on **267** vote(s)

Add to TODO List
Mark as DONE

Feedback Add Notes Improve Article

Basic Easy Medium Hard Expert

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments Share this post!