Find the two numbers with odd occurrences in an unsorted array

Given an unsorted array that contains even number of occurrences for all numbers except two numbers. Find the two numbers which have odd occurrences in O(n) time complexity and O(1) extra space.

Examples:

```
Input: {12, 23, 34, 12, 12, 23, 12, 45}
Output: 34 and 45

Input: {4, 4, 100, 5000, 4, 4, 4, 4, 100, 100}
Output: 100 and 5000

Input: {10, 20}
Output: 10 and 20
```

A **naive method** to solve this problem is to run two nested loops. The outer loop picks an element and the inner loop counts the number of occurrences of the picked element. If the count of occurrences is odd then print the number. The time complexity of this method is O(n^2).

We can **use sorting** to get the odd occurring numbers in O(nLogn) time. First sort the numbers using an O(nLogn) sorting algorithm like Merge Sort, Heap Sort.. etc. Once the array is sorted, all we need to do is a linear scan of the array and print the odd occurring number.

We can also **use hashing**. Create an empty hash table which will have elements and their counts. Pick all elements of input array one by one. Look for the picked element in hash table. If the element is found in hash table, increment its count in table. If the element is not found, then enter it in hash table with count as 1. After all elements are entered in hash table, scan the hash table and print elements with odd count. This approach may take O(n) time on average, but it requires O(n) extra space.

A O(n) time and O(1) extra space solution:
The idea is similar to method 2 of this post. Let the two odd occurring numbers be x and y. We **use bitwise**

XOR to get x and y. The first step is to do XOR of all elements present in array. XOR of all elements gives us XOR of x and y because of the following properties of XOR operation.

1) XOR of any number n with itself gives us 0, i.e., n ^ n = 0
2) XOR of any number n with 0 gives us n, i.e., n ^ 0 = n
3) XOR is cumulative and associative.

So we have XOR of x and y after the first step. Let the value of XOR be xor2. Every set bit in xor2 indicates that the corresponding bits in x and y have values different from each other. For example, if x = 6 (0110) and y is 15 (1111), then xor2 will be (1001), the two set bits in xor2 indicate that the corresponding bits in x and y are different. In the second step, we pick a set bit of xor2 and divide array elements in two groups. Both x and y will go to different groups. In the following code, the rightmost set bit of xor2 is picked as it is easy to get rightmost set bit of a number. If we do XOR of all those elements of array which have the corresponding bit set (or 1), then we get the first odd number. And if we do XOR of all those elements which have the corresponding bit 0, then we get the other odd occurring number. This step works because of the same properties of XOR. All the occurrences of a number will go in same set. XOR of all occurrences of a number which occur even number number of times will result in 0 in its set. And the xor of a set will be one of the odd occurring elements.

```c
// Program to find the two odd occurring elements
#include<stdio.h>

/* Prints two numbers that occur odd number of times. The
   function assumes that the array size is at least 2 and
   there are exactly two numbers occurring odd number of times. */
void printTwoOdd(int arr[], int size)
{
  int xor2 = arr[0]; /* Will hold XOR of two odd occurring elements */
  int set_bit_no;  /* Will have only single set bit of xor2 */
  int i;
  int n = size - 2;
  int x = 0, y = 0;

  /* Get the xor of all elements in arr[]. The xor will basically
     be xor of two odd occurring elements */
  for(i = 1; i < size; i++)
    xor2 = xor2 ^ arr[i];

  /* Get one set bit in the xor2. We get rightmost set bit
     in the following line as it is easy to get */
  set_bit_no = xor2 & ~(xor2-1);

  /* Now divide elements in two sets:
    1) The elements having the corresponding bit as 1.
    2) The elements having the corresponding bit as 0.  */
  for(i = 0; i < size; i++)
  {
     /* XOR of first set is finally going to hold one odd
       occurring number x */
    if(arr[i] & set_bit_no)
      x = x ^ arr[i];

     /* XOR of second set is finally going to hold the other
       odd occurring number y */
    else
      y = y ^ arr[i];
  }
  printf("\n The two ODD elements are %d & %d ", x, y);
}

/* Driver program to test above function */
int main()
{
  int arr[] = {4, 2, 4, 5, 2, 3, 3, 1};
  int arr_size = sizeof(arr)/sizeof(arr[0]);
  printTwoOdd(arr, arr_size);
  getchar();
  return 0;
}
```

Run on IDE

Output:

```
The two ODD elements are 5 & 1
```

Time Complexity: O(n)

Auxiliary Space: O(1)

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

59 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value

- Find zeroes to be flipped so that number of consecutive 1’s is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

3.5 Average Difficulty : **3.5/5.0**
Based on **6** vote(s)

☐ Add to TODO List
☐ Mark as DONE

Like Share 3 people like this. Be the first of your friends.

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