Single Number

Given an array of integers, every element appears *twice* except for one. Find that single one.

Note:
Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

Solution 1

known that A XOR A = 0 and the XOR operator is commutative, the solution will be very straightforward. `

```
int singleNumber(int A[], int n) {
    int result = 0;
    for (int i = 0; i<n; i++)
    {
        result ^=A[i];
    }
    return result;
}
```

`

written by Ivantsang original link here

Solution 2

Logic: XOR will return 1 only on two different bits. So if two numbers are the same, XOR will return 0. Finally only one number left. A ^ A = 0 and A ^ B ^ A = B.

```
class Solution {
    public:
        int singleNumber(int A[], int n) {
            int result=A[0];
            for(int i=1;i<n;i++)
            {
                result= result^A[i];  /* Get the xor of all elements */
            }
            return result;
        }
    };
```

written by Deepalaxmi original link here

Solution 3

we use bitwise XOR to solve this problem :

first , we have to know the bitwise XOR in java

1. **0 ^ N = N**
2. **N ^ N = 0**

So..... if N is the single number

N1 ^ N1 ^ N2 ^ N2 ^.....^ Nx ^ Nx ^ N

= (N1^N1) ^ (N2^N2) ^.....^ (Nx^Nx) ^ N

= 0 ^ 0 ^ .....^ 0 ^ N

= N

```
public int singleNumber(int[] nums) {
    int ans =0;

    int len = nums.length;
    for(int i=0;i!=len;i++)
        ans ^= nums[i];

    return ans;

}
```

written by Nkeys original link here