

540. Single Element in a Sorted Array

Solution 1 (brute force)

INTUTION:

Here we have given that the array contains every elements twice and only element appear only once. So we get solution by XOR each element. As we know XORing same numbers will give 0 and XORing 0 with any number give the number itself. So we will get the single element by XORing all the elements.

ex . $1 \wedge 1 = 0$ and $0 \wedge 5 = 5$

```
class Solution{
public int singleNonDuplicate(int[] nums) {

    int x=0;

    for(int i =0; i<nums.length; i++){

        x= x^nums[i];

    }

    return x;

}
```

Time complexity : $O(n)$

Space complexity : $O(1)$

Solution 2 Binary Search

Page No. _____
Date _____

0 1 2 3 4 5 6 7 8

nums: [1, 1, 2, 3, 3, 4, 4, 8, 8]

↑
Single
Element

- We can see that,
On Right side of Single element the
Elements that appear twice are on
odd - Even Index

- ON left side, Element that appear twice
are on Even - odd Index

0	1	2	3	4	5	6	7	8
1	1	2	3	3	4	4	8	8
↑	↑		↑	↑				
ε	0		0	ε				

- We will apply binary search with,
 $s = 0$, $e = n - 1$
 ↑
 length of array,

we will check if mid is even, then
 $arr[mid] == arr[mid + 1]$

- If above condition satisfied then, it will
 be Even-odd position. Hence we are on
 left side of single element.

Hence we will update $start = mid + 1$

- 0

- ~~Also if $arr[mid] ==$~~

- Also if $mid = \text{odd}$,
then we will check if
 $arr[mid] == arr[mid-1]$
if condition satisfies, then $start = mid + 1$

- If both above case fails, we will move
 $end = mid$

~~The above case can be handled by a single case~~

→

0	1	2	3	4	5	6	7	8
1	1	2	3	3	4	4	8	8
↑							↑	
s							e	

- $Mid = \frac{0+8}{2} = 4$ even.

$arr[mid] == arr[mid+1]?$
 $4 == 5? \times$
 $\therefore e = mid$

→

0	1	2	3	4	5	6	7	8
1	1	2	3	3	4	4	8	8
↑		↑		↑				
s		M		e				

- $m(\text{even})$ $arr[mid] == arr[mid+1]? \times$
 $\therefore e = mid$

→

0	1	2	3	4	5	6	7	8
1	1	2	3	3	4	4	8	8
↑	↑	↑						
s	M	e						

- $M(\text{odd})$ $\therefore arr[mid] == arr[mid-1]? \text{ yes}$
 $1 == 1 \therefore s = mid + 1$


```

class Solution {
    public int singleNonDuplicate(int[] nums) {
        int n = nums.length;
        int start = 0;
        int end = nums.length-1;

        int mid = 0;
        if(nums.length == 1) return nums[0];
        while(start<end)

            /* why not start<=end? because it will lead to infinite loop. At last step we can
            see that start, mid and end are on same index and if we make condition start<=end, then w
            e will never exit the loop*/

            {

                mid = start +(end-start)/2;
                if((mid%2==0 && nums[mid]==nums[mid+1] )|| (mid%2==1 &&nums[mid]==nums[mid-1])) {
                    start = mid+1;
                }
                else{
                    end = mid;
                }

            }
        return nums[start]; //we can also return nums[end]

    }
}

```

Time complexity : $O(\log n)$

Space complexity : $O(1)$