Notes

Day - 2

Date __/__/_____

Problem: Single Element in a sorted array

nums = [1,1,2,3,3,4,7,8,8]

output: 2

Self

// I think take a ds (data structure) that will count freq of each element.

// by took a map and insert all element return the element whose freq is one.

```
unordered_map<int,int> m;
int c=0;
for(int i=0; i<n; i++)
    mp[nums[i]]++;
for(auto it = mp.begin(); it != mp.end(); it++)
{
    if(it->second==1)
    {
        c = it->first;
    }
}
return c;
```

Brute force

TC: O(N)

SC: O(N)

Remarks

Date__/__/_____ No

Optimal:
using BS —
low = 0, high = n-2 (because we grow desired element at last idx)

high = n-2
when low crosses high then desired element at low.

TC: O(log N)
SC: O(1)

Remarks

Notes

Date__/__/_____

Problem: Search in rotated sorted array.

Self nums = [9,5,6,7,0,1,2], target = 0

// the solution I think for this approach is using for loop linearly traverse and check arr[i] == target return i else -1.

// optimal using BS in O(logN)

low = 0, high = n-1, mid = (l+h)/2

// if mid == target return mid

// if mid >= low

// if arr[mid] >= target and arr[low] <= tar

high = mid-1;

low = mid+1;

// and vice versa

return -1 //

Remarks

Date__/__/------

Problem: Median of two sorted array. nums1 = [1,3], nums2 = [2] output

// without BS, I think like create another vector of nums

then traversing in nums1 and insert in nums3

same with nums2 and then sort nums3 vector.

After that find median.

// find size of nums3

if size % 2 == 0 ; idx = k/2

(nums[idx] + nums3[idx-1]) / 2

else

nums[idx]

return res //.

Tc = O(log(m+n))

Sc = O(N)

Remarks

Notes Date__/__/_____

Abdul's Video (optimal)

// calculate mid and then take two pointer low = 0, high = n-1

// after getting mid check if the left half has total number of element equal to mid position. If not get the remaining element from second array.

// condition of partitioning $l_1 <= r_2$ and $l_2 <= r_1$ if satisfied return $\max(l_1, l_2)$ <- if odd $(\max(l_1, l_2) + \min(r_1, r_2))/2$

TC: O(log(m,n))
SC: O(1) //

Remarks _____

Date__/__/_____

Problem: kth element of two sorted array.

Self

arr1[] = {2, 3, 6, 7, 9}
arr2[] = {1, 4, 8, 10}
K = 5
output: 6

// create another arr3 = [1, 2, 3, 4, 6, 7, 9,

return k arr3[k-1]

= 6 //

Basically I did put all element of arr1 and arr2 in arr3 and sort arr3 after that return (k-1) element.

TC: O(NlogN)
SC: O(N)

So Brute force //

marks

optimal is same as we did in median of two sorted.

// apply BS in an array of small size. Start iterate with two pointer i.e left and right.

// find middle

// take element from low to middle of arr[i] and rem. from second arr

// check left half is valid print max of both array's last element if not move range l2 > r1 toward right else l1 > r2 towards left.

TC: log(min(m,n))
SC: O(1) //

Remarks