SEARCHING AND SORTING LEVEL 3

21 September 2023 11:48

CLASS-13 → 20/09/2023

https://www.linkedin.com/in/manojoffcialmj/

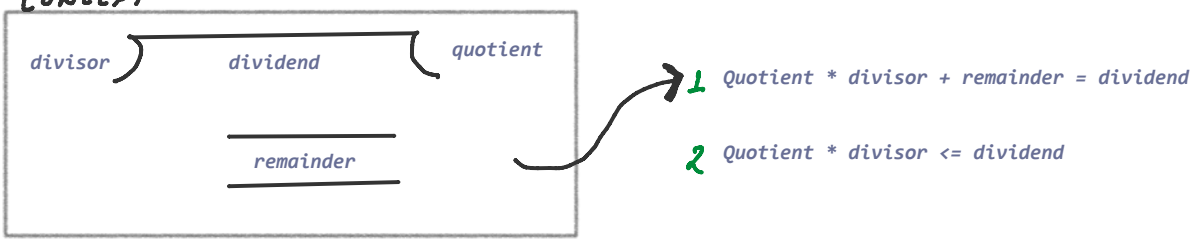
SEARCHING & SORTING
LEVEL-03

1. Divide two number using Binary search without using any / and % operator

PROBLEM:
*Given two integers **one is a dividend** and **the other is the divisor**, we need to **find the quotient** when the dividend is divided by the divisor without the use of any " / " and " % " operators.*

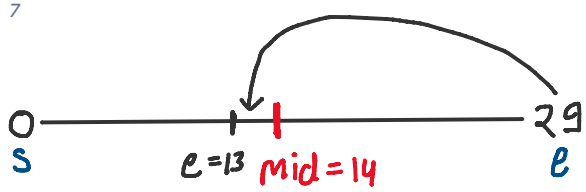
Examples 01: *Input: dividend = 10, divisor = 2* *Output: 5* *Explanation: 10/2 = 5*	***Examples 02:*** *Input: dividend = 10, divisor = 3* *Output: 3* *Explanation: 10/3 = 3.33333… which is truncated to 3*
Examples 03: *Input: dividend = 10, divisor = -2* *Output: -5* *Explanation: 10/-2 = -5.*	***Examples 04:*** *Input: dividend = -10, divisor = 2* *Output: -5* *Explanation: -10/2 = -5.*

CONCEPT



DRY RUN

DRY RUN THIS EXAMPLE:
Input: dividend = 29, divisor = 7
Output: 4



S=0 *Is 14 a possible ANS?*

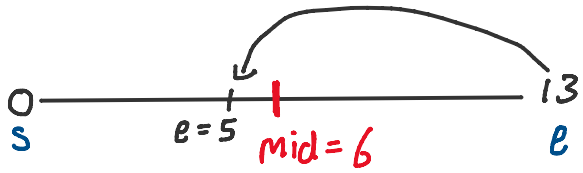
s = 0
e = 29
mid = $\frac{0+29}{2}$
= 14

Is 14 a possible ANS?

mid * divisor <= 29

14 * 7 ⇒ 98 <= 29 FALSE

↳ go to left side
e = mid - 1



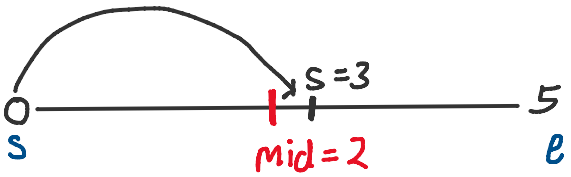
s = 0
e = 13
mid = $\frac{0+13}{2}$
= 6

Is 6 a possible ANS?

mid * divisor <= 29

6 * 7 ⇒ 42 <= 29 FALSE

↳ go to left side
e = mid - 1



s = 0
e = 5
mid = $\frac{0+5}{2}$
= 2

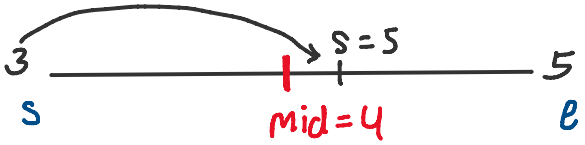
Is 2 a possible ANS?

mid * divisor <= 29

2 * 7 ⇒ 14 <= 29 TRUE

↳ store mid in ANS
ANS = 2

↳ go to right side
s = mid + 1



s = 3
e = 5
mid = $\frac{3+5}{2}$

Is 4 a possible ANS?

mid * divisor <= 29

e = 5

mid = $\frac{3+5}{2}$

= 4

↳ mid * divisor <= 29

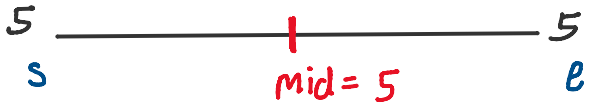
4 * 7 ⇒ 28 <= 29 True

↳ store mid in ANS

ANS = 4

↳ go to right side

s = mid + 1



s = 5

e = 5

mid = $\frac{5+5}{2}$

= 5

↳ Is 5 a possible ANS?

mid * divisor <= 29

5 * 7 ⇒ 35 <= 29 False

↳ go to left side

e = mid - 1

(s <= e) s = 5 <= e = 4

END

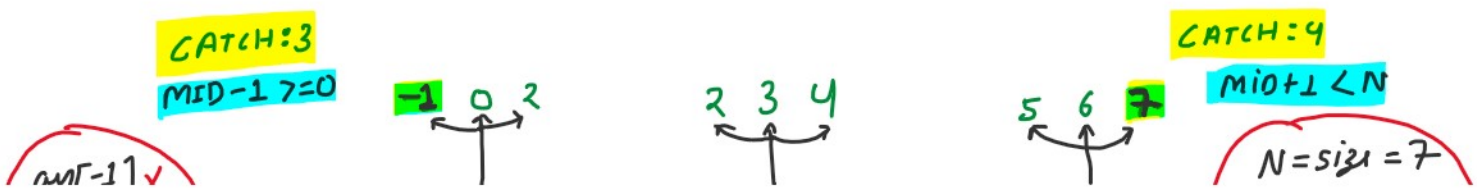
↳ NO RUK jao

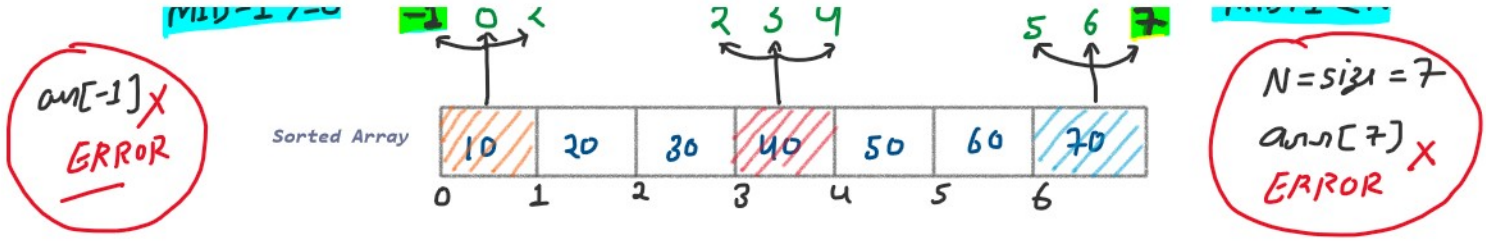
```cpp
// Program 01: Divide two number using Binary search without using any / and % operator
#include<iostream>
using namespace std;

// Divide two number using Binary search
int getQuotient(int dividend,int divisor){
    int s = 0;
    int e= dividend;
    int mid = s+(e-s)/2;
    int ANS = -1;

    while (s<=e)
    {
        // Return quotient
        if(mid * divisor == dividend){
            return mid;
        }
        // Goto to left side to get possible quotient
        else if(mid * divisor > dividend){
            e = mid - 1;
        }
        // Store possible mid in ANS and goto to right side to get possible quotient
        else if(mid * divisor < dividend){
            ANS = mid;
            s = mid + 1;
        }

        // Updated mid
        mid = s+(e-s)/2;
    }
    return ANS;
}

// Time Complexity: O(logN), where N is dividend.
// Space Complexity: O(1)

int main(){
    int dividend = 29, divisor = -7;

    int quotient = getQuotient(abs(dividend),abs(divisor));

    // Now we need to decide ki sign konsa lagaye +VE ya -VE
    if((dividend > 0 && divisor < 0) || (dividend < 0 && divisor > 0)){
        quotient *=-1;
    }
    cout<<"Quotient: "<<quotient<<endl;
}
```

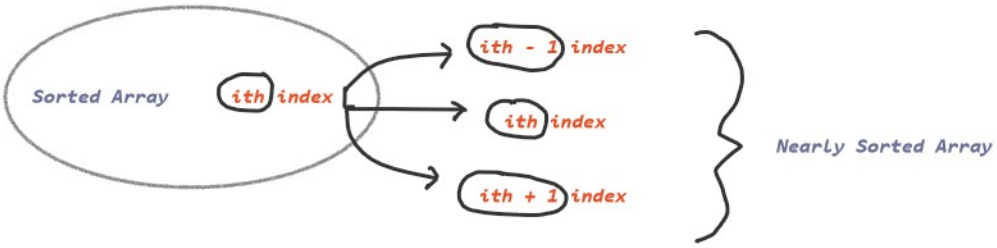
2. Binary search on nearly sorted array U.U.U.Imp

What is nearly sorted array?

Real World Example:
*Lets assume ki ek ladka hai jiska name **SOHAN** hai or wo **gali no. 10** me rahata hai to hum bol sakte hai ki **SOHAN** nearly **gali no. 9, 10,** yaa **11** me se kisi ek gali me rahta hoga.*







Actual Example:
*Sorted array me jo **40 value at index 3** par hai to nearly sorted array me 40 value ke three **possible index 2, 3, yaa 4** me se kisi ek index par lie zaroor karti hogi.*

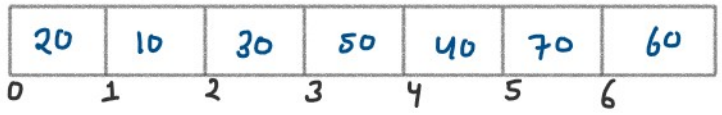
Normal sorted array	Nearly sorted array

→ if(arr[mid] == target)
{
return mid;
}
→ if(target > arr[mid]
{
↳Right s=mid+1
}
Else
{
↳ left e=mid-1
}
	→ if (arr[mid-1] == target) CATCH-3 mid-1 >=0
	{
	return mid-1;
	}
	if(arr[mid] == target)
	{
	return mid;
	}
	if(arr[mid+1] == target) CATCH-4 mid+1 < N
	{
	return mid+1;
	}
	→ if(target > arr[mid]
	{
	↳Right s=mid+2 CATCH:1
	}
	Else
	{
	↳ left e=mid-2 CATCH:2
	}
	Mid+2 and Mid-1 kyun kar rahe hai?, kyunki hum 1 value ko target se 2 times compare nahi karna chahte hai

DRY RUN

Nearly Sorted Array

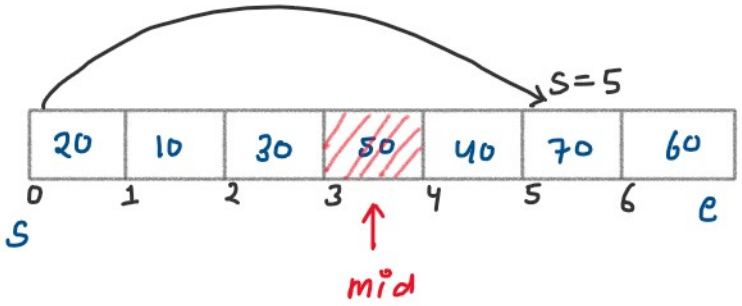
arr



Target = 70

Output = 5

Iteration 0:



s = 0

e = 6

mid = $\frac{0+6}{2}$

= 3

arr[mid-1] ⇒ 30 == 70 ✗

arr[mid] ⇒ 50 == 70 ✗

arr[mid+1] ⇒ 40 == 70 ✗

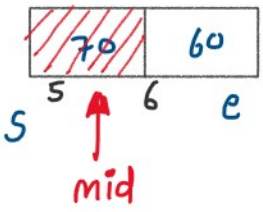
arr[mid] < target ✓

→ Go to right side

s = mid + 1 ✗ Agar mid+1 karunga to iska matlb yeh hai ki me target se 40 value ko 2 times compare kar rha hu.. CATCH: 01

→ s = mid+2 ✓

Iteration 1:



s = 5

e = 6

mid = $\frac{5+6}{2}$

= 5

arr[mid-1] ⇒ 40 == 70 ✗

arr[mid] ⇒ 5 == 70 ✓

arr[mid+1] ⇒ 60 == 70 ✗

→ Return MID

END

```
// Program 02: Binary search on nearly sorted array
#include<iostream>
#include<vector>
using namespace std;
```

```cpp
// Program 02: Binary search on nearly sorted array
#include<iostream>
#include<vector>
using namespace std;

// Find target in nearly sorted array
int searchNearlySorted(vector<int> arr,int target){
    int n=arr.size();
    int s=0;
    int e=n-1;
    int mid = s+(e-s)/2;

    while(s<=e){
        // CATCH 03 -> ARR[-1] PAR CODE FAT JAYEGA
        if(mid-1>=0 && arr[mid-1]==target){
            return mid-1;
        }
        if(arr[mid]==target){
            return mid;
        }
        // CATCH 04 -> ARR[N] PAR CODE FAT JAYEGA
        if(mid+1<n && arr[mid+1]==target){
            return mid+1;
        }
        // CATCH 01 -> GOTO RIGHT SIDE TO GET TARGET'S INDEX
        if(arr[mid]<target){
            s=mid+2;
        }
        // CATCH 02 -> GOTO LEFT SIDE TO GET TARGET'S INDEX
        else{
            e=mid-2;
        }
        mid = s+(e-s)/2;
    }
    return -1;
}

/*
Time Complexity: O(logN), where N is array's length
Space Complexity: O(1)
*/

int main(){
    vector<int> arr{20,10,30,50,40,70,60};
    int target = 100;

    int index = searchNearlySorted(arr,target);

    if(index == -1){
        cout<<"Target not found"<<endl;
    }else{
        cout<<"Target "<<target<<" found at index "<<index<<endl;
    }

    return 0;
}

/*
Example 01:
Input: arr[20,10,30,50,40,70,60] , N=7 , and target = 70
Output: Target 70 found at index 5

Example 02:
Input: arr[20,10,30,50,40,70,60] , N=7 , and target = 20
Output: Target 20 found at index 0

Example 03:
Input: arr[20,10,30,50,40,70,60] , N=7 , and target = 60
Output: Target 60 found at index 6

Example 04:
Input: arr[20,10,30,50,40,70,60] , N=7 , and target = 100
Output: Target not found
*/
```

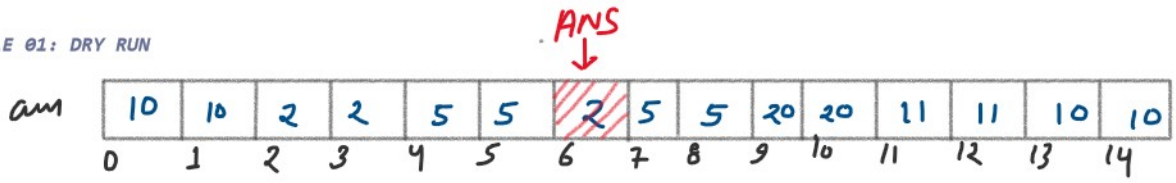
3. Find the Number Occurring Odd Number of Times

Conditions:
1. *All elements - even number of times occurs except one* → ODD Element
2. *All repeating elements - pairs repeat and pairs are not repeated*
3. *Ek element ek bar me 2 times nahi aa skta hai*

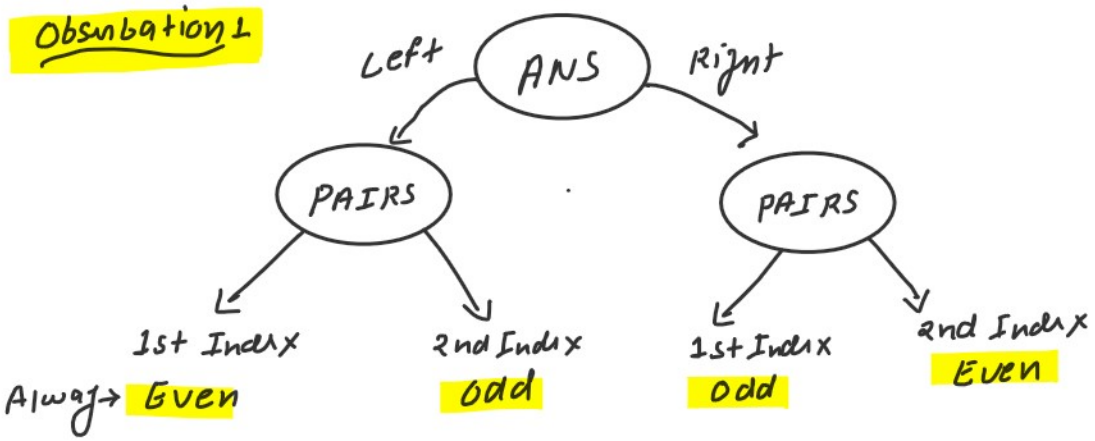
1. All elements - even number of times occurs except one
2. All repeating elements - pairs repeat and pairs are not repeated
3. Ek element ek bar me 2 times nahi aa skta hai

OUTPUT --> Find element that occurs odd time

EXAMPLE 01: DRY RUN



Observation 1



Observation 2

Ans ⇒ Always lies at Even Index

Observation 3

Single Element

arr [2]

if(s==e)
↳ return s

DRY RUN

Iteration 01

s = 0

e = 14

mid = $\frac{0+14}{2}$

= 7

if(MID%2 != 0) {

if(arr[mid] == arr[mid-1])

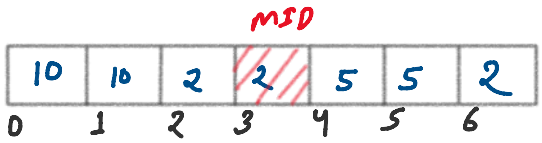
↳ 5 == 2 False

Else

↳ go to left side

e = mid - 1

Iteration 01



s = 0

e = 6

mid = $\frac{0+6}{2}$

= 3

if(MID%2 != 0) {

if(arr[mid] == arr[mid-1])

↳ 2 == 2 True

go to right side

s = mid+1

Iteration 02

s = 4

e = 6

mid = $\frac{4+6}{2}$

= 5

if(MID%2 != 0) {

if(arr[mid] == arr[mid-1])

↳ 5 == 5 true

go to right side

s = mid+1

Iteration 03

s = 6

e = 6

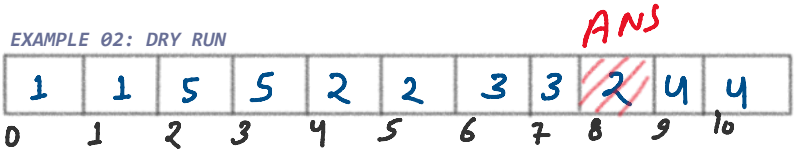
if(s == e)

↳ Return s END

OUTPUT = 2

DRY RUN

EXAMPLE 02: DRY RUN



Iteration 0

$s = 0$

$e = 10$

$mid = \frac{0+10}{2}$

$= 5$

```
if(mid %2 != 0){
    if(arr[mid] == arr[mid-1])
```

↳ 2 == 2 True

go to right side

$s = mid+1$

Iteration 01

$s = 6$

$e = 10$

$mid = \frac{6+10}{2}$

$= 8$

```
if(mid %2 == 0){
    if(arr[mid] == arr[mid+1])
```

↳ 2 == 4 False

ELSE

↳ store mid in ANS and go to left side

$ANS = mid$

$e = mid$

END/ANS

END/ peak

Iteration 02

$s = 6$

$e = 8$

$mid = \frac{6+8}{2}$

$= 7$

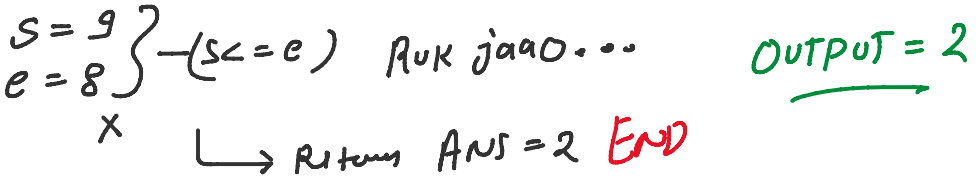
$ANS = 2$

```
if(mid %2 != 0){
    if(arr[mid] == arr[mid-1])
```

3 == 3

↳ go to right side

$s = mid+2$




```cpp
// Program 03: Find the Number Occurring Odd Number of Times
#include<iostream>
#include<vector>
using namespace std;

// Find number occurring odd number of times
int findOddOccuringElement(vector<int> arr){
    int n=arr.size();
    int s=0;
    int e=n-1;
    int mid=s+(e-s)/2;
    int ans = -1;

    while(s<=e){
        // Single element
        if(s==e){
            return s;
        }
        // when mid index is odd
        if(mid & 1){

            // Goto right side and CATCH: 01 mid-1>=0 because arr[-1] nahi ho skta hai
            if(mid-1>=0 && arr[mid]==arr[mid-1]){
                s=mid+1;
            }
            // Goto left side
            else{
                e=mid-1;
            }
        }
        // when mid index is even
        else{

            // Goto right side to get ans and CATCH: 02 mid+1<n because agar n=7 hai to arr[n] nahi ho skta hai
            if(mid+1<n && arr[mid]==arr[mid+1]){
                // CATCH: 03 we don't want to capare twice for one element
                s=mid+2;
            }
            // CATCH: 04 Store mid in ans and go to left side to get ans (ans/end || peak/ans)
            else{
                ans=mid;
                e=mid;
            }
        }
        // updated mid
        mid=s+(e-s)/2;
    }
    return ans;
}
/*
Time Complexity: O(logN), where N is array's length
Space Compexity: O(1)
*/

int main(){
    vector<int> arr{10,10,2,2,5,5,2,5,5,20,20,11,11,10,10};

    int ansIndex = findOddOccuringElement(arr);

    if(ansIndex==-1){
        cout<<ansIndex<<endl;
    }
    else{
        cout<<"Final ans is "<<arr[ansIndex]<<endl;
    }

    return 0;
}

/*
Example 01:
Input: arr[10,10,2,2,5,5,2,5,5,20,20,11,11,10,10], N=15
Output: Final ans is 2

Example 02:
Input: arr[1,1,5,5,2,2,3,3,2,4,4], N=11
Output: Final ans is 2

Example 03:
Input: arr[1,1,5,5,2,2,3,3,4,4], N=10
Output: -1

Example 04:
Input: arr[1,1,5,5,2,2,3,3,4,4,4,4]
Output: -1
*/
```