19-02-2023

Date.................

Que) search in a nearly sorted array.

0	1	2	3	4	5	6
10	3	40	20	50	80	70

Nearly sorted array means, if the array was sorted then it was like:-

0	1	2	3	4	5	6
3	10	20	40	50	70	80

Now, a number that was supposed to be on ith position in sorted array. Now it possibly on any of the 3 positi-ons.

→ i+1th index.

↘ ith index

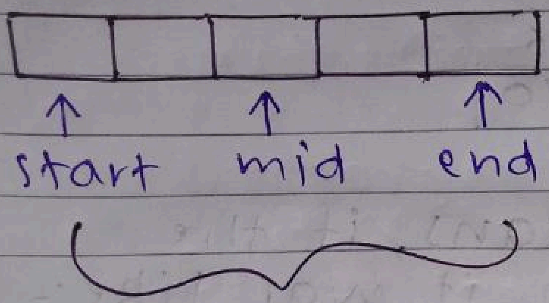
↓ i-1th index

For example, 3 was supposed to be on 0th index in sorted array, but it is on i+1, i.e., 0+1 = 1st index in the original array.

10 was supposed to be on 1st index in sorted array, but it is on i-1, i.e., 1-0 = 0th index in the original array.

50 was supposed to be on 4th index in sorted array, but it is on same ith, i.e., 4th index in the original array.

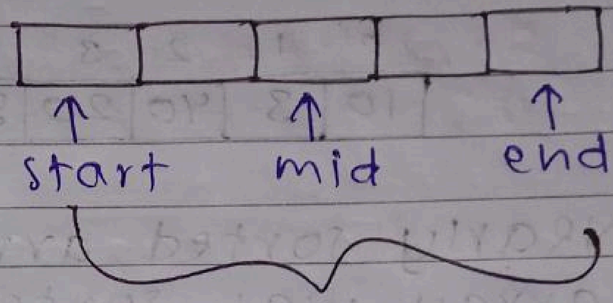
sorted array



Here, first we find mid element then we compare that mid element with the target, only one if condi-tion is required.

```
if(arr[mid] == k)
{
      return mid;
}
if(arr[mid] > k)
{
      e = mid-1;
}
else
{
      s = mid+1;
}
```

nearly sorted array



Here, first we find mid element, then we compare the target with mid, mid-1 & mid+1, 3 if conditions is required.

```
if(arr[mid] == k)
{
      return mid;
}
if(arr[mid-1] == k)
{
      return mid-1;
}
if(arr[mid+1] == k)
{
      return mid+1;
}
if(k > arr[mid])
{
      s = mid+2;
}
else {
      e = mid-2;
}
```

s = mid+2; is because we've already checked mid+1 eleme-nt so it's start from mid+2 now code :-

```
#include<iostream>
#include<vector>
using namespace std;

int binarySearch(vector<int> arr,
                              int target )
{
    int start = 0;
    int end = arr.size()-1;
    int mid;
    while(start <= end) {
        mid = (start + end)/2;
        if(target == arr[mid]) {
            return mid;
        }
        if(mid-1 >= 0 && target ==
                         arr[mid-1]){
            return mid-1;
        }
        if(mid+1 < arr.size() && target
                      == arr[mid+1]) {
            return mid+1;
        }
        if(target < arr[mid]) {
            end = mid-2;
        }
        else {
            start = mid+2;
        }
    }
return -1;
```

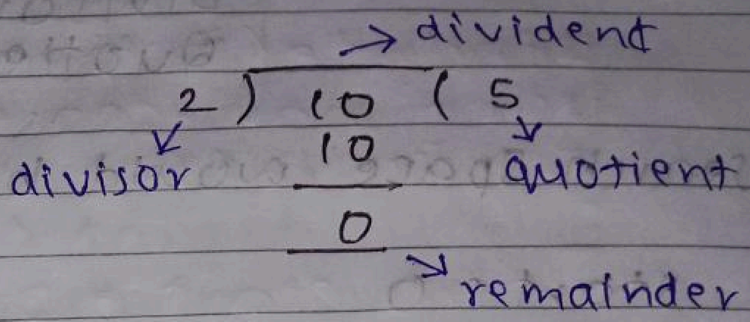
```
int main()
{
    vector<int> arr{10, 3, 40, 20, 50, 80,
                                        70};
    int target;
    cout << "enter element: ";
    cin >> target;

    int ans = binarySearch(arr, target)

    if(ans == -1){
        cout << "element not present";
    }
    else{
        cout << "element " << target <<
        " is present at " << ans;
    }
    return 0;
}
```

Ques. Divide two numbers using Binary search.

dividend = 10
divisor = 2
quotient = 5
remainder = 0



if dividend is 10 then the quotient always lies between 0 → 10. Means, whatever the dividend is, Quotient always lies between 0 → dividend.

this is our search Space.

Formula to find the dividend,

Quotient * Divisor + Remainder = Dividend

In our given question, we don't need remained value, so, we can use the formula,

Quotient * Divisor <= Dividend

In this scenario also, we use the same strategy that we used in square root Question.

For example, Dividend = 22
Divisor = 7
Quotient = ?

Search space would be, $\boxed{0 \rightarrow 22}$

start = 0
end = 22
mid = 0+22/2 = 11
↳ may be this is the Quotient. So, we apply the formula.

Quotient * Divisor <= Divident
11 * 7 <= 22
$\boxed{77 <= 22}$

↙

If
Quotient * Divisor > Dividend,
it means that mid > our target value, we have to search it on the left side, i.e.,
end = mid - 1

Now, start = 0
end = 11 - 1 = 10
mid = 5
↳ again apply the formula on this mid.

Quotient * Divisor <= Divident

$5 * 7 <= 22$

$\boxed{35 <= 22}$

this again means that mid > target, again search on left side, i.e., end = mid - 1.

Now, start = 0
end = 5 - 1 = 4
mid = 2
↳ again apply the formula on this mid.

Quotient * Divisor <= Dividend

$2 * 7 <= 22$

$\boxed{14 <= 22}$ → in this case, Quotient * Divisor < Dividend means, mid < target, we have to search on the right side & along with that store the mid.

By using the same approach, we can solve the problem.

code :-

```
#include<iostream>
using namespace std;
int solve(int dividend, int divisor)
{
    int start = 0;
    int end = abs(dividend);
    int mid;
    int ans = 0;
    while (start <= end){
        mid = (start + end)/2;
        if(abs(mid * divisor) ==
                    abs(dividend)){
            return mid;
        }
        else{
            if(abs(mid * divisor) >
                        abs(dividend)){
                end = mid - 1;
            }
            else {
                ans = mid;
                start = mid + 1;
            }
        }
    }
    if((divisor < 0 && dividend < 0) ||
       (divisor > 0 && dividend > 0)){
       return ans;
    }
    else {
        return -ans;
    }
}

int main()
{
    int dividend = 22;
    int divisor = 7;

    int ans = solve(dividend,
                         divisor);

    cout << "\n Ans is: " << ans;

    return 0;
}
```

Ques. Find the odd occurring element in an array.

In this problem, we've given an array where all element appear even no. of times except one element.

0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	2	3	3	4	4	3	600	600	4	4

- all elements occur even no. of times except one element.
- all repeating occurrence of element appear in pairs, and pairs are not adjacent (there cannot be more than 2 consecutive occurrence of any element).
- we have to find the element that appears odd no. of times.

we can solve this question using XOR approach, & at the end we left with the odd element,

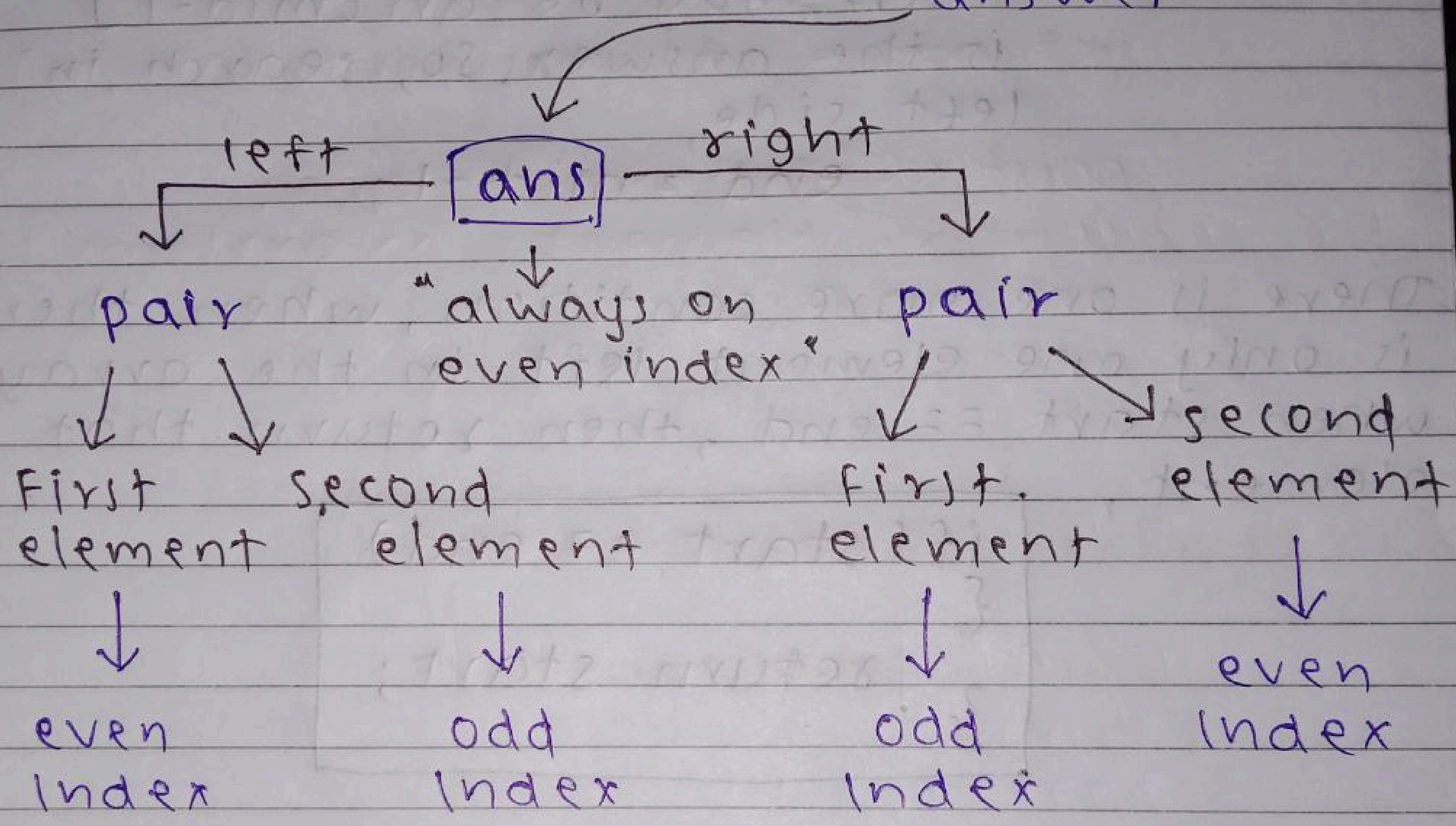
```
int ans = 0;
for(int i = 0; i < n; i++)
{
    ans = ans ^ arr[i];
}
```

↳ time complexity is O(n) because of Linear search approach.

But we can solve it in Binary search approach,

0	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	2	3	3	4	4	3	600	600	4	4

↓ this is the answer



First calculate the mid, then check if "mid is even" :-

↳ then we check if arr[mid] == arr[mid+1], means we are at left side & the ans is at right side. So,

start = mid + 2;

because, we already checked mid+1 element

↳ If arr[mid] != arr[mid+1], then may be mid is the answer. So, search left side,

end = mid;

If "mid is odd" :-

↳ then we check if arr[mid] == arr[mid - 1], means we are at left side & the ans is at right side. So,

start = mid +1;

↳ If arr[mid] != arr [mid - 1], then may be arr[mid-1] is the answer. So, search in left side,

end = mid -1;

There is one more condition, when there is only one element left in the array when start == end, then return that element,

```
if(start == end)
{
    return start;
}
```

code:-

```
#include<iostream>
using namespace std;

int oddOccurrence(int arr[], int n)
{
    int start = 0;
    int end = n-1;
    int mid;
    while(start <= end){
        mid = (start + end)/2;
        if(start == end){
            return start;
        }
        if(mid % 2 == 0){
            if(arr[mid] == arr[mid+1])
            {
                start = mid + 2;
            }
            else
            {
                end = mid;
            }
        }
        else{
            if(arr[mid] == arr[mid-1]){
                start = mid + 1;
            }
            else{
                end = mid - 1;
            }
        }
    }
    return -1;
  }

int main()
{
    int arr[] = {1, 1, 2, 2, 3, 3, 4, 4, 3,
                              600, 600, 4, 4};
    int n = 13;
    int ans = oddOccurrence(arr, n);

    cout << "Index : " << ans;
    cout << "element: " << arr[ans];
    return 0;
}
```