WEEK-4

SEARCHING AND SORTING

CLASS ⇒ III

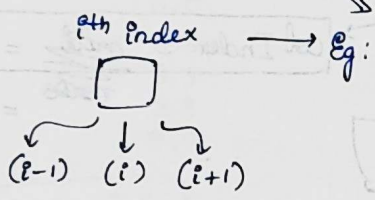
Q: Binary Search in a nearly sorted array.

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

target = 40

Sol:

sorted → 3 (0), 10 (1), 20 (2), 40 (3), 50 (4), 70 (5), 80 (6)



Eg: 3 can lie on (-1) (0) (1) indexes.

Eg: 20 (2) can be on (2-1) or (2) or (2+1)
(1) or (2) or (3)
↳ in ques, 20 is on 3

Eg: 70 (5) can be on (5-1) or (5) or (5+1)
(4) or (5) or (6)

Similar for all elements for an array.

Now,

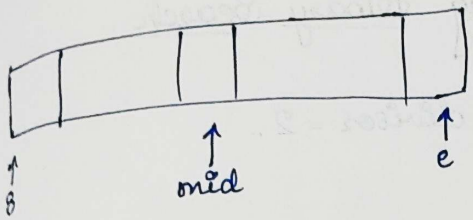
10	3	40	20	50	80	70
0	1	2	3	4	5	6

Approach 1: Linear search for (0 → < n) ⇒ T.C = O(n)

O(n log n) is even more worst than O(n)

Approach 2: → Sort array
→ Then Binary search } TC ⇒ O(log n + n log n)
= O(n log n)

Normal sorted



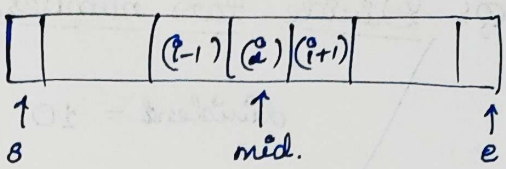

```
if (arr[mid] == target)
    return mid;
```

↓

```
if (target > arr[mid])
    RS → s = mid + 1;
else if ( LS → e = mid - 1;
```

Difference in Binary Searches

Nearly sorted.




```
if (arr[mid] == target)
    return mid;
else if (arr[mid-1] == target)
    return mid-1;
else if (arr[mid+1] == target)
    return mid+1;
```

↓

```
if (target > arr[mid])
    ↳ RS ⇒ s = mid + 2;
else
    LS ⇒ e = mid - 2;
```

NOTE: `mid + 1 <= arr.size();`

`mid - 1 >= start` or `mid - 1 >= 0`

↓ more optimised than zero ∵ less checkings.

TC → O(log n)

↳ will be same as Binary search

Q> Divide two numbers using Binary Search.

dividend = 65 and divisor = 4.

find quotient = ?

★ like that square root wala question, first create a search space.

∴ Search space = {0, dividend}

Conditions:

① if (mid × divisor == dividend) → return mid;

② if (mid × divisor > dividend) → left search [e = mid − 1]

③ else → store ans

→ right search [s = mid + 1]

Dry Run:

65 ← dividend
4 ← divisor

0 (start) 32 (mid) 65 (end)

mid × divisor = 32 × 4 = 128 > 65

↳ Left search.

0 (s) 15 (mid) 31 (e)

15 × 4 = 60 < 65 → store ans

ans = 15

→ right search.

16 (s) 23 (m) 31 (e)

23 × 4 = 92 > 65 → left

mid x divisor

16 19 22
↑ ↑ ↑
s mid e

$= 19 \times 4 = 76 > 65$

↳ left.

16 17 18
↑ ↑ ↑
s mid e

$17 \times 4 = 68 > 65$

↳ left.

16 16
↑ ↑
s e

start end
16
↑
mid

$16 \times 4 = 64 < 65$

↳ store ans

ans = 16

↳ right search.

$s = mid + 1 \Rightarrow s = 17$

s = 17 and e = 16

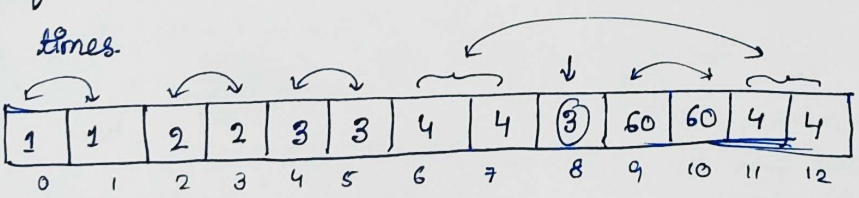
↳ Ruk jao ;!

Q) Find the Odd Occuring Element in an Array.

↳ all elements occur even numbers of times except one.

↳ all repeating occurance of elements appear in pair, and pairs are not adjacent. (there cannot be more than 2 consecutive occurance of any pair)

↳ find the element that appears odd number of times.



1	1	2	2	3	3	4	4	3	60	60	4	4	

* XOR approach $\{a \wedge a = 0,\ a \wedge b = 1\}$

↳ $1 \wedge 1 \wedge 2 \wedge 2 \wedge 3 \wedge 3 \wedge 4 \wedge 4 \wedge (3) \wedge 60 \wedge 60 \wedge 4 \wedge 4$

↳ ans = 3, but T.C = O(n)

★ IMP BINARY SEARCH ★

↳ very less type of questions

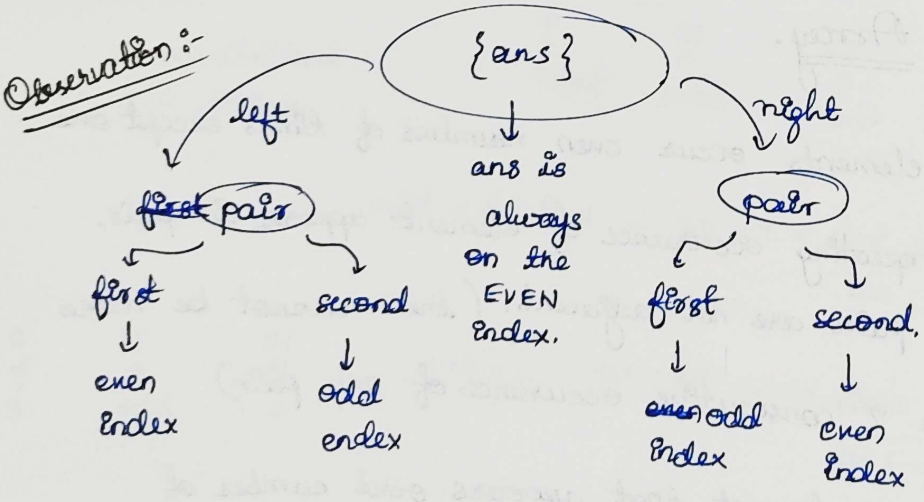
↳ i) Classic questions : BS, lower bound, upper bound, total occurance, pivot element, peak ele, search in rotated sorted array, sq. root.

ii) Find out in Search space : Apply BS for a range.

iii) Index ke andar Observation : This question.

1	1	2	2	3	3	4	4	3	60	60	4	4	
0	1	2	3	4	5	6	7	8	9	10	11	12	
even	odd	e	o	e	o	e	o	e	o	e	odd	even	

Observation :-



$s=0$, $e=n-1$, $mid = s+(e-s)/2$ or $s+e/2$.

```
while (s <= e) {
    if (s == e) → return s;        // only single element array
    if (mid % 2 == 0)
    {
        // Even case
        [ ]
    }
    else
    {
        // odd case
        [ ]
    }
}
```

```
if (a[mid] == a[mid+1])
    // it means we are on
    // the left side of ans
    {
        // right search.
        s = mid+1;
    }
else
    {
        e = mid;
    }
```

① mid → even.

[ ] = [ ] → Left me khada hu → search right.

(mid) == (mid+1)

Even odd

$s = mid+1;$

ans 1

ans 1 or Left me hoga

[ ] ≠ [ ] → ans mid bhi ho sakta hai, ya fir left me bhi ho sakta hai.

(mid) != (mid+1)

Even odd

∴ $e = mid$

② mid → odd

compare ~~check this~~

[ ] == [ ] → left side me khada hu.

(mid-1) == (mid)

Even odd

↳ ans Right me hoga.

↳ Right search.

$s = mid+1;$

↳ why, because last time we had checked mid+1, not this time.

ans

even odd → (because ans cant be on odd index)

[ ] != [X] → ~~ans 1 or right me hoga~~

mid-1 mid

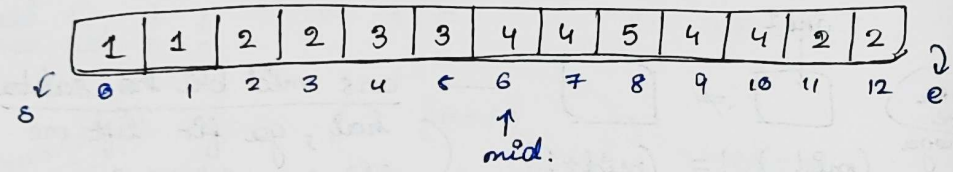
Left search

∴ ~~Right search~~ $e = mid-1;$

Summary :

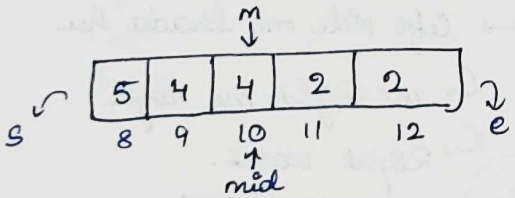
```
if (mid % 2 == 0)
{
    if (a[mid] == a[mid+1])
        s = mid+2;
    else
        e = mid;
}

else
{
    if (a[mid] == a[mid-1])
        s = mid+1;
    else
        e = mid-1;
}
```

Dry Run :



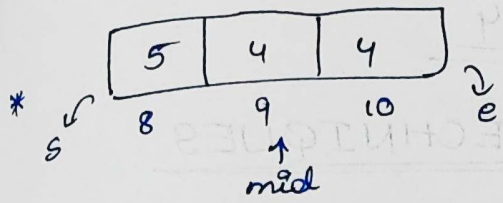
* mid is even

[4] == [4] → right → s = mid+2
mid even (6) mid+1 odd (7)
= 6+2
s = 8



* mid is even

[4] != [2] → e = mid
mid even (10) mid+1 odd (11)
e = 10



* mid is odd

[5] != [4] → left search.

(mid-1) (mid)

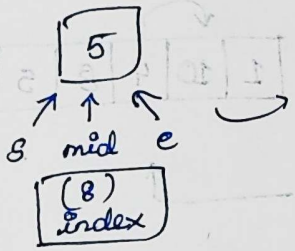
even(8) odd(9)

e = mid - 1

e = 9 - 1

e = 8

Now,



s == e → return mid; (or return s or e)