Today's Content

- * Every element is repeated twice except for one element
- * Find sqrt()
- * Search in rotated sorted array.

Qn: In array a[N]. Every element occurs twice except one unique element. Find the unique element.

Note: duplicate elements are together

idea 1: Take XOR of all elements. TC: O(n), SC: O(1)

Eg:-

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	3	1	1	8	8	10	10	19	6	6	2	2	4	4

obs: * Before the unique element, all first occurence are at EVEN IDX
* After the unique element, all first occurence are at ODD IDX

To apply BS: → Target : unique element.
↳ Search Space : entire array.

Case 1:

```
if a[mid] is unique ?   return a[mid]
if (a[mid-1] != a[mid] && a[mid] != a[mid+1])
```

Case 2:

```
if (a[mid-1] == a[mid]) {
        mid = mid-1
}
if (mid % 2 == 0) { // Even idx, go right
        l = mid + 2
}
else { //odd idx, go left.
        r = mid - 1
}
```

Dry-run

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
3	3	1	1	8	8	10	10	19	6	6	2	2	4	4

l	r	m	is m unique?	Am I first occurence?	Idx
0	14	~~7~~ 6	✗	✗ m = m-1 = 6	Even : l = m+2
8	14	11	✗	✓	Odd : r = m-1
8	10	9	✗	✓	odd : r = m-1
8	8	8	✓	found. return a[8] = 19. a[m] →	

// Pseudocode:

```
int findUnique (int a[], int n) {
    l=0, r=n-1
    if (n==1) { return a[0] }
    if (a[0] != a[1]) { return a[0] }
    if (a[n-1] != a[n-2]) { return a[n-1] }
    while (l <= r) {
        m = (l+r)/2
        if (a[m-1] != a[m] && a[m] != a[m+1]) { return a[m] }
        // Case 2
        if (a[m-1] == a[m]) { m = m-1 }
        // m is at Ist occurence.
        if (m%2==0) { l = m+2 } // go right
        else { r = m-1 } // go left
    }
}
```

TC: $O(\log n)$

SC: $O(1)$

Qn: Given a positive number N. Find sqrt(N).

↳ floor(sqrt(N))

↳ integer part only

sqrt(25) = 5

sqrt(20) = 4

sqrt(10) = 3

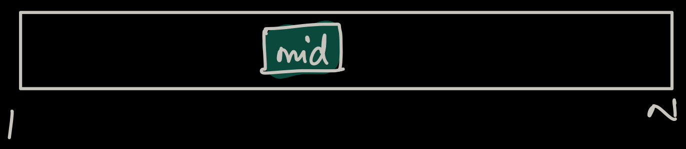
idea 1: Intermediate Class TC: $O(\sqrt{N})$, SC: $O(1)$

```
i=1, ans
while(i*i <= N) {
    ans = i
    i++
}
```

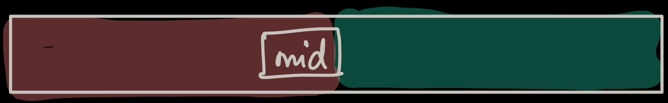
idea 2: Binary Search → Target: floor(sqrt(n))

↳ Search Space: [1 – N]

Can we discard?

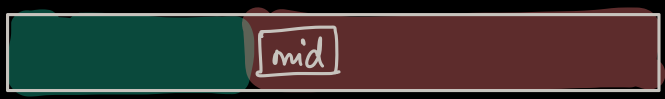


Case 1: mid* mid == N return mid



Case 2: mid * mid < N : ans = mid

l = mid+1



Case 3: mid * mid > N : r = mid - 1

Dry-Run

N = 50

ans = ~~-0~~ ~~6~~ 7

l	r	m	
1	50	25	25*25 > 50 : go left : r=m-1
1	24	12	12*12 > 50 : go left : r=m-1
1	11	6	6*6 < 50 : go right : l = m+1
7	11	9	9*9 > 50 : go left : r=m-1
7	8	7	7*7 < 50 : go right : l = m+1
8	8	8	8*8 > 50 : go left : r=m-1
8	7	STOP [l > r]!	

Code: To Do   * Use long to store multiplication.

TC: ↳ O(logn)

SC: O(1)

Break till 8:30am

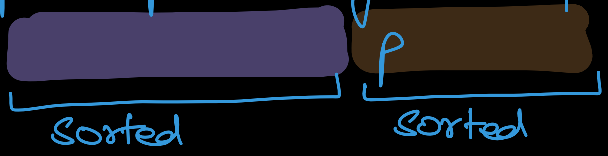
Qn: Search for element in a sorted but rotated array.

0	1	2	3	4	5	6
4	5	8	10	1	2	3
				↳ pivot		

BF idea: Linear Search   TC: O(n) , SC: O(1)

Binary Search:

(i) If pivot point is given = p idx



Apply BS in both

TC: O(logn)

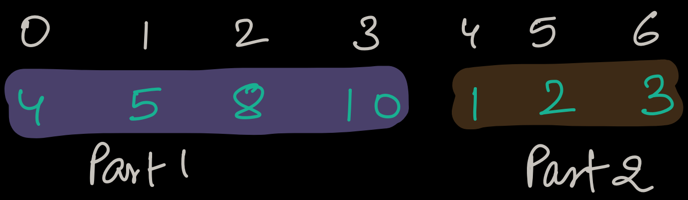
SC: O(1)

(ii) Pivot point is NOT given.

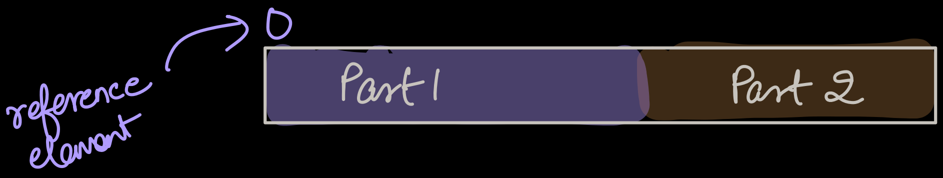
0	1	2	3	4	5	6
4	5	8	10	1	2	3

(a) Try to find pivot linearly.

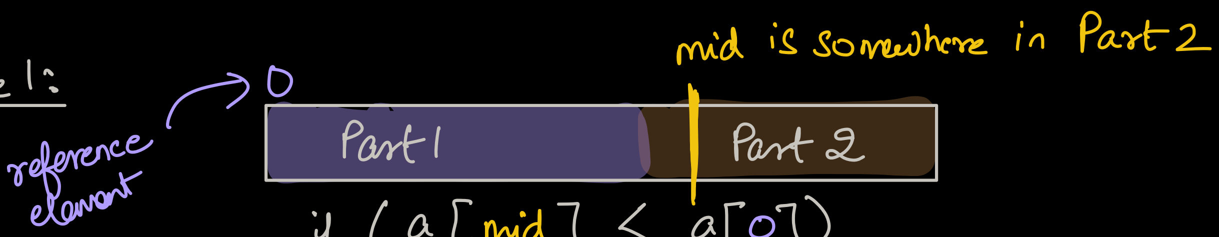
↳ TC: O(n)



obs: all elements of [Part 1 > Part 2]



Case 1:

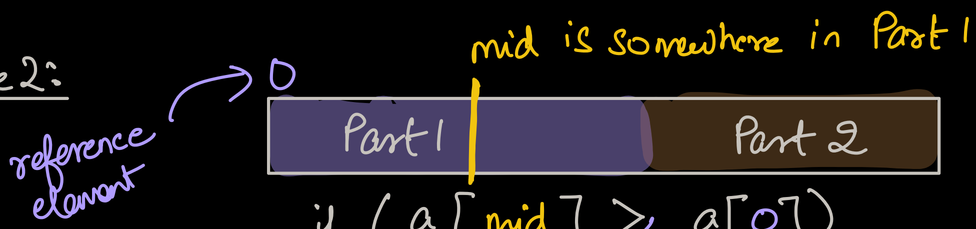


if (a[mid] < a[0])

∴ go left

+ update potential ans (pivot is in Part 2)

Case 2:



if (a[mid] $\geq$ a[0])

∴ go right.

	0	1	2	3	4	5	6	7	8	9	10	11
a[] =	10	20	30	1	2	3	4	5	6	7	8	9

Potential pivot = ~~5~~ 3

l	r	mid	a[mid] v/s a[0]
0	11	5	3 < 10 [Part 2] ∴ go left
0	4	2	30 > 10 [Part 1] ∴ go right
3	4	3	1 < 10 [Part 2] ∴ go left
3	2	STOP [l > r]	

Pivot found! Apply BS in left & right parts.

	0	1	2	3	4	5	6	7
Eg:	60	70	80	90	100	10	20	30

Potential pivot = 5

l	r	mid	a[mid] v/s a[0]
0	7	3	90 > 60 [Part 1] ∴ go right
4	7	5	10 < 60 [Part 2] ∴ go left.
4	4	4	100 > 60 [Part 1] ∴ go right
5	4	STOP! [l>r]	

Now apply BS in both parts.

Pseudocode :

```
int findPivot(int a[]) {
    l=0, r=n-1
    pivot = -∞/n
    while(l <= r) {
        m=(l+r)/2
        if (a[m] < a[0]) { // mid in Part 2, go left
            pivot = m
            r = m-1
        }
        else { // mid is in Part 1, go right
            l = m+1
        }
    }
    return pivot
}
```

To search for element:

(a) pivot = findPivot(a[])

(b) Apply BS in (0 pivot-1) (pivot n-1)

& find the element.

TC: $O(\log n) + O(\log n) + O(\log n)$

$= O(\log n)$

SC: $O(1)$

Follow Up Qn: Do this in a single BS

Do this in a single BS (w/o pivot)

[Strictly optional content; won't be necessarily asked in interviews]

Get mid ⟶ In which part is mid in ? In which part is target in ?

If they are in same part, apply BS

If they are in diff parts, move mid to target.

0	1	2	3	4	5	6	7	8	9	10	11
10	20	30	1	2	3	4	5	6	7	8	9

K = 20

Target v/s a[0]

20 > 10 ⇒ Target is in Part 1

l	r	mid	a[mid] v/s a[0]
0	11	5	3 < 10 [Part 2]
			→ Go left r = mid-1
0	4	2	30 ≥ 10 [Part 1] (Same parts)
			To find 20, go left. r = mid-1
0	1	0	10 ≥ 10 [Part 1] (Same parts)
			To find 20, go right l = mid+1
1	1	1	20 == 20 found!

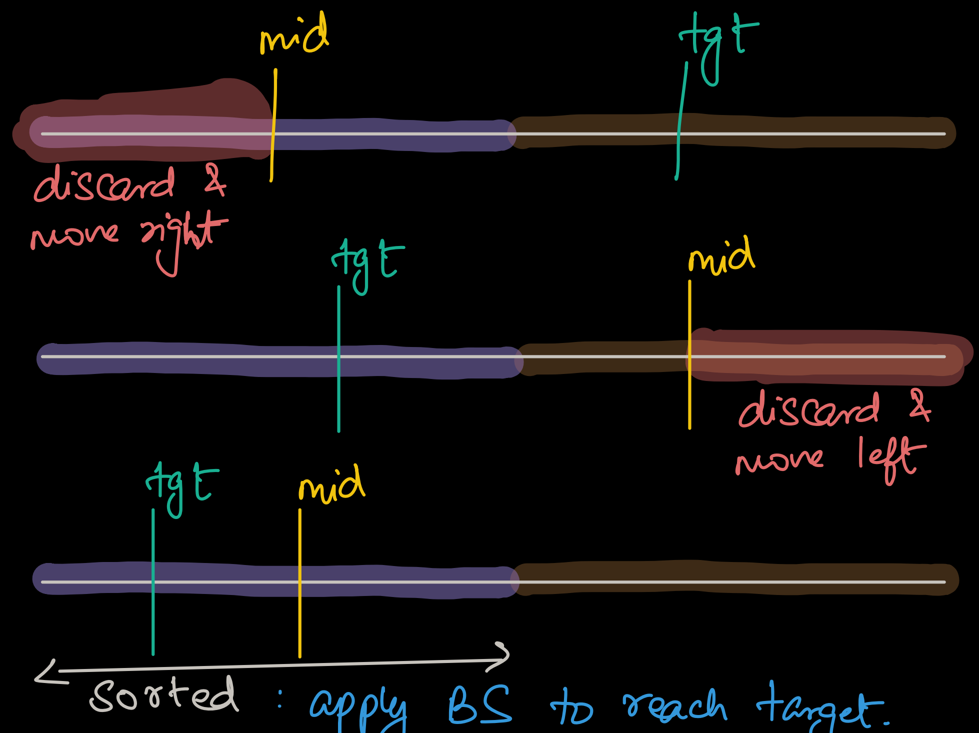
0	1	2	3	4	5	6
70	80	90	100	40	50	60

Eg:
K=60

Target v/s a[0]
60 < 70 ⇒ Target is in Part 2

l	r	mid	a[mid] v/s a[0]
0	6	3	100 > 70 [Part 1] ∴ go right l=m+1
4	6	5	50 < 70 [Part 2] Apply BS
			a[mid] = 50 < 60, go right l=m+1
6	6	6	60 == 60 found!

Qn: Why does this work?



sorted : apply BS to reach target.

Code : To do