Today's content

(i) Unique element

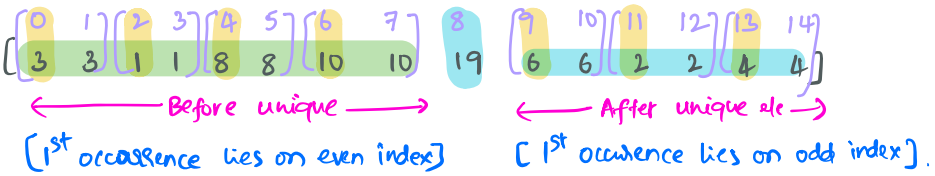
(ii) Sqrt of a number

(iii) Search in sorted and rotated array.

10. Every element occurs twice except for one element, find the unique element.

Note: Duplicates are adjacent to each other.

Ex:



idea: using XOR. } TC: O(N).
linear search } SC: O(1).

Idea 2:

Can we apply BS?

Case 1:

If ar[mid] is unique.

if (ar[mid-1] != ar[mid] && ar[mid+1] != ar[mid]),

Make mid land on 1st occurrence.

(i) if (ar[mid-1] == ar[mid])

{

mid = mid-1

}.

[8 8].

(ii) if (mid % 2 == 0)

{ // we are left side of unique ele

or

goto right

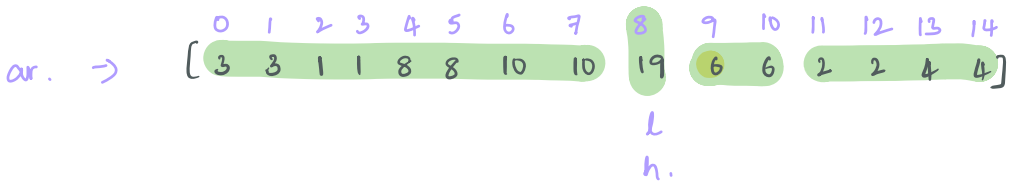
}

else

{ // we are on right side of unique ele

goto left

}.



l	h	m	isUnique	1st occurrence (ar[m] == ar[m-1])
0	14	7	×	m = m-1 → m = 6, m%2 = 0, goto right, l = m+2.
8	14	11	×	m = 11, m%2 = 1, goto left, h = m-1.
8	10	9	×	m = 9, m%2 = 1, goto left, h = m-1.
8	8	8	✓	return ar[m].

Code:

```
def uniqueElement(arr, n)
{
    if (n==1)
        return ar[0];          ] There's only one element.
    if (ar[0] != ar[1])
        return ar[0]
    if (ar[n-1] != ar[n-2])
        return ar[n-1].
    l = 1, h = n-2.
    while (l <= h).
        m = (l+h)/2;
        if (ar[m] != ar[m-1] && ar[m] != ar[m+1])
            return ar[m]  // unique element.
        if (ar[m] == ar[m-1])
```

[0 → l, m = 0, n-1] → h, m = n-1.

```
        m = m-1      // since ar[m] is second occurence.

    if ( m % 2 == 0)
        // unique element lies on right
        l = m+2

    else
        h = m-1
```

}

}

Doubts.

To avoid overflow.

$m = \frac{l+h}{2}$, $m = l + \frac{(h-l)}{2}$. , $h = 127$, $l = 126$.

inds. ⟶ 128.

l = integer.

h = integer.

$m = 128 + \frac{(128-128)}{2} = 128$.

$m = \frac{l+h}{2}$.

$= \frac{128+128}{2} = \frac{256}{2}$

$=$

$m = 126 + \frac{(127-126)}{2} = 126 + 0.5$

$\equiv 126$.

28. Given a +ve number N, find sqrt(N).

↳ Greatest i such that i*i ≤ N.

↓ integer.

Ex: Sqrt(25) = 5

Sqrt(30) = 5

Idea:

N = 30.

i	i*i ≤ N	ans.
1	✓	1
2	✓	2
3	✓	3
4	✓	4
5	✓	5
6	x	return 5.

Idea 2: Binary search

- Target : Sqrt(N).
- Search space : 1 --- N.

[Sqrt(N) will definitely lie in [1--N]].

Can I apply BS?

Case 1:

mid*mid = N,

return mid.

Case 2:

mid*mid < N

ans = mid

goto right.

Case 3:

mid*mid > N

goto left.

Tracing.

[1 _ _ _ _ 6 7 _ _ 9 _ _ _ 11 12 _ _ _ 24 25 _ _ 49 50 _ _ 100].

N = 100, ans = 1.

l	h	m	update answer.
1	100	50	50*50 > 100, goto left, h = m-1
1	49	25	25*25 > 100, goto left, h = m-1.
1	24	12	12*12 > 100, goto left, h = m-1.
1	11	6	6*6 < 100, ans = 6, goto right, l = m+1.
7	11	9	9*9 < 100, ans = 9, goto right, l = m+1.
10	11	10	10*10 = 100, return ans.

Hlw, do a dry for N = 99, ans = 9.

Code:

Hlw.

Break.

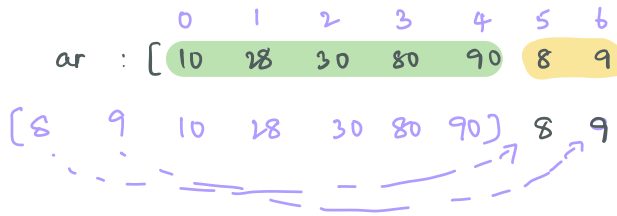
10:13:00

10:19:00

10:20:00

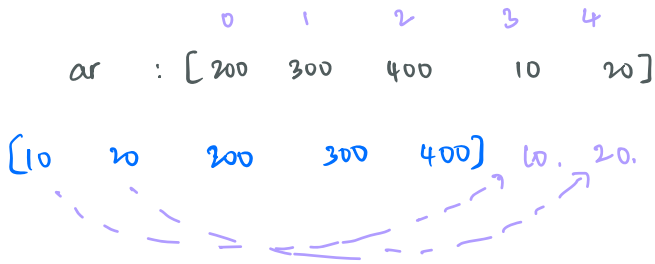
39. Given a **sorted** and **rotated array**, search for an element k.

Ex: ① ar : [10 28 30 80 90 8 9] , k = 30.



, o/p = 2.

② ar : [200 300 400 10 20] , k = 40.



, o/p = -1.

Idea: linear search for the element k.

TC : O(N), SC : O(1).

Idea 2:

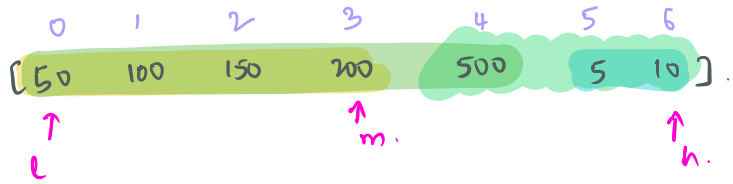
x = 50 ..

Claim: Put the **middle** at **any position**, one **half of the array** is **sorted.**

Question: 1) Which half is sorted?.

compare **mid** element with **ar[0]** (ar[l]).

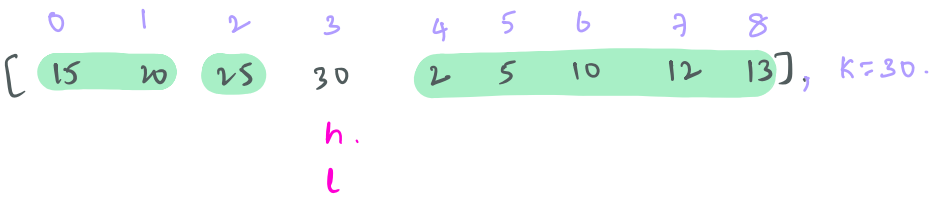
2) Can you figure out in which half the element 'k' lies in?



K = 50 → [200 > 50, 1st half is sorted, 50 lies in [50--200], Search left

K = 5 → [200 > 50, 1st half is sorted. 5 does lie in [50--200], Search right

Tracing



l	h	m = $(\frac{l+h}{2})$	Sorted Part	K lies in which part	goto	update
0	8	4	R	L	left	h = m-1
0	3	1	L	R	right	l = m+1.
2	3	2	L	R	right	l = m+1
3	3	3	(K = ar(mid)) => return mid.			

```
def SortedAndRotatedSearch ( ar, n, k)
{
    l = 0,  h = n-1
    while ( l <= h)
    {
        m = (l+h) / 2
        if (ar[m] == k)
            return m
        // left half is sorted.
        if (ar[m] >= ar[l])
            // check if k lies in left half.
            if ( k >= ar[l] && k <= ar[m]) // lies in first half.
                h = m-1
            else
                l = m+1
        else   // right half is sorted.
            // check if k lies in right half.
            if (k >= ar[m] && k <= ar[h]) // lies in right half.
                l = m+1
            else
                h = m-1
    }
    return -1
}
```