Binary Search : Searching words in a dictionary.

Divide into 2 parts.

	0	1	2	3	4	5
Eg	3	10	17	18	24	32

25

$N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow \cdots\cdots \rightarrow 1$

$\log_2 n$

Q1) Given sorted array with distinct elements, search for K. If K is not present return -1

(return index)

0	1	2	3	4	5	6	7	8	9
3	6	9	12	14	19	20	23	25	27

Idea : Use binary search $k = 20$

Case 1 : ar[mid] == k return mid

Case 2 : ar[mid] < k goto right

[mid] .... k

Case 3 : ar[mid] > k goto left

0	1	2	3	4	5	6	7	8	9
3	6	9	12	14	19	20	23	25	27

l	h	mid	K = 20
0	9	4	goto right
5	9	7	
5	6	5	
6	6	6	return 6

Code

```
int search ( int ar[], int N, int k ){
    l = 0        h = n-1
    while (l <= h) {
        mid = (l+h)/2
        if (ar[mid] == k)   return mid
        if (ar[mid] < k)    l = mid+1
        else                h = mid-1
    }
    return -1
}
```

TC: O(logn)

SC: O(1)

Q2 Sorted array, find index of first occurence of elem K.

0	1	2	3	4	5	6	7	8	9	10
-5	-5	-3	0	0	1	1	5	5	5	5

K = 5 → 7

K = 0 → 3

K = -3 → 2

Use Binary Search

Case I arr[mid] == k

k k mid k k k

ans = mid

go to = left

Case II arr[mid] < k

mid k

ans =

goto right

Case III arr[mid] > k

k mid

ans =

goto left

0	1	2	3	4	5	6	7	8	9	10
-5	-5	-3	0	0	1	1	5	5	5	5

$k = 5$

l	h	m	
0	10	5	right
6	10	8	ans = 8
6	7	6	right
7	7	7	ans = 7
7	6	STOP	

Code

```
int search ( int ar[], int N, int k )
{
    l = 0        h = n-1        ans = -1
    while ( l <= h )
    {
        mid = (l+h)/2
```

```
        if (ar[mid] == k)
            ans = mid
            h = mid - 1
        }
    else if (ar[mid] < k)      l = mid + 1
    else                       h = mid - 1
    }
    return ans
}
```

TC: $O(\log n)$

SC: $O(1)$

- For last occ

```
        if (ar[mid] == k)
            ans = mid
            l = mid + 1
        }
```

Q3 Find any local maxima

Peak elem , a[i] is maxima if $\geq$ both neighbours.

a[0] & a[n-1] have only 1 neighbour

Eg1 a[7] = [9, 6, 3, 14, 5, 7, 4] (0 1 2 3 4 5 6)

Eg2 a[7] = [24, 21, 19, 17, 15, 9, 7]

Brute: Iterate on each elem & check. TC: O(n)

$a[i] \geq a[i-1]$ $a[i] \geq a[i+1]$

Idea: For any elem a[mid]

1) If $\geq$ both neighbours $\Rightarrow$ return a[mid]

2) If left neighbour bigger $\Rightarrow$ goto left

6 8 10

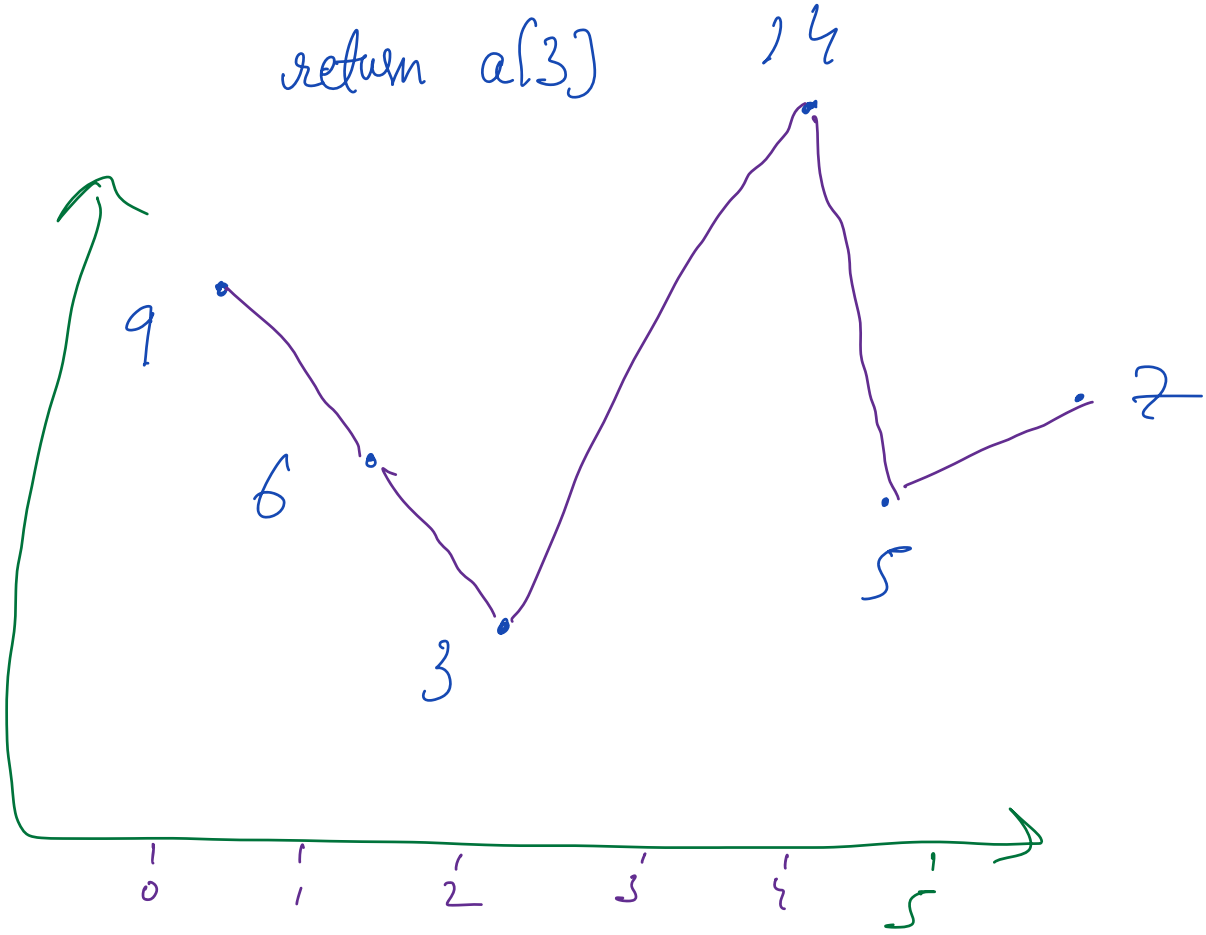
3) If right neighbour bigger ⇒ goto right

0 1 2 3 4 5

[9, 6, 3, 14, 5, 7]

l	h	m	
0	5	2	Can goto either side
3	5	4	lets go right l=mid+1
3	3	3	→ left h = mid-1

return a[3]



Code

```
int peak (int a[]) {
    l = 0          h = n-1
    while (l <= h) {
        mid = (l+h)/2
        if ( a[mid] >= a[mid-1]  &&
             a[mid] >= a[mid+1] )
            return a[mid]
        else if ( a[mid-1] > a[mid] ){
            h = mid - 1
        }
        else {
            l = mid + 1
        }
    }
}
```

mid != 0

mid != n-1

TC : $\log n$

SC : $O(1)$

Q4 Every element occurs twice except 1 which appears once. Find unique element. Note: Duplicates are adjacent

	0	1	2	3	4	5	6	7	8	9	10
Eg-	3	3	1	1	8	8	10	10	19	6	6

Idea : xor of all elem

TC: $O(n)$

Obs: Before unique elem → All 1st occurences at even idx
After unique elem → All 1st occurences at odd idx

Hence binary search

target ⇒ unique elem

what to search ⇒ indexes of the array

Case I ar[mid] = ⇒ unique return ar[mid]

Case II if (ar[mid] == ar[mid-1])

A A

mid-1 mid

new_mid --

if (new_mid is even)

goto right l = mid+1

else

goto left h = mid-1

0	1	2	3	4	5	6	7	8	9	10
3	3	1	1	8	8	10	10	19	6	6

l	h	m
0	10	5
6	10	8

return a[8]

19 17 17

0 2 1

0 0 0

Code

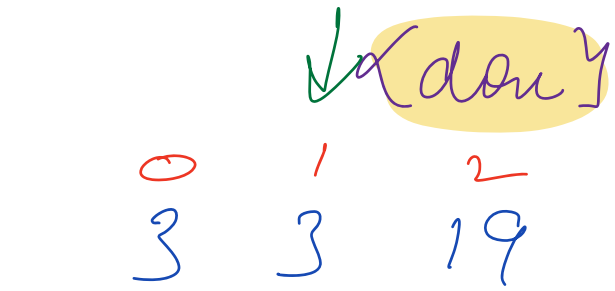
```
int findUnique ( int arr [], int N) {
    l = 0      h = n-1
    if (n == 1)    return a[0]
    if (ar[0] != ar[1])
        return a[0]
    if (ar[n-1] != ar[n-2])
        return a[n-1]
    while (l <= h) {
        m = (l+h)/2
        if ( a[mid] != a[mid-1] && a[mid] != a[mid+1] )
            return a[mid]

        new_mid = mid
        if (a[mid] == a[mid-1]) new_mid--
        if (new_mid % 2 == 0)  l = mid+1
        else    h = mid-1
    }
    return ar[mid]
}
```

19 19 17

handle for 0 & n-1

TC : O(log n)
SC : O(1)



0	1	2	3	4	5	6
3	3	19	8	8	1	1

0	6	3
0	2	1
2	2	