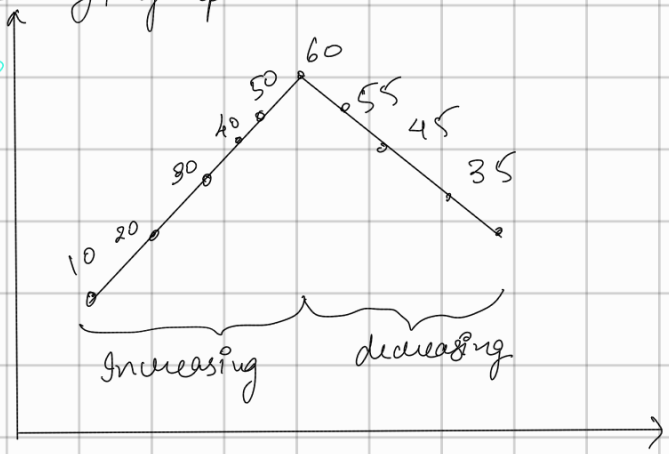
① Maximum element in Bitonic array
② Search element in Bitonic array
③ Peak element.
④ Unique element - Sorted array
⑤ Missing element of Arithmetic Progression

} Bitonic array

① Bitonic array / graph

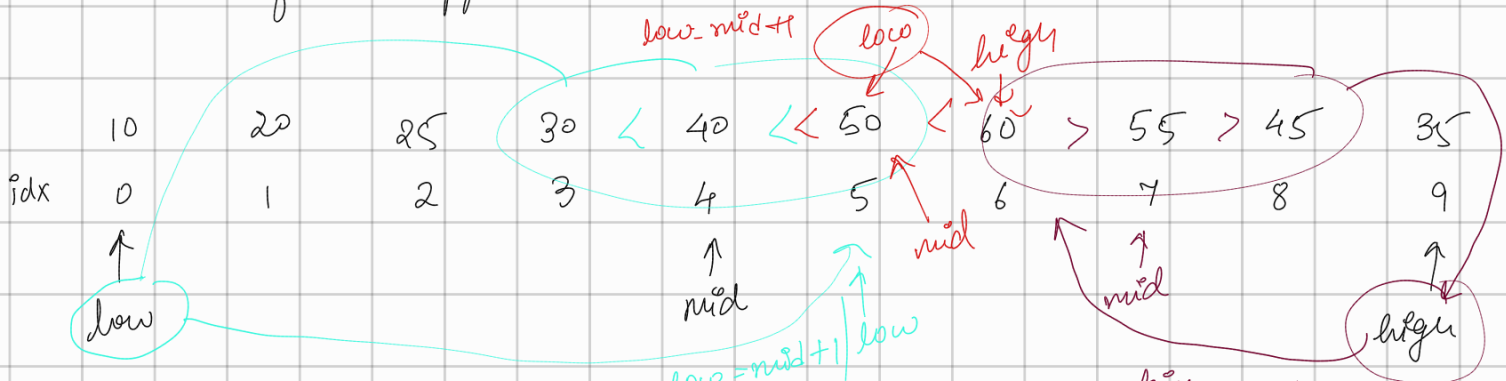
Never increases after decreasing

Bitonic array will either increase to a point then decrease



Repetition is allowed

Brute force approach → O(n) → linear search

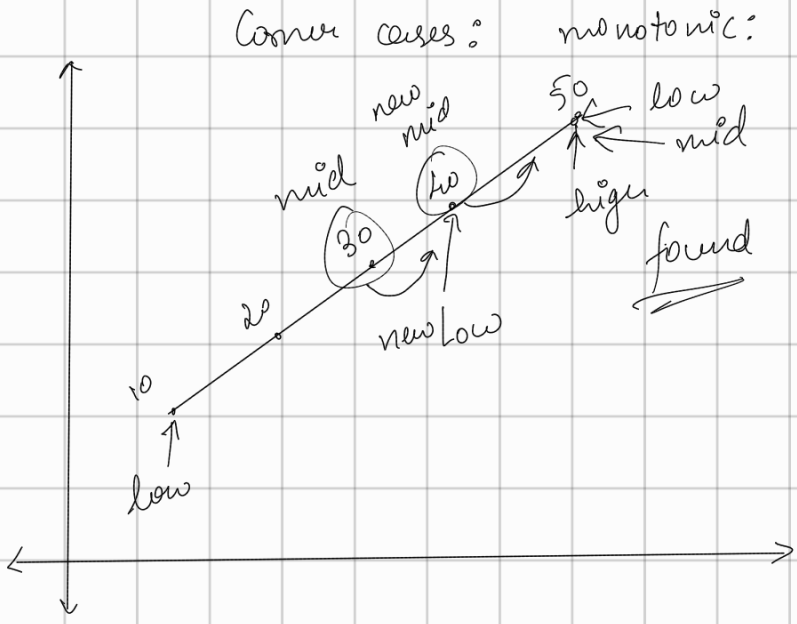


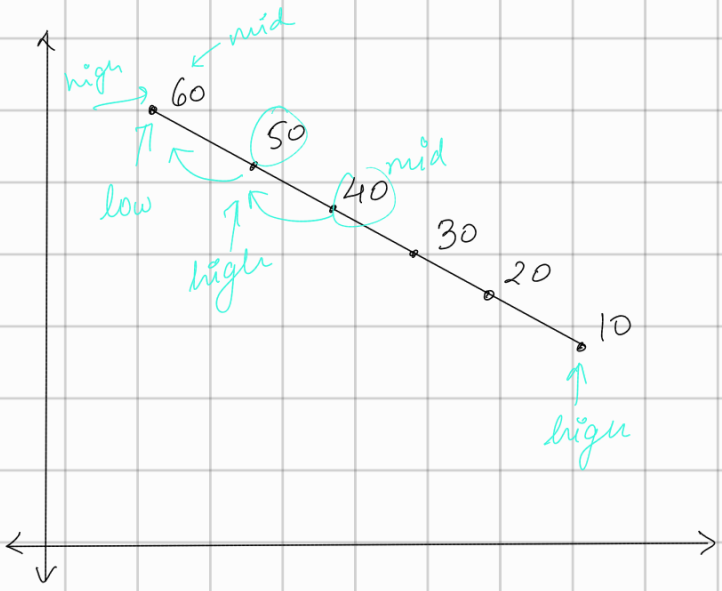
① mid = 0+(9-0)/2 = 4.

if (mid >= mid-1 && mid >= mid+1)
then peak found
else {
move towards higher
if (mid+1 > mid) → low = mid+1
else → high = mid+1

at idx = 6, low = high = mid & it is peak element.

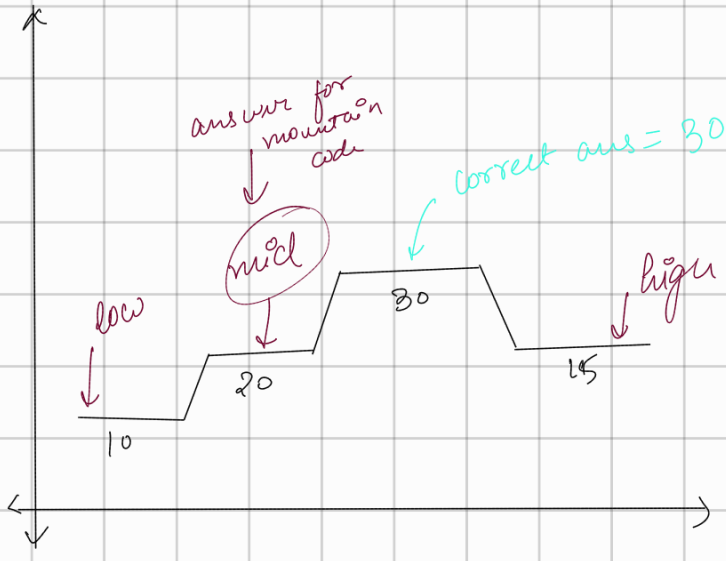
Corner cases: monotonic:





Code will fail in duplicate elements

Eg 10 10 10 20 20 20 30 30 30 15 15 15



or discard lower one of (high, low) linearly upto when low == high.

at that point, low = high = ans

O(N)

Variation needed, check for left & right of equal values. & compare using them

⇒ Approaches $O(n)$ from $O(\log_2 n)$

∴ CODE

```
int low = 0, high = n-1;
while (low <= high) {
    int mid = low + (high - low)/2;
    int val = (mid > 0) ? arr[mid-1] : INT_MIN;
```

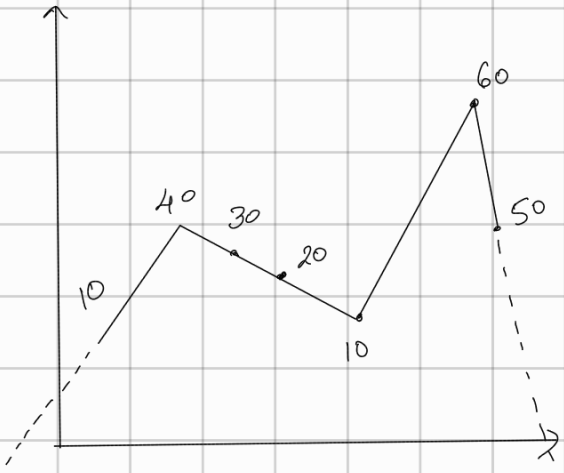
```
int lval = (mid < n-1) ? arr[mid+1] : INT_MIN;

if( arr[mid] > lval && arr[mid] < rval)
        return mid;
else if ( arr[mid] < lval)
        high = mid - 1;
else
        low = mid + 1;

}

return -1;
```

(2) Peak element.

★ Array is not sorted ★

★ No two adjacent elements are equal

★ nums[-1] = nums[n] = $-\infty$



0	1	2	3	4	5	6
10	40	30	20	10	60	50
↑	↑	↑	↑			↑
l	mid	high	mid			h

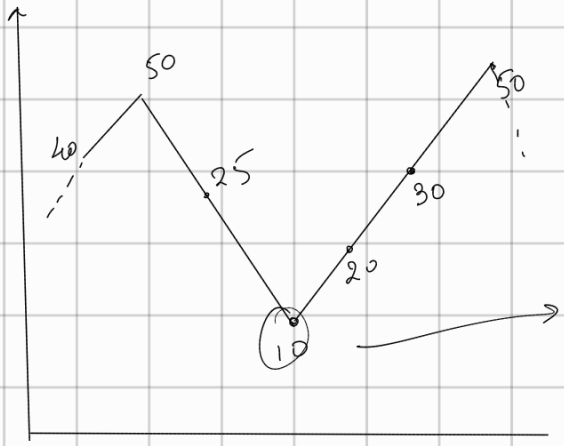
$10 < 40 > 30$

So local maxima is found.

$30 > 20 > 10$

So search on left half

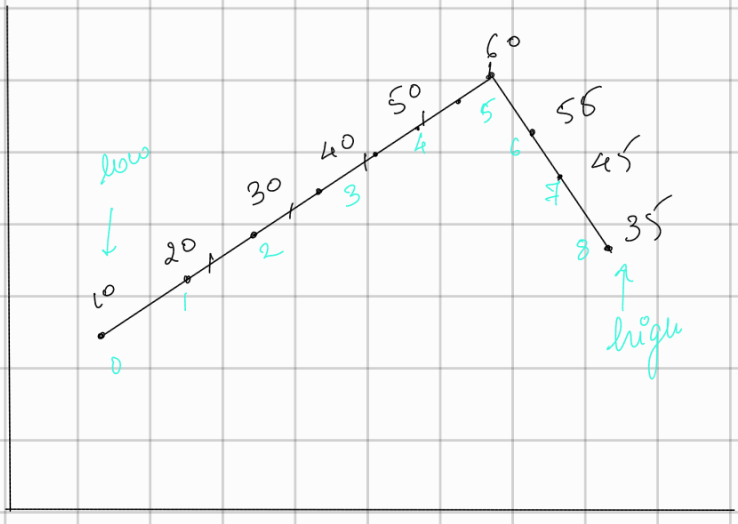
If mid element is valley



→ go either ways, both sides are guaranteed to have a peak.

③ Search in Bitonic array

Bitonic array: first increasing & then decreasing



Approach: find peak in $O(\log n)$

& apply 2 binary search in $O(\log n)$

→ $3 \times \log n \Rightarrow O(\log n)$

④ Single Elements in a Sorted Array

XOR approach ⇒ XOR all elements & return ans

because $x \wedge x = 0$

{ 1 1 2 3 3 4 4 8 8 }

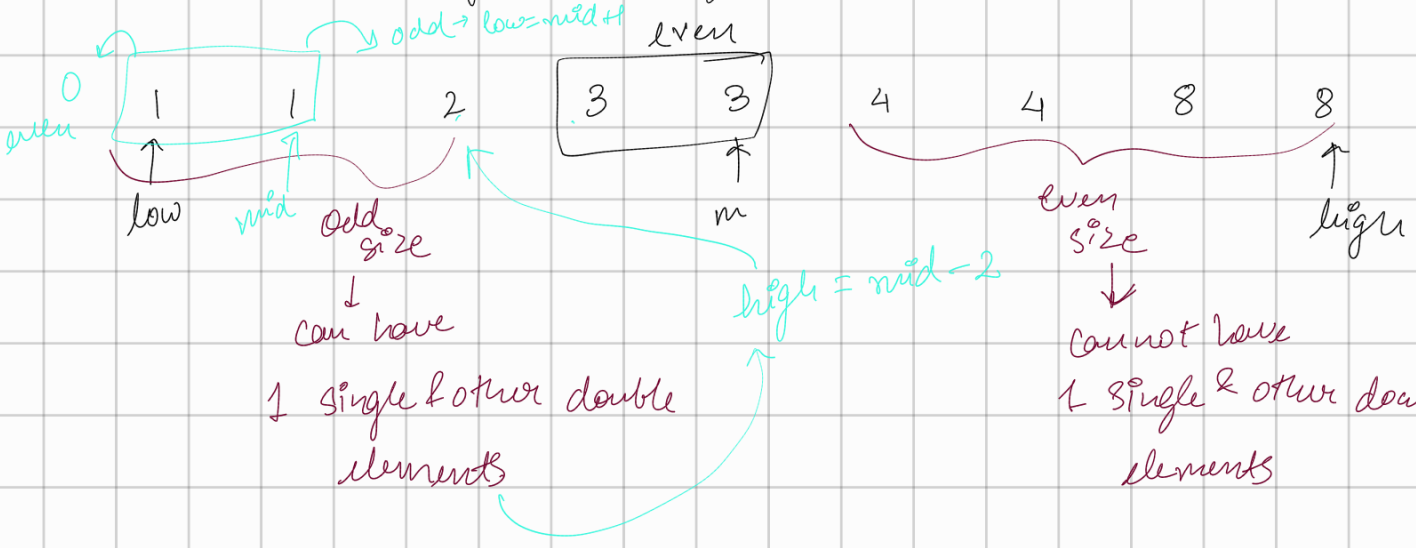
XOR → 1 0 2 1 2 6 2 10 2 → (2) ✓

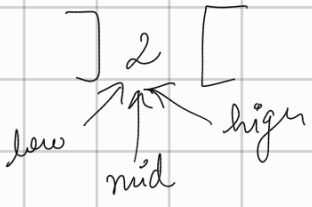
10
100

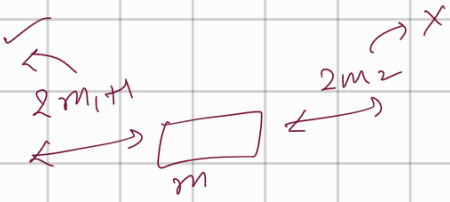
10
1000
8+2=10

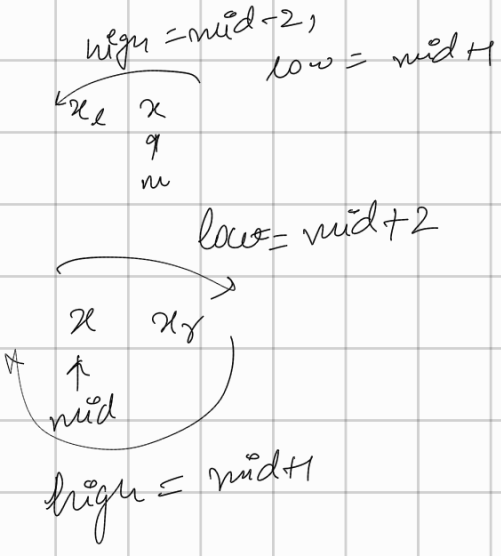
But this is O(n)

So use binary search, for O(logn)









xl x
↑ m
left range = m-l-1
right range = h-m

x xr
↑ m
left range = m-l
right range = h-m-1

(8) Missing element in AP