Searching 1 : Binary Search on Array

Search space → eg. Library

Target → eg. Book

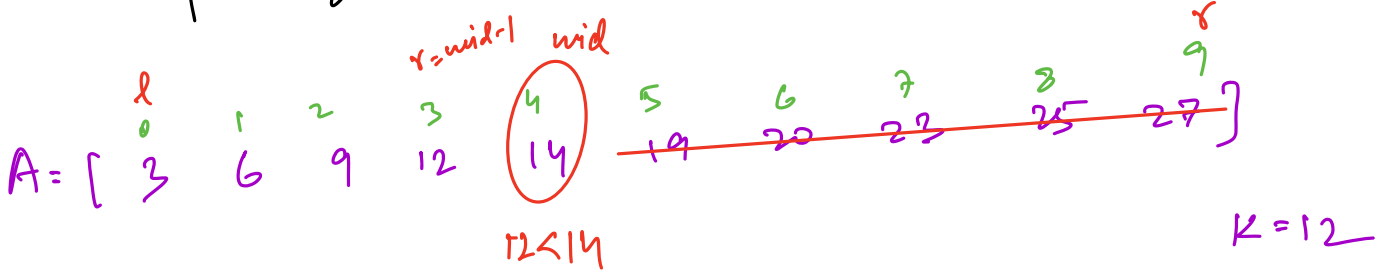
Condition → helps in finding target by reducing search space.

Binary Search ⇒ divide search space into 2 parts & keep neglecting one part based on condition

↳ organised data ⇒ use binary search

Question 1

Given a sorted array of distinct elements. find index of a given element K, if not present return -1.



Brute force →

```
for ( i=0 to n-1 ) {
    if (A[i] == K)
        return i
}
return -1
```

// linear search

TC = O(N)

SC = O(1)

Binary Search → 3 steps

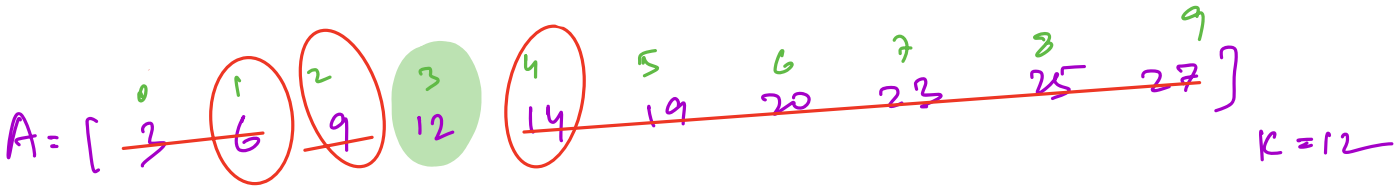
```
// 1. Define search space [0, n-1]   // index l to r
l=0 , r=n-1
while ( l <= r) {
// 2. check if middle element is the answer
    mid = (l+r)/2        // l + (r-l)/2
    if (A[mid] == K)
        return mid
// 3. Decide whether to go left or right
    if (K < A[mid])
        r = mid-1
    else
        l = mid+1
}
return -1
```

TC: $N \to N/2 \to N/4 \to \ldots \to 1$

iterations = $\log_2 N$

$TC = O(\log_2 n)$

$SC = O(1)$



l	r	mid
0	9	(0+9)/2 = 4
0	3	(0+3)/2 = 1
2	3	(2+3)/2 = 2
3	3	(3+3)/2 = 3

K<14 → go left

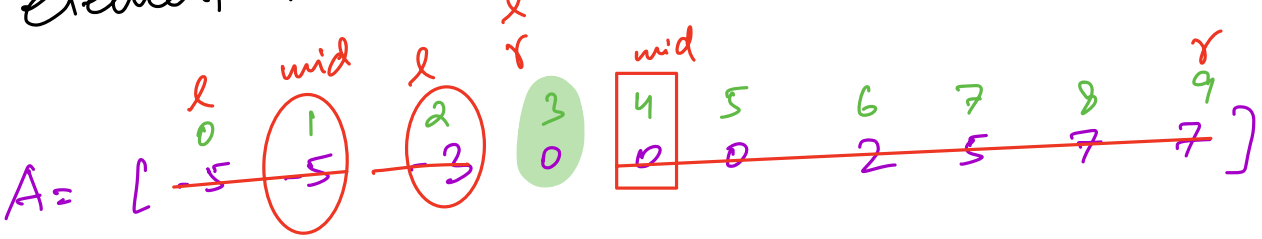
K>6 → go right

K>9 → go right

K = 12

Question 2

Given a sorted array. find first index of given element K.



K = 0 ans = 3

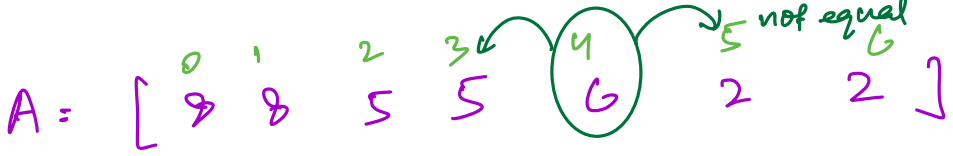
K = 1 ans = -1

```
// 1. Define search space [0, n-1]   // index l to r
    l = 0 , r = n-1
    while ( l <= r) {
// 2. check if middle element is the answer
        mid = (l+r)/2          // l + (r-l)/2
        if (A[mid] == K && (mid == 0 || A[mid-1] != K))
            return mid
// 3. Decide whether to go left or right
        if ( K <= A[mid])
            r = mid - 1
        else
            l = mid + 1
    }
    return -1
```

K < A[mid] → go left
K = A[mid] → go left
K > A[mid] → go right

Question 3

Given an array where every element occurs twice except for 1 element that appear once. Find the unique element. All equal pair of elements are together. (unsorted but organised)



Ideal : ans = $\oplus_i$ (A[i]) // xor of all elements

TC = O(N) SC = O(1)

// 1. Define search space [0, n-1) // index l to r

l = 0 , r = n-1

while (l <= r) {

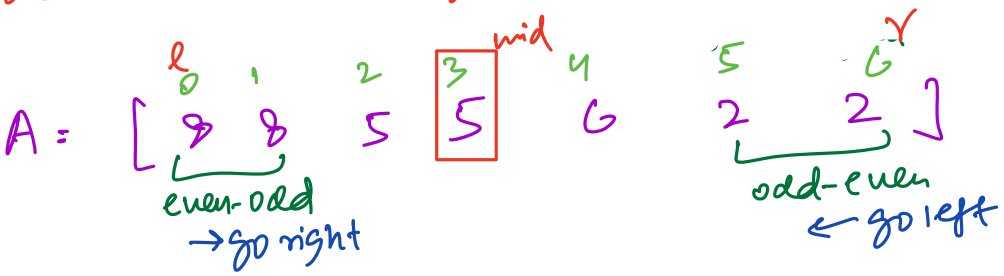
// 2. check if middle element is the answer

mid = (l+r)/2 // l + (r-l)/2

if ((mid == 0 || A[mid-1] != A[mid]) &&
(mid == N-1 || A[mid+1] != A[mid]))

return A[mid]

// 3. Decide whether to go left or right

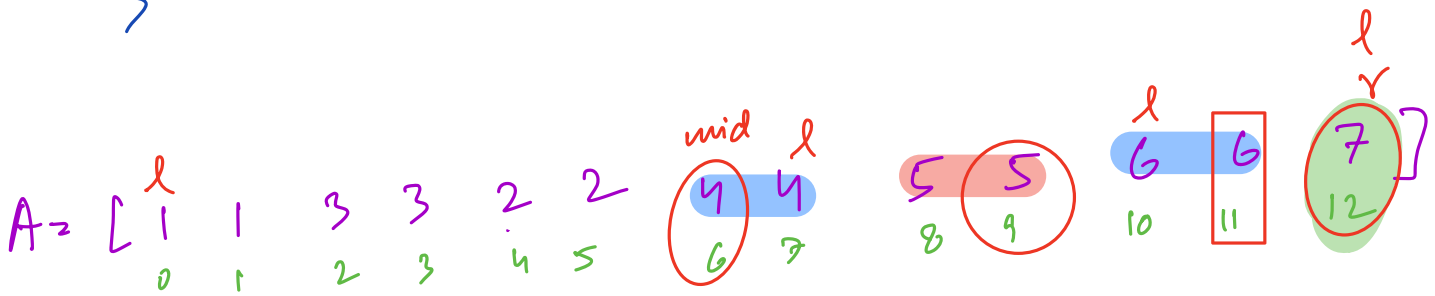


2 2 5 5 6 6 9 1 1
0 1 2 3 4 5 6 7 8

```
if (mid==0 || A[mid-1] != A[mid]) {  // (mid, mid+1) pair
    if (mid%2 == 0)   l = mid+1   // (even-odd)
                      or +2
    else              r = mid-1   // (odd-even)
}
else {   // (mid-1, mid)
    if (mid%2 == 0)   r = mid-1   // (odd-even)
                      or -2
    else              l = mid+1   // (even-odd)
}
}
```

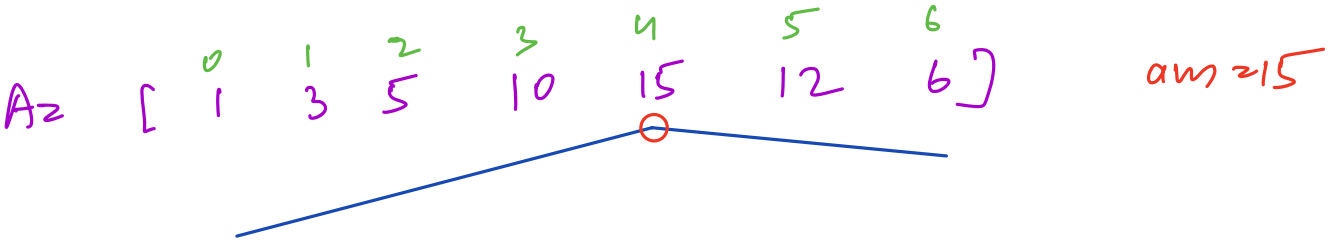
TC = O(logN)

SL = O(1)



Question 4

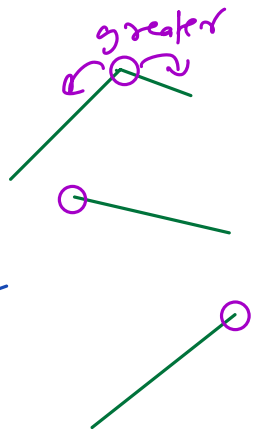
Given an increasing-decreasing array. Find max element. (Peak element)



// 1. Define search space [0, n-1] // index l to r

```
l = 0 , r = n-1
while ( l <= r) {
```

// 2. check if middle element is the answer




```
mid = (l+r)/2        // l + (r-l)/2
if ((mid == 0 || A[mid-1] < A[mid]) &&
    (mid == N-1 || A[mid+1] < A[mid]))
    return A[mid]
```

// 3. Decide whether to go left or right


```
if ( mid == 0 || A[mid-1] < A[mid]) {
    l = mid+1
else
    r = mid-1
}
```

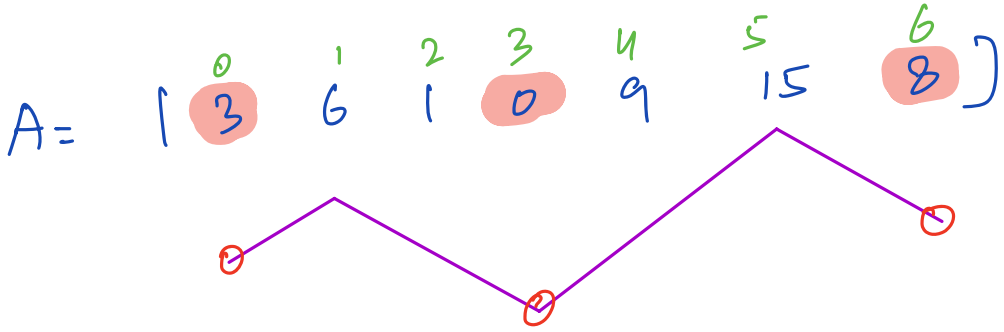
$TC = O(\log N)$

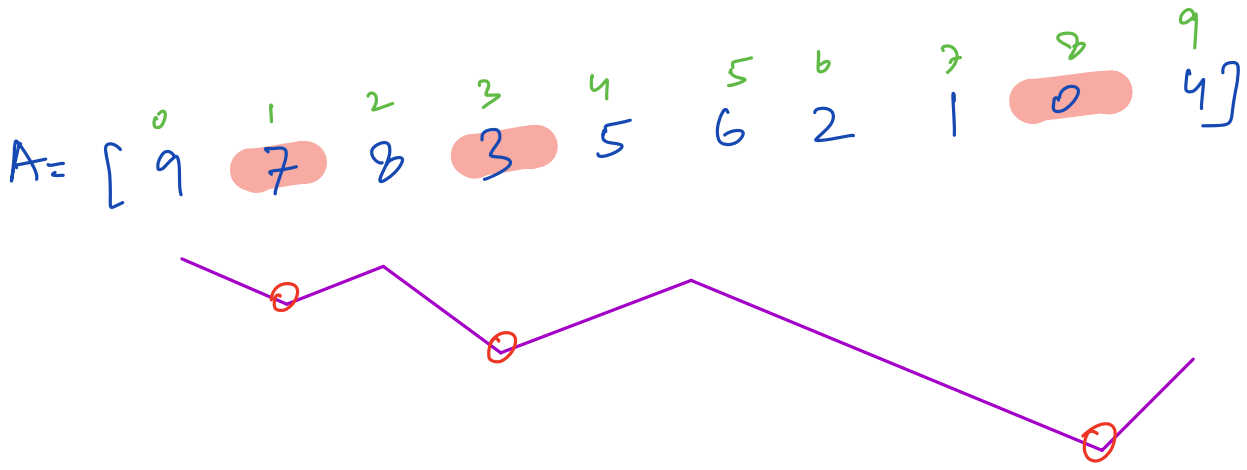
$SC = O(1)$

Question 5

Given an array of distinct elements, find any one local minima i.e

$$A[i-1] > A[i] < A[i+1]$$





Bruteforce : $\forall i$, check if $A[i]$ is local minima

$TC = O(N)$. $SC = O(1)$

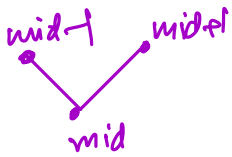
Search in $TC < O(N)$?

Binary Search

// 1. Define search space [0, n-1] // index l to r

```
l=0 , r=n-1
while (l<=r) {
```

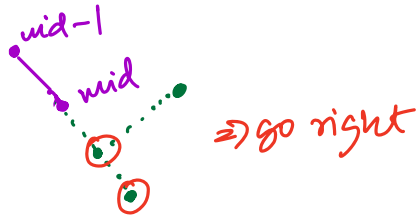
// 2. Check if middle element is the answer

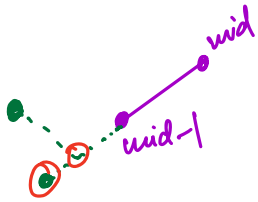



```
mid = (l+r)/2        // l + (r-l)/2
if ((mid==0 || A[mid-1] > A[mid]) &&
    (mid==N-1 || A[mid+1] > A[mid]))
    return A[mid]
```

// 3. Decide whether to go left or right

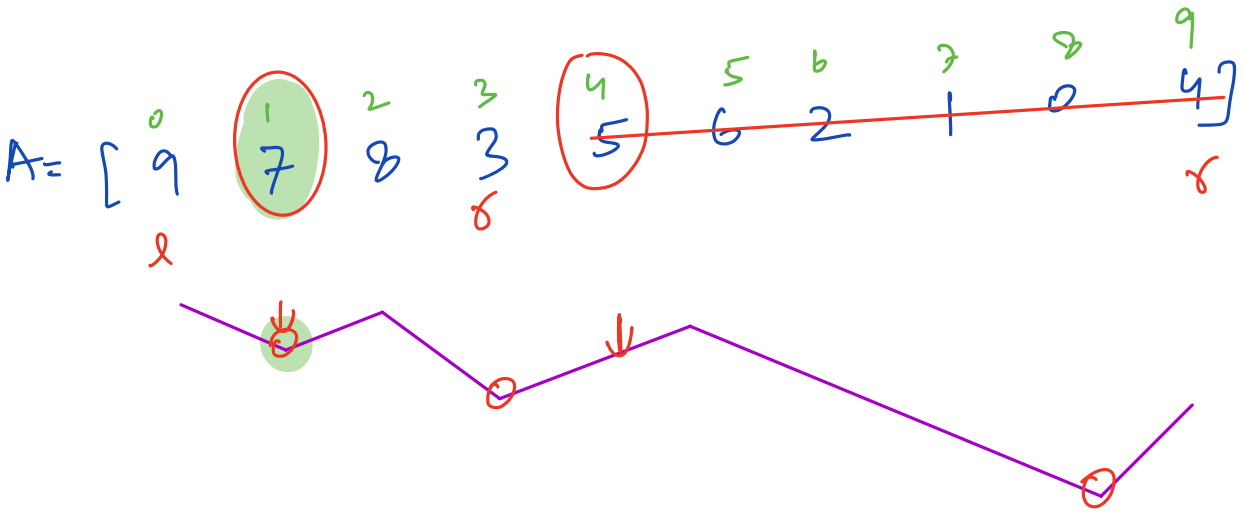
```
if (mid==0 || A[mid-1] > A[mid]) {
    l = mid+1
else
    r = mid-1
}
```

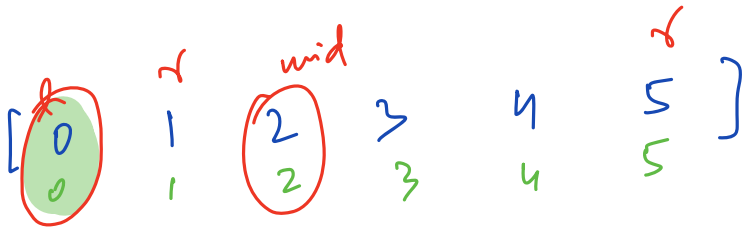




TC = O(log N)

SC = O(1)


A= [9 7 8 3 5 6 2 1 0 4]
0 1 2 3 4 5 6 7 8 9
l
r
r



[0 1 2 3 4 5]
l
r
mid
r
0 1 2 3 4 5