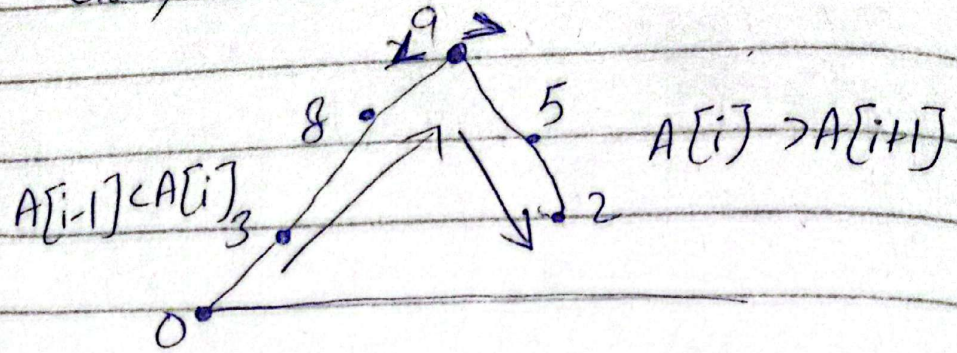
→ Peak Index in Mountain Array

arr = [0, 3, 8, (9), 5, 2]

Increasing order → Peak Element ← Decreasing order



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

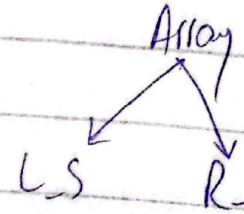
① Linear Search is the Brute Force Approach. $O(n)$ → T.C. —

② Optimal Approach is Binary Search

Index = 0, n-1 ≠ Peak Element

① $mid = s + \left(\frac{e-s}{2}\right)$

② A[mid-1] < A[mid] > A[mid+1]
↳ return mid

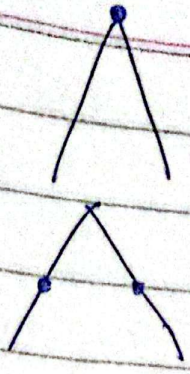


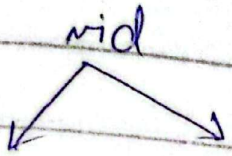
One will be discarded then the T.C will be $O(\log n)$

Case : 1

mid → peak

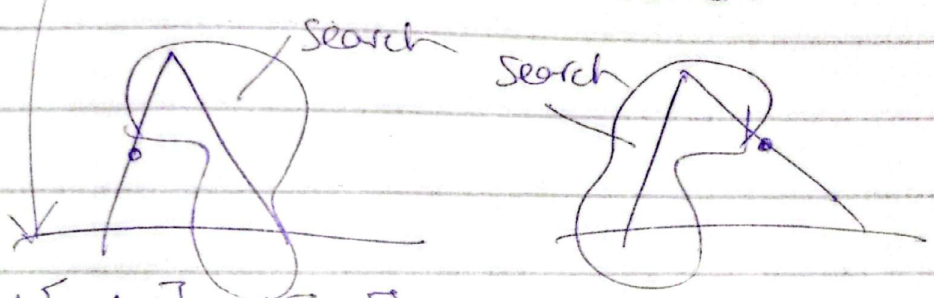
Case : 2





left (Inc) → Right Search

Right (Dec) → left Search



A[mid-1] < A[mid]

st = mid+1

end = mid-1

Pseudocode :

```
while(st<=end){
mid = st(e-s/2)
if(A[mid-I]<A[mid]>A[mid+1]) -> return mid
if(A[mid-1]<A[mid]) //Increment Right
        st = mid+1
else  dec -> left
        end = mid-1
```

If the mid exist like

mid / 0 there will be no 0-1 / mid

Similarly

mid / n-1 there will be no overflow (n)

① 0, n-1 ✗

② Unnecessary checks ✗

So the

st=1 , end = n-1