Todays content

1. Peak Element
2. Find Unique element

Given a unsorted arr[] with all distinct elements return any one local maxima

localmaxima: An element is said to be local maxima, if > than it's adjacent elements [immediate left & right]

ar[i] is local maxima if = ar[i-1] < ar[i] > ar[i+1]

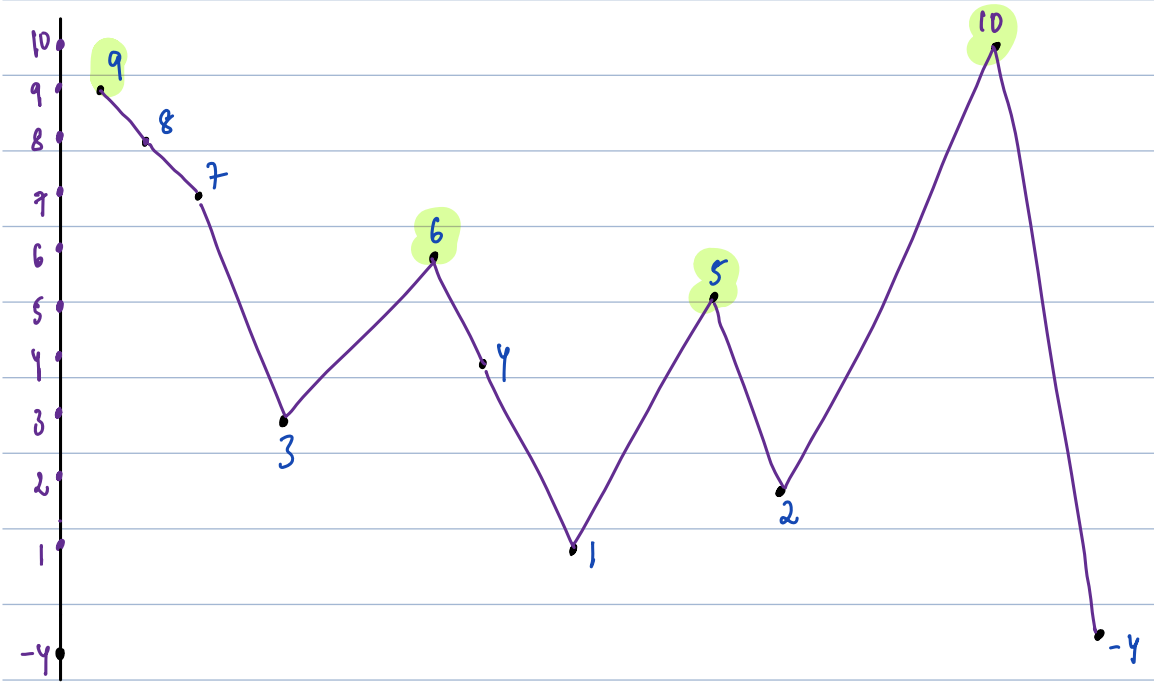
ar[0] is local maxima if = ar[0] > ar[1]

ar[N-1] is local maxima if = ar[N-2] < ar[N-1]

	0	1	2	3	4	5	6	7	8	9	10
Ex: ar[11] = {	9	8	7	3	6	4	1	5	2	10	-4 }

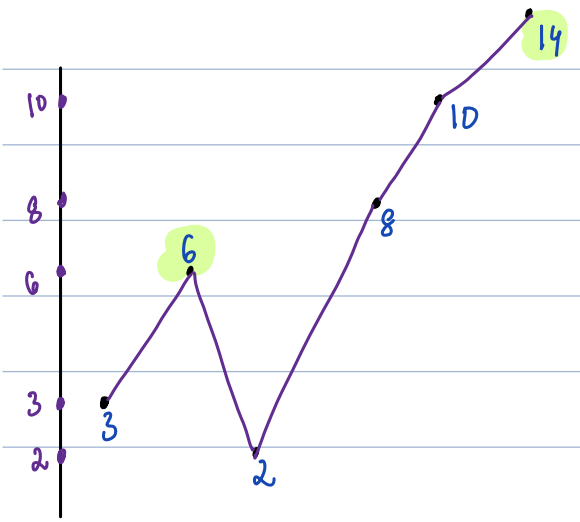
localmaxima: 9 6 5 10

Note: Return any one.



Ex: ar[6] = {3 6 2 8 10 14} local Maxima = 6 14

Note: Return any if them.



obs1: Atleast 1 local maxima exists, because max ele will be > its adjacent elements.

#Idea: Iterate & return max of arr[]

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

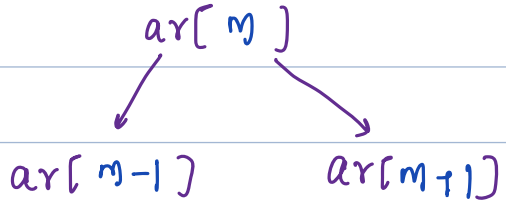
#Ideal: Iterate on arr[]: TC: O(N) SC: O(1)

For every ar[i], check if it's local maxima by comparing with adjacent elements, if it is return ar[i];

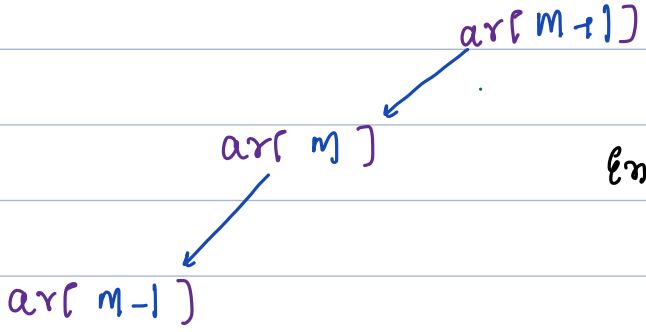
Idea:

Target = Any one local maxima Search Space = In arr[]

Case-1 if [arr[m] > arr[m-1] && arr[m] > arr[m+1]] { return arr[m] }



Case-2 if (arr[m] < arr[m+1]) { #goto right, because it's a guarantee we can find atleast 1 local maxima in right }



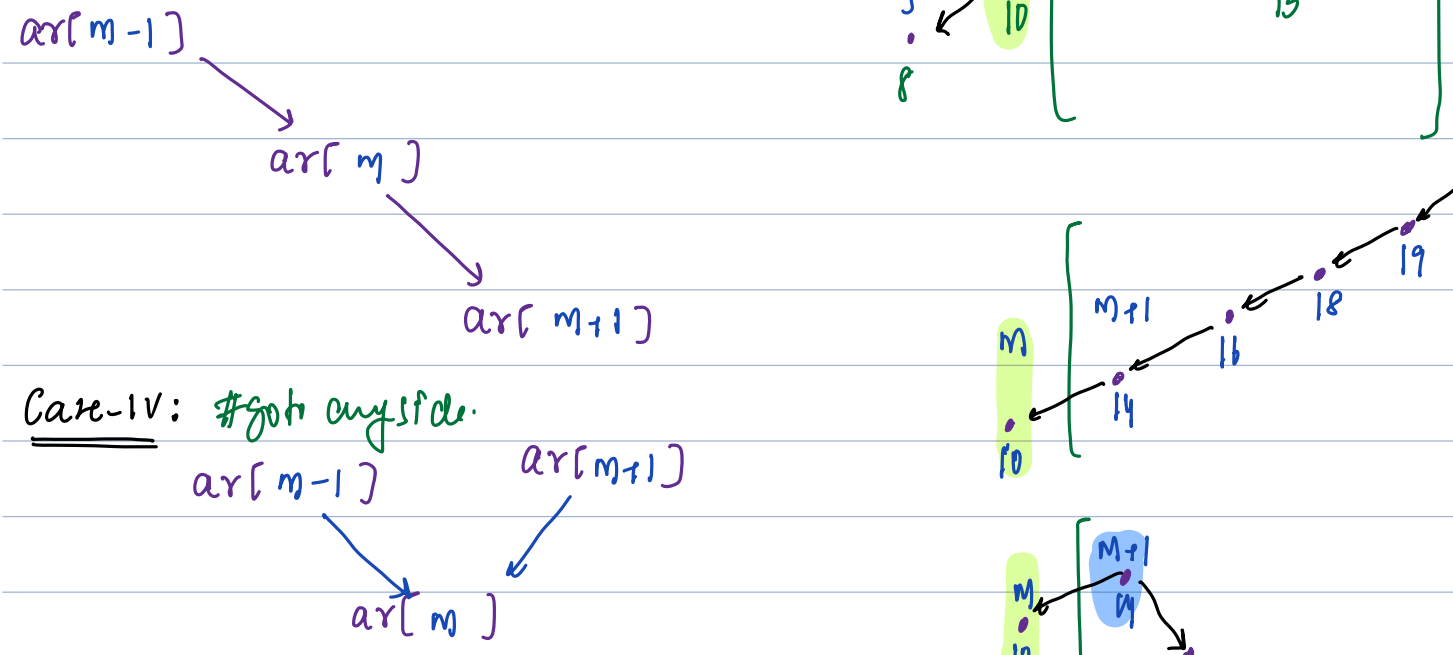
Ex:

	0	1	2	3	4	5	6	7	8
	6	4	12	8	10	14	16	13	19
	l								h

m-1 m m+1

l	h	m
0	8	4

Case-3 #goto left:
guarantee local maxima exists



Case-IV: #goto anyside.

#Con: After landing at mid:

- Goto side where data increases, it's a guarantee local maxima exists in that side.

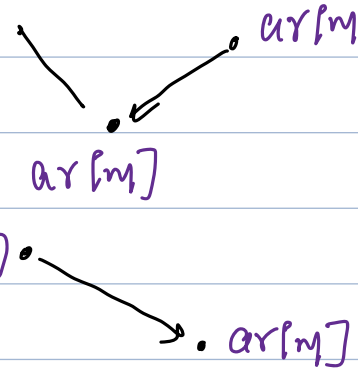
```
int localmaximaC vector<int> &ar){ TC: O(log N/2)  SC: O(1)
    int N = ar.size();
    if(N==1){ return ar[0];}
    if(ar[0] > ar[1]){ return ar[0]}
    if(ar[N-1] > ar[N-2]){ return ar[N-1]}
    int l=0, h= N-1;

    while(l<=h){
        int m = (l+h)/2;
        if( ar[m] > ar[m-1] && ar[m] > ar[m+1]){ return ar[m]}
        else if( ar[m] < ar[m+1]){
            l=m+1; #goto right;
        }
        else{
            h=m-1; #goto left;
        }
    }
}
```

Issues:

m=0: ar[0] > ar[-1]

m=N-1: ar[N-1] > ar[N-2] && ar[N-1] > ar[N]



#Note: When comparing adjacent elements, be careful about corner elements

Q: Google

Given a unsorted arr[] return anyone local maxima

localmaxima: An element is said to be local maxima, if >=than it's adjacent elements [immediate left & right]

arr[i] >= arr[i-1] && arr[i] >= arr[i+1]

If i==0: arr[0] is local maxima ⇒ arr[0] >= arr[1]

If i==N-1: arr[N-1] is local maxima ⇒ arr[N-1] >= arr[N-2]

```
int localmaximaRepeatim(vector<int>& arr){

}
```

18. Every element occurs twice, except for 1, find a unique element

Note: Duplicates are adjacent to Each other

Eg1:

	0	1	2	3	4	5	6	7	8	9	10
ar[] = {	6	6	2	2	7	9	9	4	4	10	10 }

ans = 7

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ar[] = {	3	3	1	1	8	8	10	10	19	6	6	2	2	4	4 }

ans = 19

#Idea1: Calculate xor of all arr[] elements & return unique element

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

#Idea2:

Target: Unique element Searchspace: In arr[]

	0	1	2	3	4	5	6	7	8	9	10	11	11
ar[] = {	12	12	6	6	21	21	7	9	9	4	4	10	10 }

If m is here: go right

On left:

1st occurence of ele: even

left of unique go to right

If m is here: goto left

On right

1st occurence of ele: odd

right of unique: goto left

#Note: If a posible that mid can land in 2nd occurence

if mid land in 2nd occurence bring to 1st occurence

```
if(arr[m]==arr[m-1]){
    m=m-1; # bring to 1st occurence
}
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
ar[ ] =	3	3	1	1	8	8	10	10	14	6	6	2	2	4	4
							m, 1st	m+1	lh	m, 3rd		m, 2nd			

l	h	m	ar[m-1]==ar[m]	left or Right
0	14	7	2nd occ m=m-1;	m%2==0: left: goto right, l=m+2;
8	14	11	1st occ	; m%2==0: right: goto left, h=m-1;
8	10	9	1st occ	; m%2==0: right; goto left, h=m-1;
8	8	8	ar[8] != ar[7] && ar[8] != ar[9] return ar[8]	

```
int unique(vector<int> &ar){ TC: O(log N/2)  SC: O(1)
    int N=ar.size();

    int l=0, h=N-1;
    while(l<=h){
        int m=(l+h)/2;
        if(m==0 || m==N-1){return ar[m];} problem
        if(ar[m]!=ar[m-1] && ar[m]!=ar[m+1]){return ar[m];}
        if(ar[m-1]==ar[m]){ #update m to 1st occurence.
            m=m-1;
        }

        if(m%2==0){ #1st occurs even: left go right
            l=m+2
        }
        else{ # 1st occurence odd right: go left
            h=m-1;
        }
    }
}
```

Note: if m came to start or last index, in that case, that itself is ans, in this case above