**Constraints:**

1 <= 'n' <= 10^5

1 <= 'arr[i]' <= 10^5

'arr[i]' != 'arr[i + 1]' for all 'i' in range 0 <= 'i' < 'n' - 1

Brute:

def findPeakElement(arr: [int]) -> int:

    if arr[len(arr)-1]>arr[len(arr)-2]:

        return len(arr)-1

    for i in range(1,len(arr)-2):

        if arr[i-1]<arr[i] and arr[i]>arr[i+1]:

            return i

    return -1

OPTIMAL:

 def findPeakElement(self, arr):

        if len(arr)==1:

            return 0

        if arr[len(arr)-1]>=arr[len(arr)-2]:

            return len(arr)-1

        if arr[0]>=arr[1]:

            return 0

        s=1

        e=len(arr)-2

        while(s<=e):

            mid=(s+e)//2

            if arr[mid]>arr[mid-1] and arr[mid]>arr[mid+1]:

                return mid

            elif arr[mid]<arr[mid-1]:#Peak will be towards left side

                e=mid-1

            elif arr[mid+1]>arr[mid]:#Peak will be towards right side

                s=mid+1

        return -1

Other Optimal:

int findPeakElement(vector<int> &a) {

int max\_n=INT\_MIN;

int n=a.size();

int ans=-1;

int low=0;

int high=n-1;

while(low<=high)

{

int mid=low+(high-low)/2;

if(a[low]<=a[mid])

{

if(a[mid]>max\_n)

{

max\_n=a[mid];

ans=mid;

}

low=mid+1;

}

else

{

if(a[mid]>max\_n)

{

max\_n=a[mid];

ans=mid;

}

high=mid-1;

}

}

return ans;

}