03/02/2026
Tuesday

How many times array has been rotated

ex:- a = [4,5,1,2,3]
ans = 2

Simple logic is we follow same as minimum procedure but here we store minimum element index and return that index

0	1	2	3	4
4	5	1	2	3

min is 1 it's index is 2
so the array is rotated two times

1st rotate → 5 1 2 3 4

2nd rotate → 4 5 1 2 3

Another example

0	1	2	3	4	5	6
7	8	11	2	3	5	6

Here min is 2 and it's index is 3
so 3 times rotated

low, mid, high

first we take index = -1 and ans for minimum as max value
we check which one is sorted here

7<=2 a[low]<=a[mid] false

2<=6 true

we go right side

index = mid
= 3

ans = (2, max)
= 2

if (a[mid] < ans)
then only we are
assigning them

then we will go to left side

high = mid-1
= 2

low=0 high=2
mid=1

7<=8 true

ans = (2,7) Here we are checking
= 2 whether a[low] < ans(min)
index = 0 7 2

false right

→ so we do not assign index=0

we will go right side

low = mid+1 high=2
= 2
mid=2

8<=8 true 11<=11

11<2 false

so we go right side

low = mid+1

low=3 high=2

search space not exist it fails

and we can return index

```
public int noofrotations(int[] a, int n)
{
    int low=0;
    int high=n-1;
    int ans= Integer.MAX_VALUE;
    int index = -1;
    while (low<=high)
    {
        int mid= low+(high-low)/2;
        if (a[low]<= a[mid])
        {
            if (a[low] < ans)
            {
                ans = a[low];
                index = low;
            }
            low=mid+1;
        }
        else{
            if (a[mid] <ans)
            {
                ans = a[mid];
                index=mid;
            }
            high=mid-1;
        }
    }
    return index;
}
```

Find single element in Sorted Array

ex:-

a = [1, 1, 2, 2, 3, 3, 4, 5, 5, 6, 6]

Here every element appears twice except one element

logic is

```
    0  1  2  3  4  5  6  7  8
a= [1, 1, 2, 2, 3, 4, 4, 5, 5]
```

see in the array before single element they is indexes are in the order

Like (odd, even) odd (odd, even) odd

```
0  1     2  3  |  4  |  5  6     7  8
1, 1     2, 2  |  3  |  4, 4     5, 5
even odd even odd |  |  odd even  odd even
```

see Left side elements have even, odd and right side have odd even

first we will check base conditions that solves edge cases

if n==1 we return first element

```
if a[0] != a[1]
        return a[0]

if a[n-1] != a[n-2]
        return a[n-1]
```

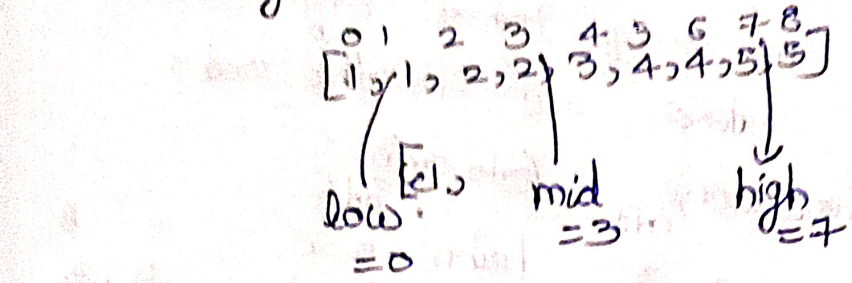
[1, 1, 2, 2, 3, 4, 4, 5, 5]

Here we are shrinking it because we need to check left and right for a element

For first and last we can't get their beside elements right so

low = 1

high = n-2



we will check if it's left is equal to that ele and also check it's right element is equal to current element if not we return current element

a[2] a[3] a[4]
2 2 3

2 != 2 && 2 != 3
F T

So it is false

So it is not the element

Now we will check we are in which side of single element

by Conditions

mid=2

mid is even

we check a[3]==a[2] or a[mid-1]==a[mid]

2==2

true

mid is odd

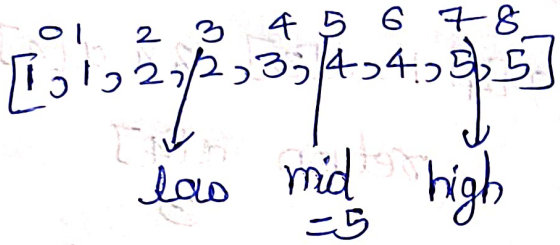
then go right side

So we are in left part

we go right side

low = mid+1

low = 3



it fails and check which part to remove

if mid is odd

a[4] == a[5]

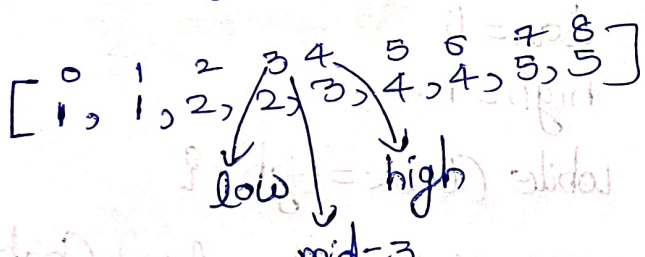
3==4

false

So we go left side

high = mid-1

high = 4



2!=2 && 2!=3 false

mid is odd I will check before element is equal to mid ele

a[mid] == a[mid-1]

a[3] == a[2]

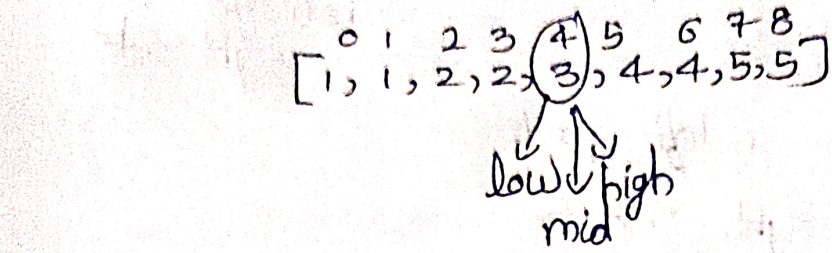
2==2 true

So we go right side

low = mid+1

low = 4 high = 4

mid = 4



we check

a[mid] != a[mid-1] && a[mid] != a[mid+1]

a[4] != a[3] && a[4] != a[5]

return a[4]

3

```
public int find(int[] a, int n) {
    if(n==1)
        return a[0];
    if(a[0] != a[1])
        return a[0];
    if(a[n-1] != a[n-2])
        return a[n-1];
    low=1;
    high=n-2;
    while(low<=high){
        int mid= low+(high-low)/2;
        if(a[mid] != a[mid-1] &&
            a[mid] != a[mid+1])
            return a[mid];
```

Now check which part to remove based on even odd

```
if (mid%2==0 && a[mid]==a[mid+1]
        || mid%2!=0 && a[mid]==a[mid-1])
    return low=mid+1;
else
    high= mid-1;
}
return -1;
}
```

Find the peak element in the array and take outside indexes as -infinity

peak element means it should be greater than next element and also greater than previous element

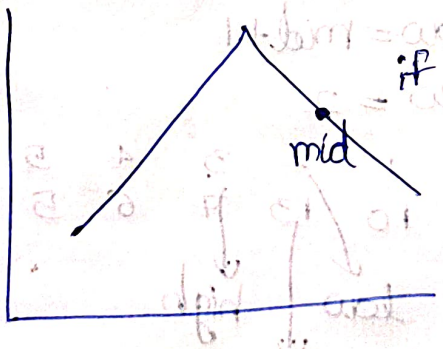
ex:- $-\infty$ [10,9,8,7,6,5] $-\infty$

Here 10 is the peek element

$-\infty$ [1,2,3,4,5] $-\infty$

Here 5 is the peek element

logic is



if mid is here we check whether a[mid] > a[mid-1] then peek is on right side

but here the condition fails so we go left side

ex:-

	0	1	2	3	4	5	6	7	8	9
a=	1	10	13	7	6	5	4	2	1	0

low, mid, high

first we check

6>7 && 6>5

false

we check which side to go

6>7 false

so we go to left side

high = mid-1

high = 3

	0	1	2	3	4	5	6	7	8	9
a=	1	10	13	7	6	5	4	2	1	0

low, mid, high

first we check

10>1 && 10>13

false

we check which side to go

10>1 true

So we go right side

low = mid+1

low = 2

	0	1	2	3	4	5	6	7	8	9
a=	1	10	13	7	6	5	4	2	1	0

low, high, mid=2

first check

13>10 && 13>7

true so

we return element or index based on question.

```
public int findpeak (int[] a, int n) {

    if (n==1)
        return a[0];

    if (a[0]>a[1])
        return a[0];

    if (a[n-1]>a[n-2])
        return a[n-1];

    low=1;
    high=n-2;
    while (low<=high){
      int mid= low+ (high-low)/2;

    if (a[mid]>a[mid+1] && a[mid]>a[mid-1])
                return mid;

     if (a[mid]>a[mid-1])
                low=mid+1;

     else
                high=mid-1;

    }
    return -1;

}
```

binary Search on Answers

Find square root of a number in logn

given n = 28

$\sqrt{28}$ 5... but question take floor value so it is 5

for this we check binary search on answers

1x1 <= 28
2x2 <= 28
3x3 <= 28
4x4 <= 28
5x5 <= 28
6x6 <= 28 false

answer must be between 1 to n only

```
public int solve(int n){
    int low=1;
    int high=n;
    int ans=1;
    while(low<=high){
        int mid= low+(high-low)/2;
        if(mid*mid<=n){
            ans=mid;
            low=mid+1;
        }
        else{
            high=mid-1;
        }
    }
}
```

return ans;

}

low = 1 high = 28

$\frac{1+28}{2}$ mid = 14

14 * 14 <= 28 false

we go left side

high = mid - 1

= 13

low = 1 high = 13

mid = 7

7 x 7 <= 28 false

high = mid - 1

= 6

low = 1 mid high = 6

= 3

3 x 3 <= 28 true ans = 3

we go right side

low = mid + 1

= 4

low = 4 high = 6

mid = 5 ans = 5

5 x 5 <= 28 true

low = mid + 1

= 6

low = 6 high = 6

mid = 6

6 x 6 <= 28 false

high = mid - 1;

= 5 high = 5

low = 6

1, 2, 3

low=6 high=5

1, 2, 3, 4, 5, 6, ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~ 28

low goes on and reaches first wrong one

high eliminates and reaches first possible one

So for answer we can return high also.