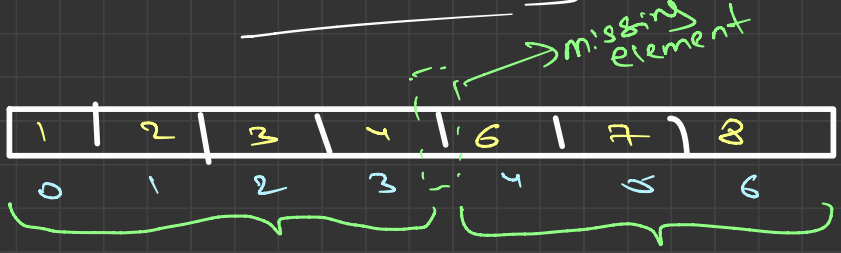
final missing



element - index == 1 ele - index == 2

So according to this :-

missing ele → before diff = 2

```
int ans = -1
while ( s <= e )
{
    int diff = arr(mid) - mid ;
    if ( diff == 1 )
    {
        s = mid+1 ;
    }
    else {
        e = mid-1 ;
        ans = mid ;
    }
}
return ans + 1 ;
}
```

Here we store the index

In actual the missing ele present at this index i.e index + 1

for bigger if the missing element are at last index

1 → n working perfectly

```
if (ans == -1)
{ return
    size+1
}
```

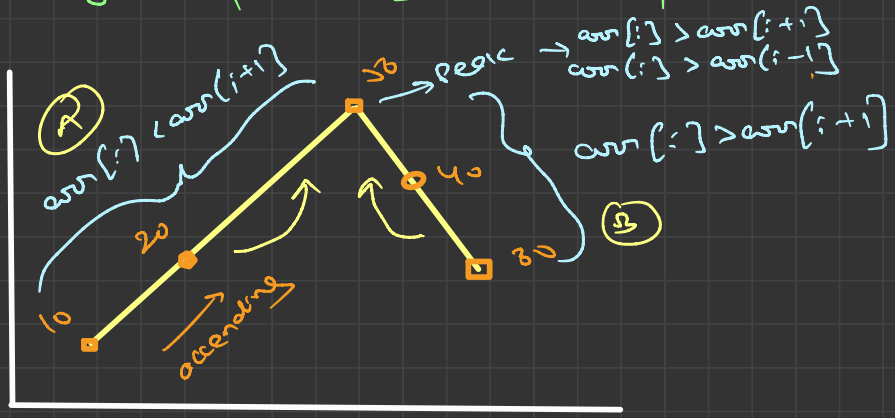
Ans = 50

Peak element in a mountain array

arr =

10	20	50	40	30
0	1	2	3	4

C



A → arr[i] < arr[i+1] → right jump

B → arr[i] > arr[i+1] → left jump

C → arr[i-1] < arr[i] > arr[i+1] → ans return

A → arr[i] < arr[i+1]

B → arr[i] > arr[i+1]

peak → arr[i] > arr[i+1] and arr[i] > arr[i-1]

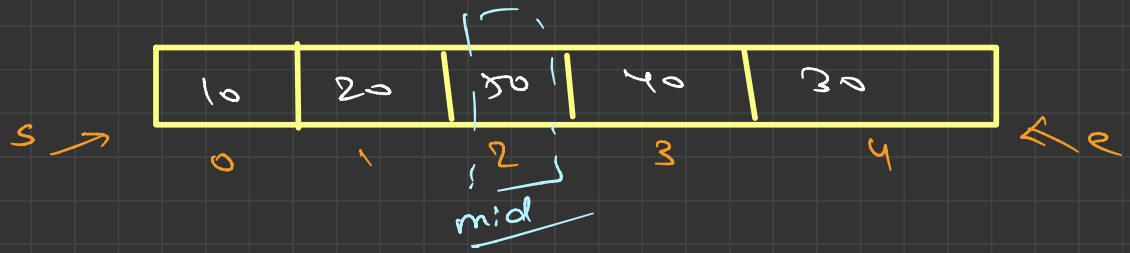
We can club both some

We can say two

arr[i] < arr[i+1] → A

↓ Not case

B + peak → return



s + e / 2 => 0 + 4 / 2 => 2

arr[mid] < arr[mid+1]

50 < 40

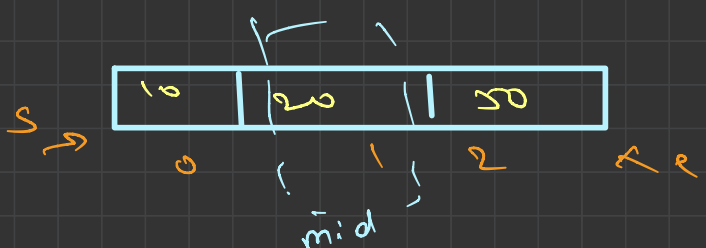
false → B or peak

left me Voyager

e = mid

why mid-1 ?

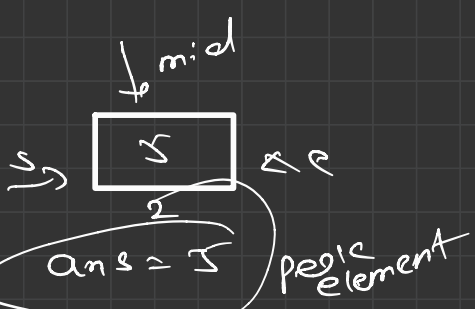
it is possible the ele is peak that's y we preserve the ele

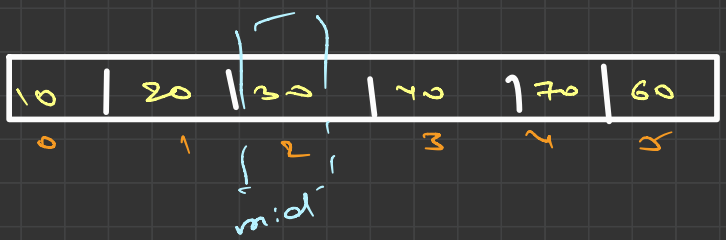


arr[mid] < arr[mid+1]

true

s = mid+1

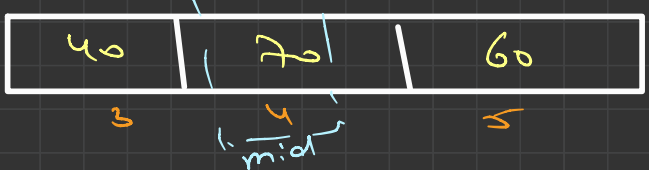




arr[mid] < arr[mid+1]

true

s = m + 1

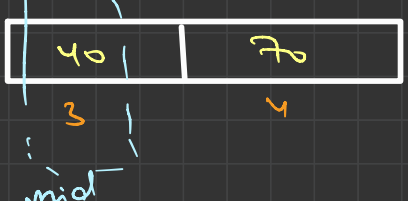


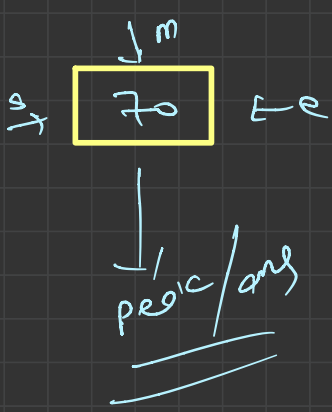
arr[mid] < arr[mid+1]

false

ans = mid

e = mid



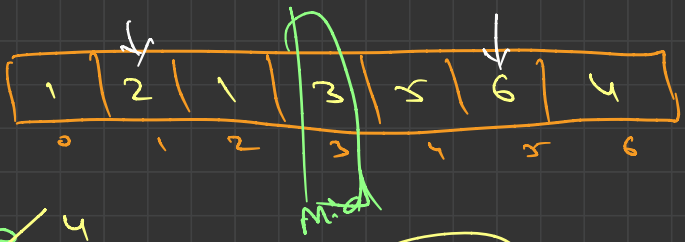


arr[mid] < arr[mid+1]

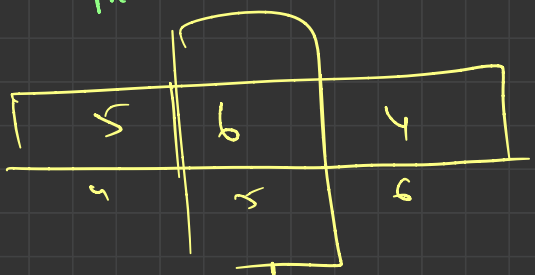
true

s = mid+1

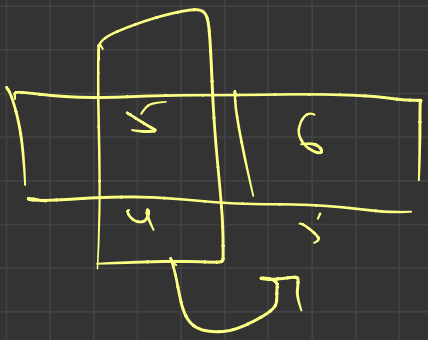
```
while (s<e)
{
    if (arr(mid) < arr[mid+1])
    {
        start = mid+1;
    }
    else
    {
        end = mid;
    }
}
return mid;
```



s = 0 4
mid = 3
e = 6



s = 4
e = 6 5
m = 5



s = 4
e = 5
m = 4

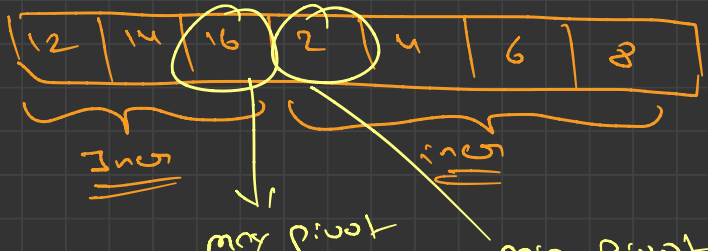

```
while ( s < e) {
    if (arr(mid) < arr(mid+1))
        s = mid+1
    else
        e = mid;

return mid
}
```

Approach

Linear Search
Binary Search
Book

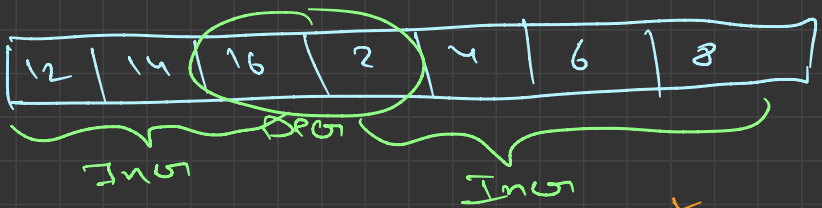
find pivot Element ← Search in a rotated & sorted array

Rotated & Sorted array ⇒

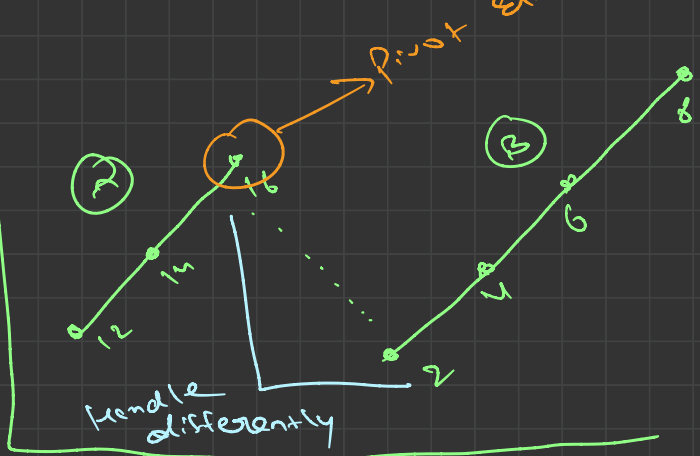


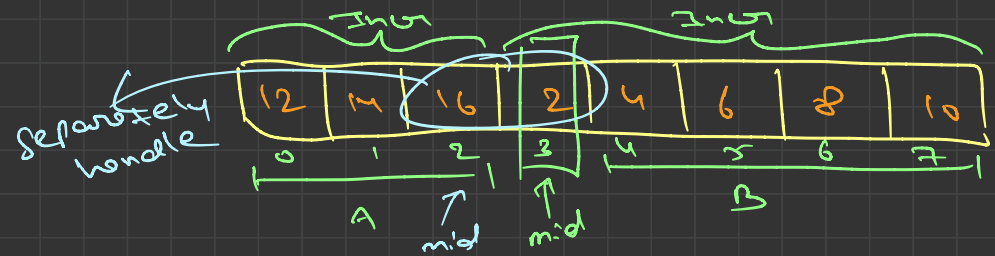
→ It is not a same as peak Question

How?



A & B are both monotonic f(n)





Suppose

If mid = 3 index

arr[mid] < arr[mid-1] → ans → mid-1

these two conditions only work these case

If mid = 2 index

arr(mid) > arr[mid+1] → ans ⇒ ans = mid

Else we are on A line or B line

How to find it ?

If we are on B line definitely we are below that index 0 element

∴ 4 < 12 ⇒ we are on line B

Move left

if arr[s] > arr[mid] → e = mid-1

Else → s = mid+1

line B always small from index 0 element

```
while ( s <= e )
{
    if (arr[mid] < arr[mid-1])
        return mid-1;

    else if ( arr[mid] > arr[mid+1])
        return mid;

    else if ( arr[s] > arr[mid] )
        e = mid-1

    else
        s = mid+1;
}
return -1;
)
```

Possibility for out of range checking

mid != 0 && mid-1 >= 0 &&

mid+1 < n &&

Corner case → Single element

↓

Handle separately

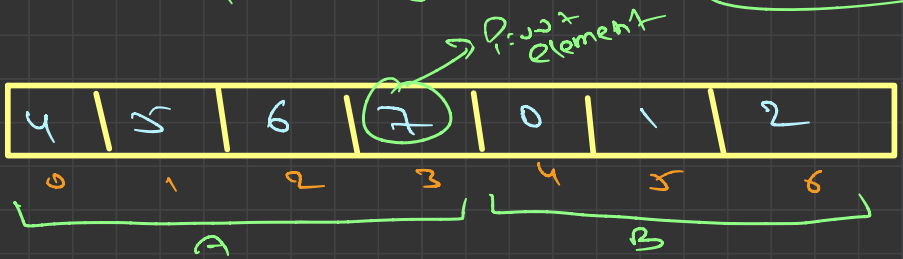
```
If ( s == e)
    return s;
```

Search in rotated & sorted array

↓

use pivot algo

target = 0



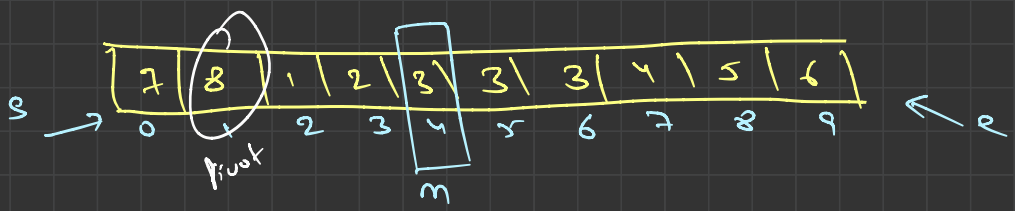
If we find pivot then it easy to find bcoz we identify two sorted array.

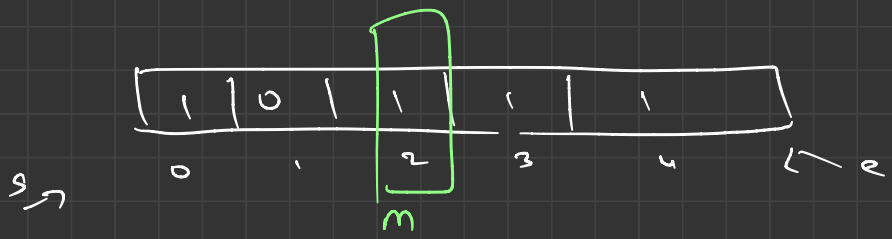
with the help of pivot ele → we can see that our target present on line B i.e → index 4 → 6

(with duplicates)

target = 3

Search in Rotated Sorted array — 2




1
0
1
1
1
0
1
2
3
4
s
e
m

Organic vegetables

→ Business WhatsApp

↓

Home delivery

↓

Orders

↓

word of mouth

↓

starts 3 months ago

↓

from Catalogue

Mistake → manually manage

↓

human mistake

↓

as a chat

↓

cannot modify

Google Doc

↓

Sheet

↓

to manage orders

Sheet

Production order

If customer

↓

ID order sheet → key

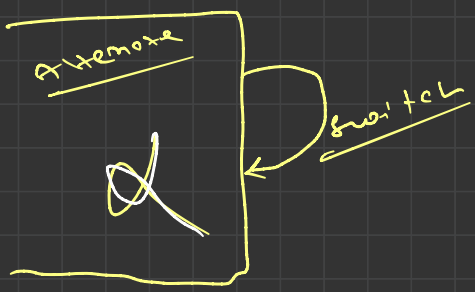
↓

order by ID key

Stock manage

Requirement

- Delivery charges → invoice
- Database manage → stocks



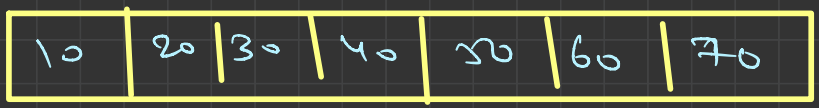
Targeting

- Personal client
- Top class
- Whatsapp mandatory

Binary Search on nearly Sorted array

Sorted array → i^{th} index → nearly Sorted array → i^{th} index, $i+1$ index, $i-1$ index

Sorted array =>



nearly sorted array =>

Sorted array

```
if (arr[mid] == target)
  { return mid; }
```

Nearly Sorted array

```
if (arr[mid-1] == target)
    return mid-1; → 2

if (arr[mid] == target)
    return mid;

if (arr[mid+1] == target)
    return mid+1; → 2
```

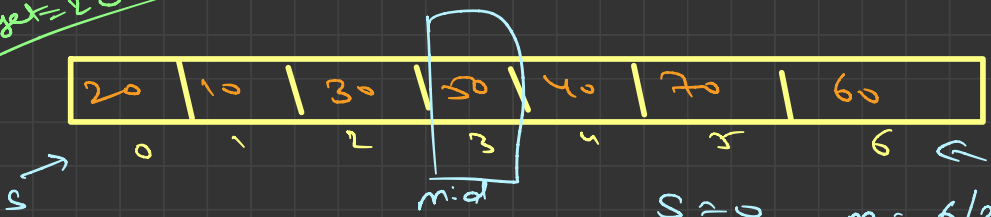
```
if (target > arr[mid])
    ↳ right

else
    ↳ left
```

```
if (target > arr[mid])
    ↳ right

else
    ↳ left
```

Jaha gaiti hogi

+2 → because we already check mid+1 & mid-1 values

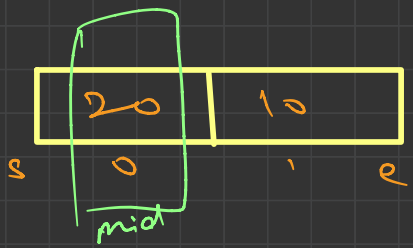
target = 20



S = 0
e = 6
m = 6/3 = 3

arr[mid] == target ✗
arr[mid-1] == target ✗
arr[mid+1] == target ✗

target > arr[mid] ✗
target < arr[mid] ✓
↳ left → e = m - 2



arr[m] == target ✓
↳ return

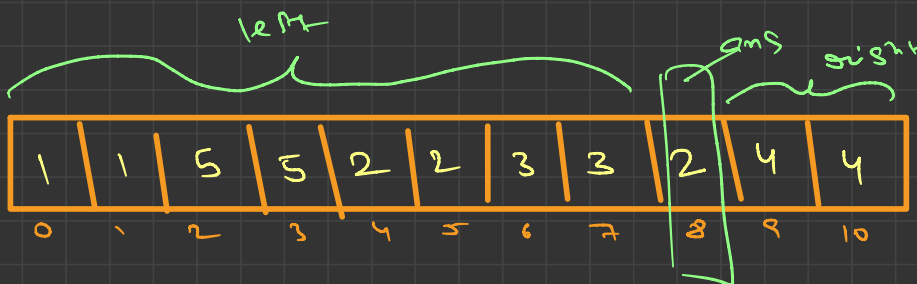
find odd occuring element → XOR (O(n))

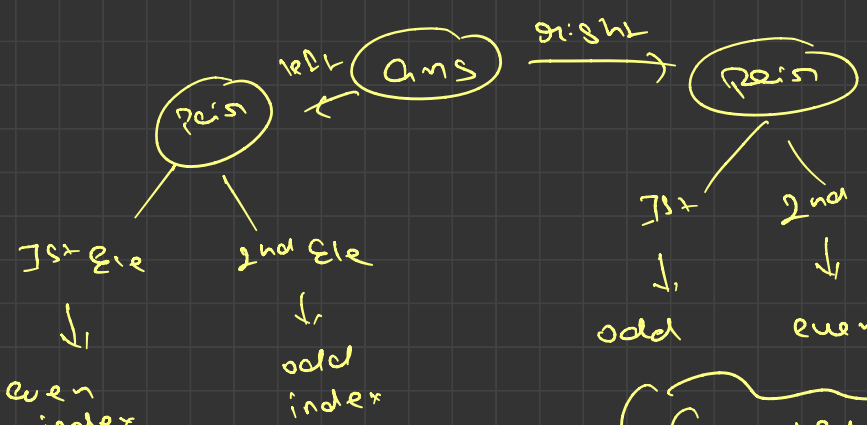
first element that occurs odd time

↳ all element → even no. of times except one → odd

↳ all repeating no. → pair
repeat & pair are not repeated

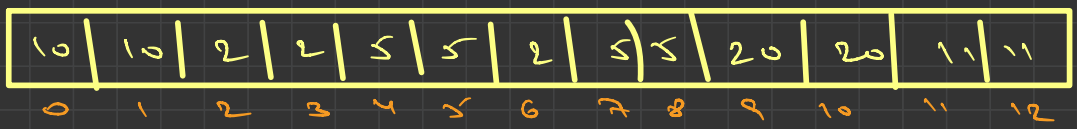
↳ ek baar me koi bhi no. 2 baar se jada nhi aaega.





Single element → ans

Guranteed
ans → Even Index



if (mid % 2 == 0) ⇒ Even

bound lagegge mid+1 < size

↳ if (arr[mid] == arr[mid+1])

↳ right → s = mid + 2

ans ke right me hu

else

↳ right / ans

e = mid;

if (mid % 2 == 1) ⇒ odd

bound lagegge mid-1 >= 0

↳ if (arr[mid] == arr[mid-1])

↳ right ⇒ s = mid + 1

ans ke left me hu

else

↳ e = mid - 1

while (s <= e) {

//Single element
if (s == e)
return s;

if (mid & 1) → odd
↳ if (arr[mid] == ans[mid-1]) → right me jaunga
↳ s = mid+1

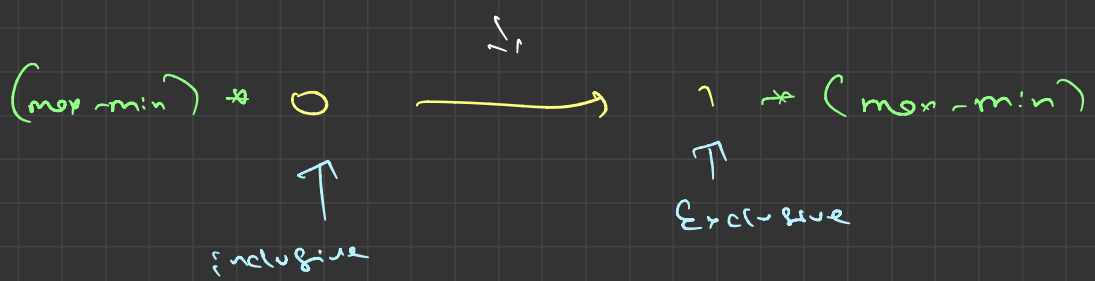
else
↳ e = mid-1

else → even
if (arr[mid] == arr[mid+1]) → left me hu → right me jaunga
↳ s = mid+2
↳ mid+1 already check ho gaya hai

else
↳ ya to ans / right me honga hai
e = mid
↳ because mid-1 se ans last ho sakta hai

e = mid;

Math.random()



0 ⟶ (max - min)

It will generate decimal value

Math.Random() * (max-min)

↓ remove decimal

Math.floor(Math.Random() * (max-min))

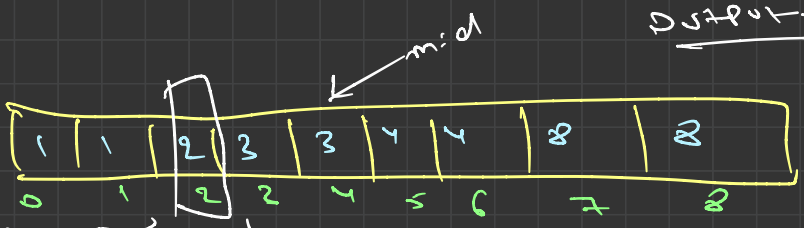
↓ But we want min → max

min + 0 ⟶ (max - ~~min~~) + ~~min~~

min ⟶ max

↓

Math.floor(Math.Random() * (max-min) + min)



Output => 2

mid = (0+8)/2 = 4

mid = 3 → 4th index

?

left

Pair of 1st ele → Even
2nd ele → odd

Check

m % 2 == 0 → even

m → m+1 are same

if not

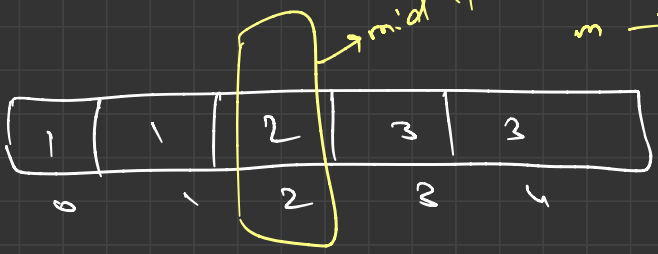
move to left

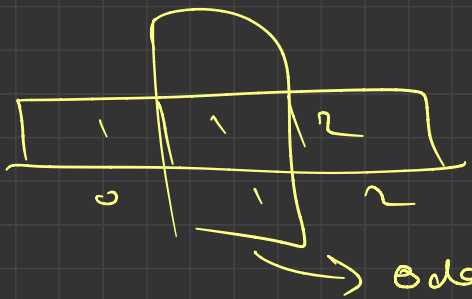
right

Pair of 1st ele → odd
2nd ele → even

Check

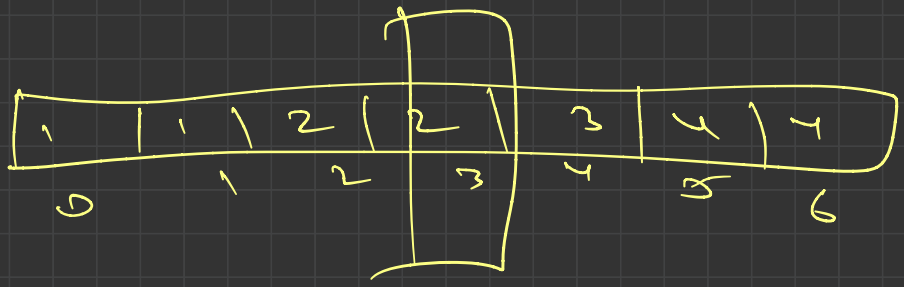
m - 1 => odd


mid + 1 → even
m → m+1 are same
no!
1
1
2
3
3
0
1
2
3
4


1
1
2
0
1
2
odd
m → m-1 are same
exist


2

Lions



2