Ceil algorithm or lower bound →

(1)

low = 0
high = n-1
ans = -1

lowest position for which arr[ind] >= target.

```
while (low <= high) {
    mid = low + (high - low) / 2;
    if (arr[mid] >= x) {
        ans = arr[mid];
        high = mid - 1;
    }
    else {
        low = mid + 1;
    }
}
return ans;
}
```

floor algorithm:

②
```
low = 0;
high = n-1;
ans = -1;
while (low <= high) {
mid = low + (high - low)/2;
if (arr[mid] <= x) {
    ans = arr[mid];
    low = mid + 1; }
else { high = mid - 1; }
}
return ans;
}.
```

Upper bound: smallest index for ~~arr~~ which arr[index] > target

(3) Find first and last position of element in Array.

Ex: $\overset{0}{[}5, \overset{1}{7}, \overset{2}{7}, \overset{3}{8}, \overset{4}{8}, \overset{5}{10}]$ target = 8

- mid = $0 + \left(\frac{5-0}{2}\right) = 2$.
- arr[mid] = 7 $(7 < 8) \Rightarrow$

  low = mid + 1 = 3.
- mid = $3 + \frac{(5-3)}{2} = 4$.

  arr[mid] = 8 $(8 = 8) \Rightarrow$

  return 4.

The crucial point now is that we have found one index, we need another index.

checkpoint 1) How do we know that there will be ≥ 2 indexes for the same key value?

checkpoint 2) After finding the index of one of the key, how to find the value of the other one?

Two methods: Both very good. Pay attention!

Method 1: Using lower bound and Upper bound.

⇒ lower bound: smallest index for which

```
nums[mid] == target
```

⇒ Upper bound: smallest index value for which

```
nums[mid] > target
```

0 1 2 3 4 5 6

Now in example : (3 5 5 8 8 99)

lowerbound gives index value = 3

Upperbound gives index value = 5.

Ans: (lowerbound, upperbound - 1)

Now implementation of lowerbound

⇒
```
low = 0, high = n-1 ;
ans = n;
while (low <= high) {
    mid = (low + high) / 2 ;
    if (arr[mid] >= x) {
        ans = mid;
        high = mid-1;
    }
    else { low = mid+1; } }
return ans; }
```

In c++ :

```
lb = lower_bound (arr.begin(), arr.end(),
                  n) - arr.begin();
```

Upper_bound implementation :

```
if (arr[mid] >= x) {
        ans = mid;
        high = mid - 1; }
else { low = mid + 1; }
```

In c++ :

```
ub = upper_bound (arr.begin(), arr.end(),
                  n) - arr.begin();
```

Method 2 : Plain Binary - search.

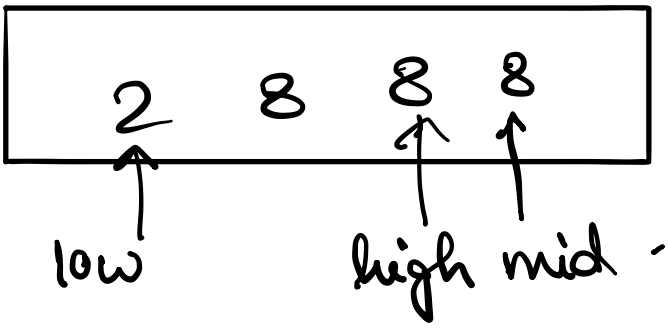
0	1	2	3	4	5	6	7
2	8	8	8	8	8	11	13

Ex:

Use two binary searches to find lowest & highest values.

→ $mid = \frac{0+7}{2} = 3$

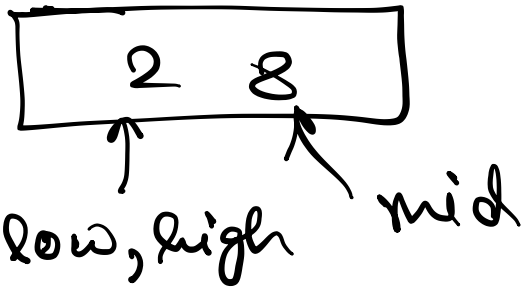
- arr[mid] = 8.
- Now to find the lowest occuring index, move mid to left side.



- high = mid - 1

Now $mid = \frac{0+2}{2} = 1$.

- arr[mid] = 8



- $mid = \frac{0+1}{2} = 0$.

$\Rightarrow$ $arr[mid] = 2$.

To find lowest 8 - index, move low forward & thus it crosses high.

So, **high = 1**

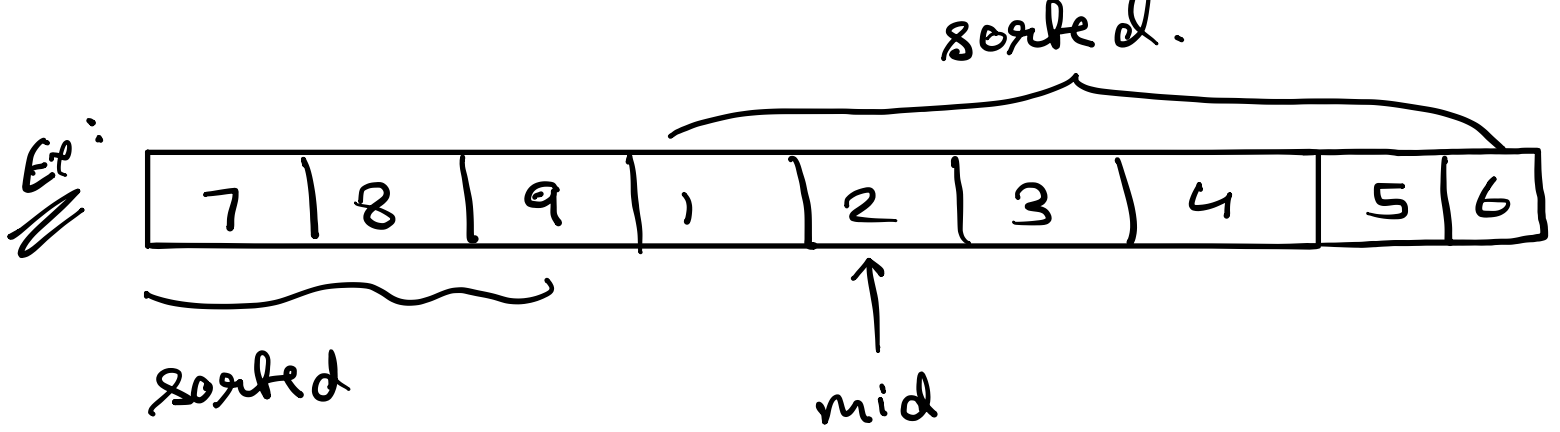
Similarly for last occurance.

(4) search in Rotated sorted Array.

Rotated at pivot index k.

Resulting array:

nums[k], nums[k+1], ----, nums[n-1],
nums[0], nums[1], -----, nums[k-1].



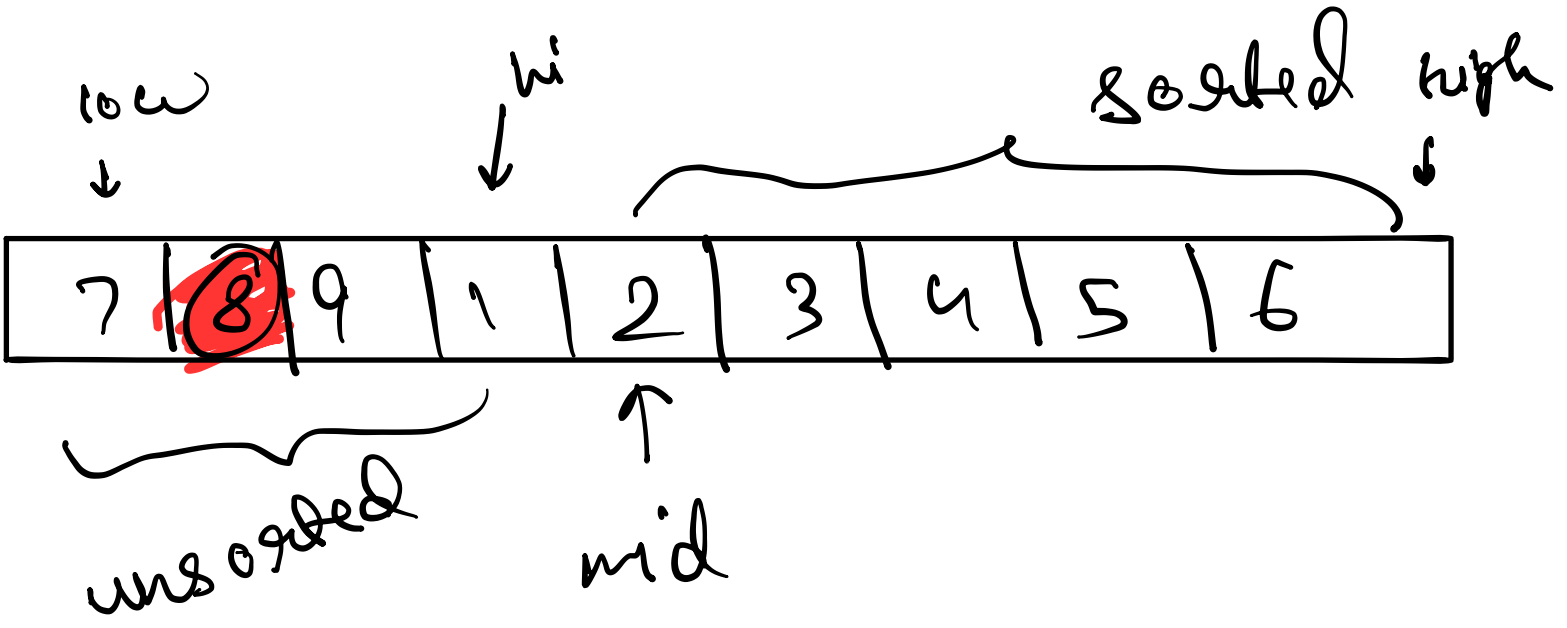
target = 8.

→ Identify the sorted half.
(either left or right).

left sorted.

if (arr[low] <= arr[mid]) {
if (arr[low] <= target &&
target <= arr[mid]) {
high = mid-1; }
else { low = mid+1; }

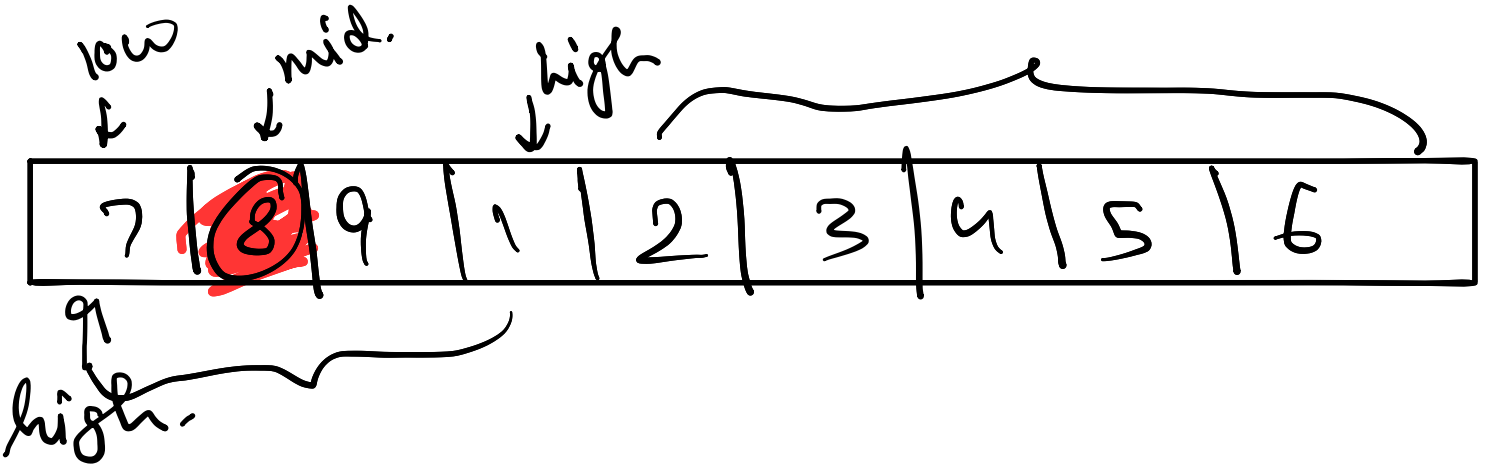
else (right sorted part).




```
if ( arr[low] <= arr[mid]) {
  if (target
```

right sorted.

```
} else if (arr[high] >= arr[mid]) {
    if (target >= arr[mid] &&
        target <= arr[high]) {
      low = mid+1;
    else  high = mid-1;
    } }.
```



left parted sorted.

⇒ if[~~target~~ arr[low] <= target && target <= arr[mid]) {

hig = mid-1;)

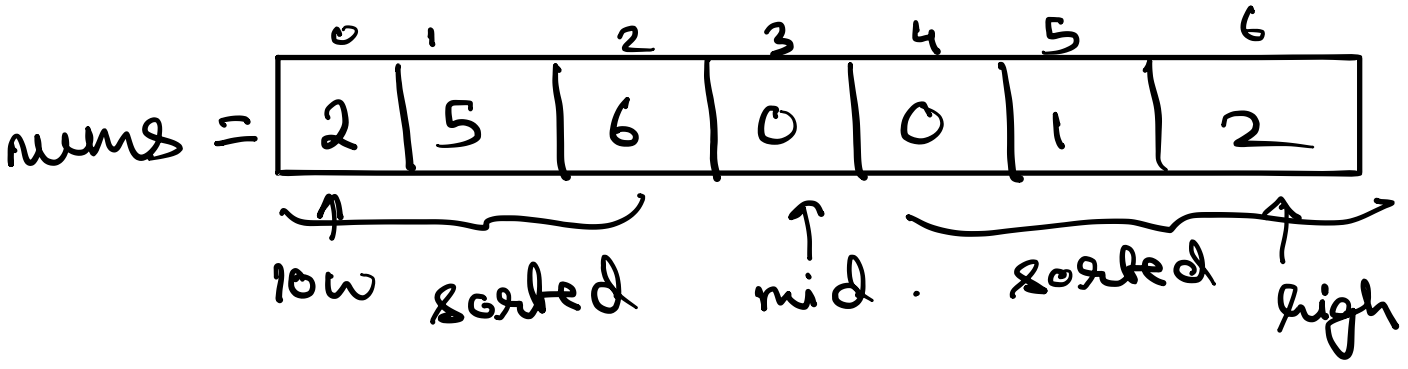
mid = 0

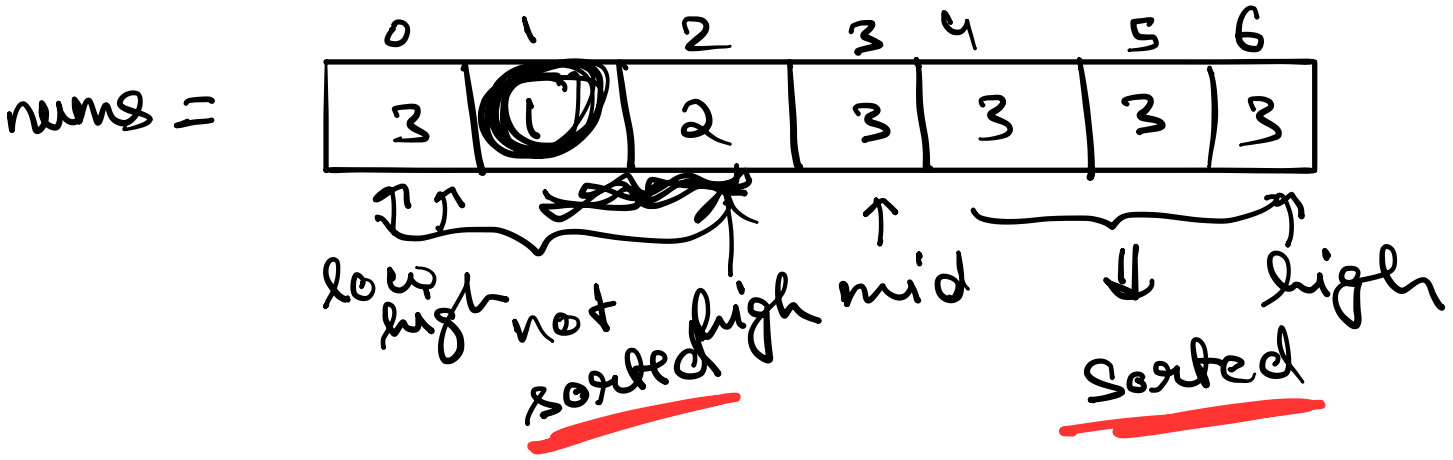
---

(5) search in Rotated Array II

nums = | 2 | 5 | 6 | 0 | 0 | 1 | 2 |

Target = 0.





target = 1.

```
if (arr[high] > arr[mid]) {
    if ( target ≥ arr[mid] &&
         target ≤ arr[high]) {
        low = mid+1; }
    else {
        high = mid-1;
    }
}
```

⋮

the problem here is by naked eyes there will be cases when (nums[low] = nums[mid] = nums[right])

thus we can't decide whether left side is sorted or right side.

thus, a condition will be added

i.e. if(arr[low] == arr[mid] == arr[high]) {

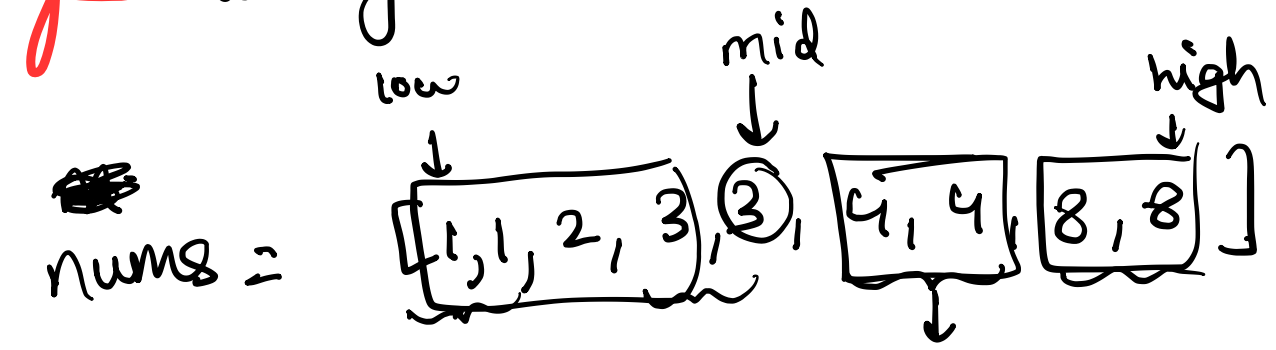
high--; low++; continue.

---

(6) Minimum in Rotated Sorted Array:

Ex: nums = [3, 4, 5, 1, 2]

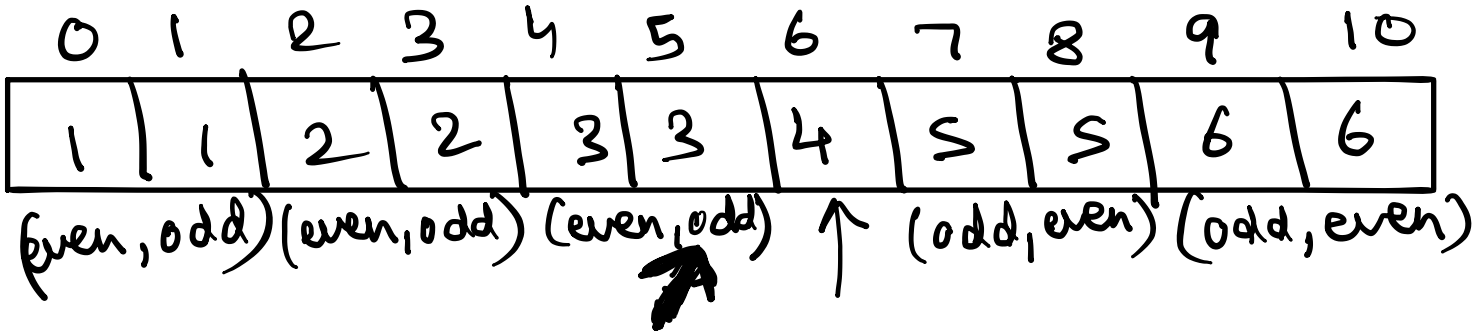
7. Single Element in a Sorted array :



output = 2

⇒

```
if (arr[mid] != arr[mid-1] ||
    arr[mid] != arr[mid+1]) {
    return arr[mid]; }
```



In the left part of the array:
if current_index is even,
element at the next
odd index will be same.

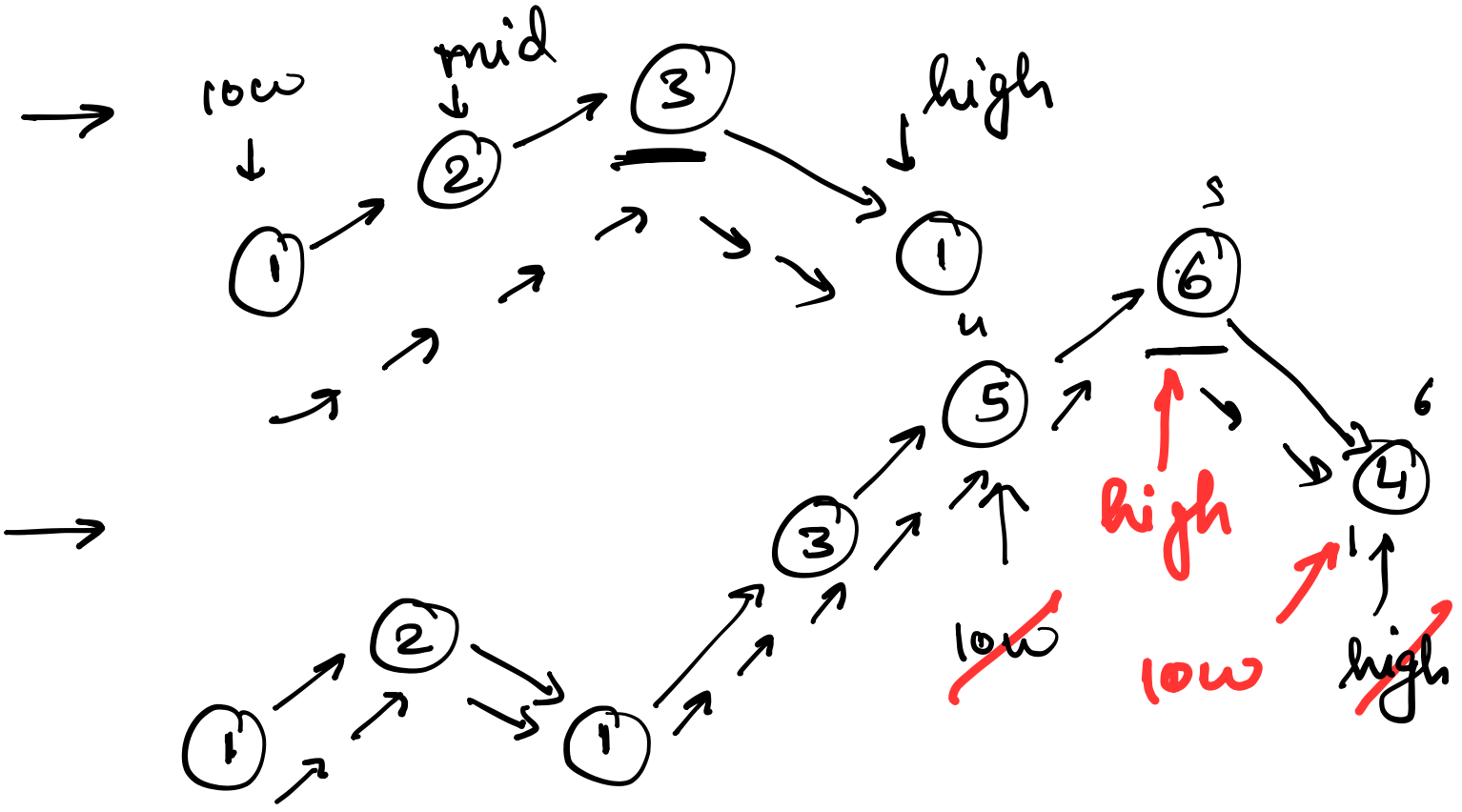
In the right part:
if current_index is ~~odd~~ odd,
element at the next even
index will be same.

if (i%2==0 && arr[i]==arr[i+1])
↳ left half.

if (i%2==0 && arr[i]==arr[i-1])
↳ right half.

left side: (even, odd)
right side: (odd, even).

⑧ Find peak element :



$\Rightarrow \left(arr[mid] > arr[mid-1]\right) \Longrightarrow$

$low = mid + 1.$

$$\left\lceil low + \frac{(high - low)}{2} \right\rceil = mid.$$

$\Rightarrow$ $arr[mid] = arr[low] = arr[high]$

$\Rightarrow$ $arr[mid-1] > arr[mid]$ &&

$arr[mid] > arr[mid+1]$.

Conclusions :

(upper bound, lower bound approach.)

1. First and last occurance of no. in sorted array.
2. search in rotated array.
3. single element in Sorted Array.
4. Peak element

either left side sorted or right side sorted, identify and solve.

before the single element same occupy (even, odd) indices, element will after that (odd, even) indices

if (mid element > (mid-1) element) ⇒
low = mid+1 ;
else high = mid-1.

Binary Search in 2D →

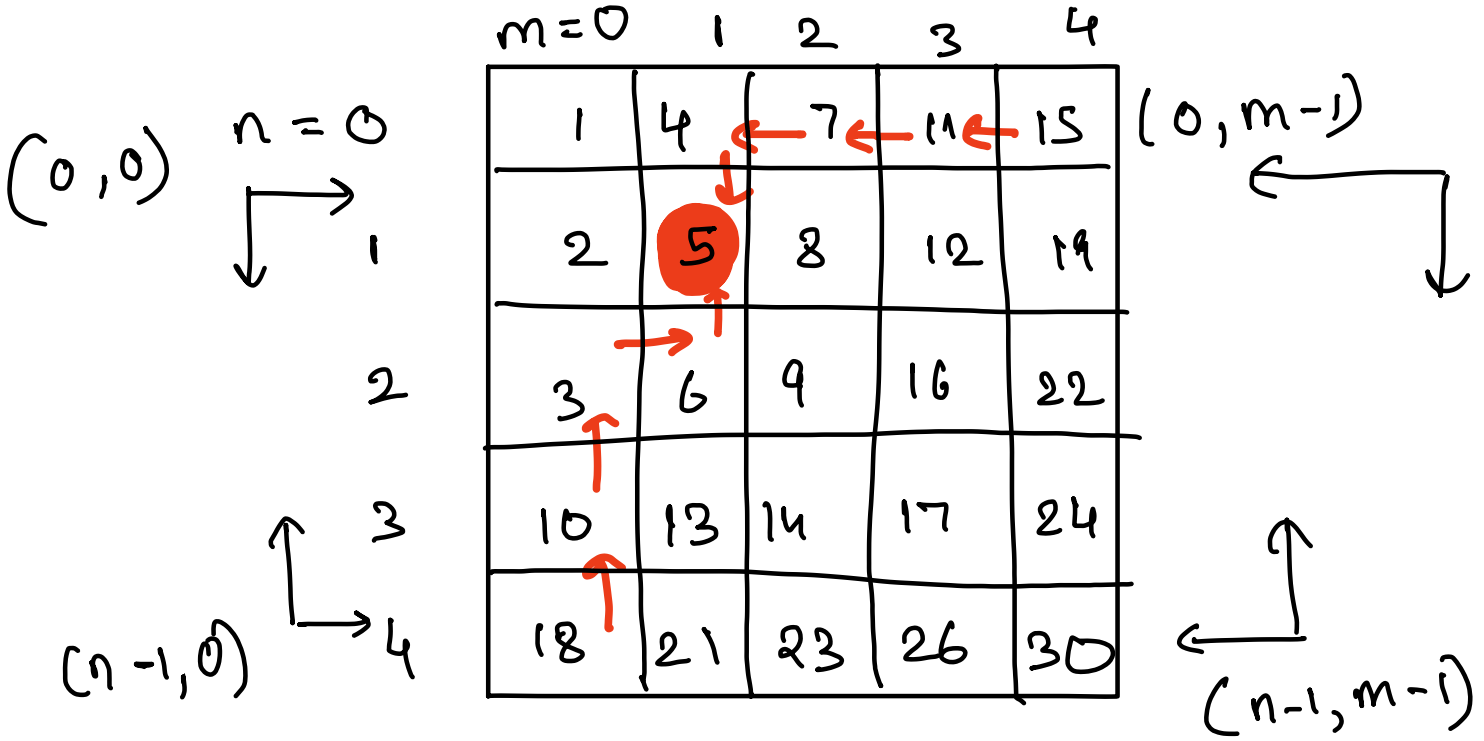
① search a 2D matrix :

target = 3

	m=0	m=1	m=2	m=3
n=0	1	3	5	7
n=1	10	11	16	20
n=2	23	30	34	60

```
low = 0; high = m*n-1;
while (low ≤ high) {
    mid = (low + high) / 2;
    row = mid/m;
    col = mid %. m;
if (arr [row] [col] == target)
        return true;
else if ( arr[row] [col] < target)
        low = mid+1;
    else    high = mid-1;
```

(2) Search a 2D-Matrix II :



↓, → : ascending

↑, ← : descending.

So, either choose (0, m-1) or (n-1, 0) as starting points as either the element is then increasing or decreasing.

if (n-1, 0) is starting point →

if (matrix[row][col] == target)

return true;

else if (matrix [row] [col] > target) {

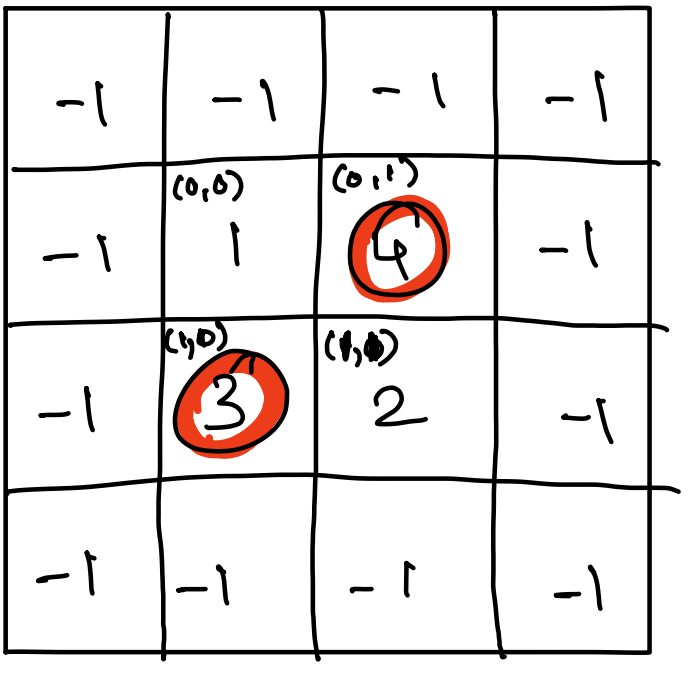
row -- ;

else col ++ ; }.

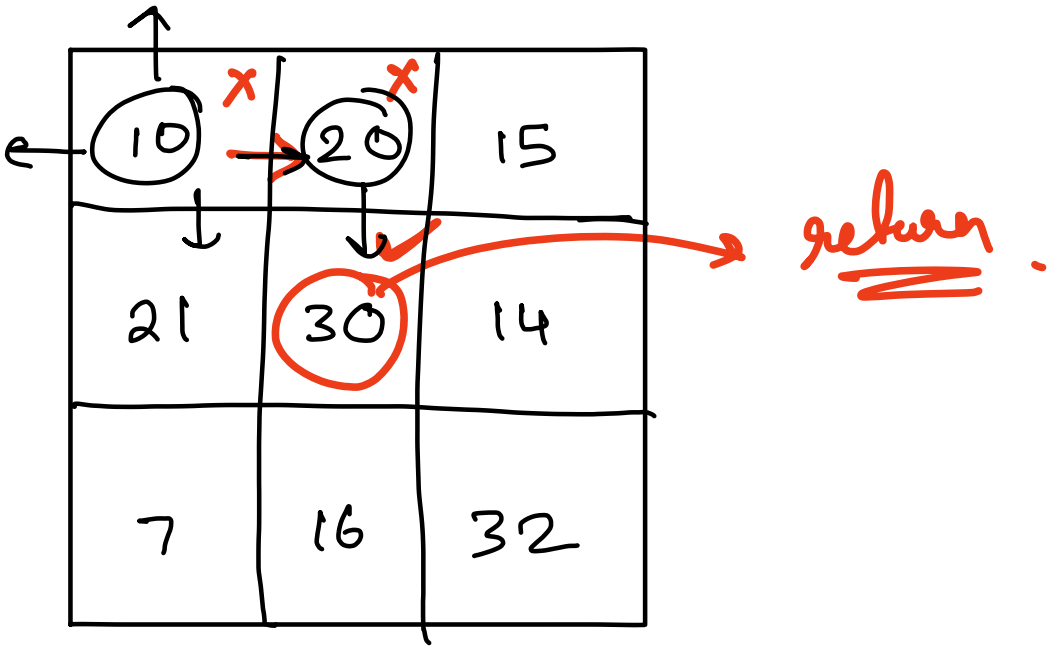
★ ③ Peak Element II →

★ Imp. question

Find any element which is greater than its surrounding values.



If (matrix [row] [col] > matrix [row-1] [col] &&


10
20
15
21
30
14
7
16
32
return

Binary search on Answers:

① Find sqrt of number using Binary search:

Ex: n = 28

```
low = 1 ; high = n .
while (low ≤ high) {
    mid = (low + high) / 2 .
    val = mid * mid ;
    if ( val ≤ n) {
            low = mid + 1 ; }
        else { high = mid - 1 ; } .
```

② n^{th} root using Binary search:

$(m)^{1/n} = x \Rightarrow \boxed{x^n = m}$

Ex: N=3, M=27

→ $X = (27)^{1/3}$

N = 4, M = 69

→ $X = (69)^{1/4}$: not integer

⇒ low = 1

high = 27

$\frac{1+27}{2} = 14$

1 2 3 - - (14) - - 27

↑

mid.

(3) Koko bananas → Find the min. eating speed so as to eat up all the piles in h hours.

Ex: n = 4,

a[] = 7, 15, 6, 3

h = 8

let say a[] = 3, 6, 7, 15 (it won't affect the time calculation)

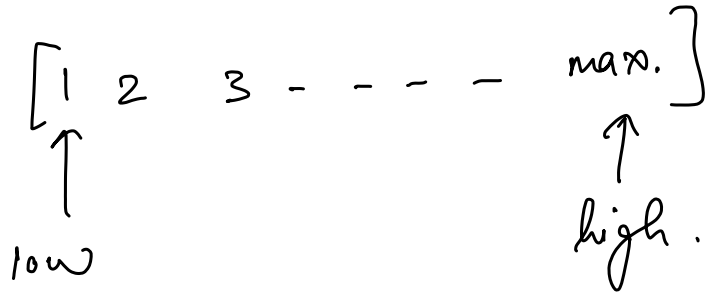
Listen the approach:

low = 1 ⇒

Total hours = sum of elements.

high = max_of_arr.

Total hours = size of arr.



$$mid = \frac{low + high}{2} = \frac{1+15}{2} = 8$$

If mid = 8 ⇒ Assume Koko eats.

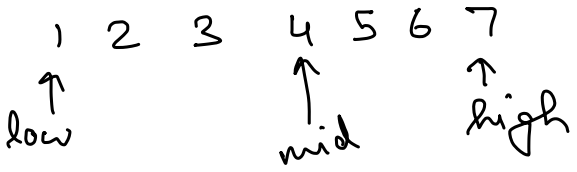
8 bananas per hr.

$$\Rightarrow \frac{3}{8} + \frac{6}{8} + \frac{7}{8} + \frac{15}{8}$$

$$= 1 + 1 + 1 + 2 = 5$$

$5 < h\ (=8) \Rightarrow$

high = mid − 1 = 7.



$mid = \frac{8}{2} = 4$

$\Rightarrow$ If Koko eats 4 bananas / hr $\Rightarrow$

$$\frac{3}{4} + \frac{6}{4} + \frac{7}{4} + \frac{15}{4}$$

$$= 1 + 2 + 2 + 4 = 9$$

$\Rightarrow$ $9 > h(=8)$

$\Rightarrow$ **low = mid + 1 = 5.**

$\Rightarrow$ [1 2 3 4 5 6 7].

low mid high

$\Rightarrow$ mid = 6

$\Rightarrow$ If 6 bananas / hr $\Rightarrow$

$$\frac{3}{6} + \frac{6}{6} + \frac{7}{6} + \frac{15}{6} = 1 + 1 + 2 + 3$$

$= 7$

$\Rightarrow$ $7 < h(=8) \Rightarrow$ high = mid − 1

=> [1 2 3 4 5 6 7]

low, high

mid = 5

=> $\frac{3}{5} + \frac{6}{5} + \frac{7}{5} + \frac{15}{5} = 1+2+2+3 = 8$ days

Done..

(4) Minimum days to make M bouquets:

→ ith flower will bloom on the a[i]th day.

→ To make a bouquet, we need K adjacent bloomed flowers

→ Here we need 2 bouquets in total.

sorted: [7 7 7] [7 7 11] 12 13

Ans = 12

=) sorted : 1, 2, 3, 10, 10

case1: mark > size =>

return -1;

case2: Sort the array, the minimum element will correspond to the low pointer and the maximum element will correspond to the high pointer.

2.1: Assume all flowers bloom on day 7. =>

Ans = low (= 7)

2.2: Assume all flowers bloom on day 14 =>

Ans = high (= 14)

Range ∈ [low, high]

mid = low + $\frac{(high - low)}{2}$.

⇒ 7 7 7 7 7 11 12 13

low ↑ high

⇒ mid = 10

(5) Smallest Divisor given a threshold:

Ex: nums = {1, 2, 3, 4, 5}
limit = 8

let divisor = 1 =>

sum = 15

let divisor = 2 =>

1 + 1 + 2 + 2 + 3 = 9 (ceil division)

sum = 9

let divisor = 3 =>

1 + 1 + 1 + 2 + 2

sum = 7.

let divisor = 4 =>

1 + 1 + 1 + 1 + 2

sum = 6

let divisor = 5 =>

1 + 1 + 1 + 1 + 1

sum = 5

search-space: {1, max(nums)}

$$mid = \frac{1 + \max(nums)}{2} = \frac{3}{}.$$

Sum = 7 < 9

{1, 2, 3, 4, 5}

low mid high

high = mid - 1 = 2.

new_mid = $\frac{1+2}{2}$ = 1

Sum = 15 > 9.

low = mid + 1

⇒ Ta-daaam ! Ans = 3

(6) Capacity to ship packages within D days

- Days ≤ d.
- least weight capacity to ship all packages with-in d days.

Ex: nums = 5 4 5 2 3 4 5 6

d = 5

- the weights are loaded in the same order as they appear in nums.

⇒ let say max_capacity = 9.

Day 1 → (5, 4) → 9 kg.
Day 2 → (5, 2) → 7 kg.
Day 3 → (3, 4) → 7 kg.

} 5days.

Day 4 → (5) → 5Kg.
Day 5 → (6) → 6Kg

search space : {max(nums), Σ nums[i]}

⇒ [6, ------, 34]

⇒ mid = $\frac{6+34}{2} = \underline{\underline{20}}$.

⇒ If weight_limit = 20

→ Day 1 : (5, 4, 5, 2, 3) → 19 Kg.

→ Day 2 : (4, 5, 6) → 15 Kg.

Total = 2 days

So, high = mid - 1 = 19.

⇒ [6, 7, 8, ------, 19]
↑ low ↑ high

mid = $\frac{6+19}{2}$ = 12.

⇒ weight_limit = 12

→ Day 1 : (5, 4) → 9
→ Day 2 : (5, 2, 3) → 10
→ Day 3 : (4, 5) → 9
→ Day 4 : (6) → 6

} 4 days.

high = mid - 1 = 11.

$mid = \frac{6+11}{2} = 8$

Day 1 : (5) → 5kg
Day 2 : (4) → 4kg.
Day 3 : (5, 2) → 7kg
Day 4 : (3, 4) → 7kg
Day 5 : (5) → 5kg
Day 6 : (6) → 6kg

} 6 days.

low = mid + 1.

$\Rightarrow \{9, 11\}$

$\Rightarrow$ mid = 10

Day 1 : (5,4) = 9 kg
Day 2 : (5,2,3) = 10 kg
Day 3 : (4,5) = 9 kg
Day 4 : (6) = 6 kg

} 4 days.

high = 9

$\Rightarrow \{9, 9\}$

ans = 9

7) k^{th} missing positive number:

Ex: nums = [4,7,9,10]

k = 1

$\Rightarrow$ missing nums :

[1 2 3 5 6 8 11 12 - -]

⇒ K=1 ⇒ Ans=1

Method 1: Create a hash-map; store all the nums that are not present in the nums array.
Then output the (K^th) index element. → O(N) soln.

Method 2:

(8) Aggressive Cows:

→ To find the maximum minimum distance between two cows such that K of them is settled.

* low = 1
high = stalls[n-1] - stalls[0].
int ans = 1;
while (low ≤ high) {
mid = $\frac{low + high}{2}$;
int last = stalls[0];
for (int i=0; i< stalls.size(); i++){
if (stalls[i] - last >= mid) {
count++;
last = stalls[i];
}
}
if (count ≥ K){
result = mid;
low = mid+1;
}
else { high = mid-1; } }

(9) Allocate minimum Number of pages:

arr[i] → number of pages in the i-th book.

Total m students.

→ Each student gets ≥ 1 book.

→ Each book allocated to only 1 student

Ex: nums = 12, 34, 67, 90.
m = 2 (no. of students).