- Binary search basics
- Transition Point
- First Bad Version
- Guess Higher or Lower
- Floor and Ceil
- First and Last occurence
- Count Occurences (HW)

① ②

- upper & Lower Bound
- Search Insert Position
- Closest element
- K closest elements
- Heaters.

③ ④ ⑤

① Binary Search

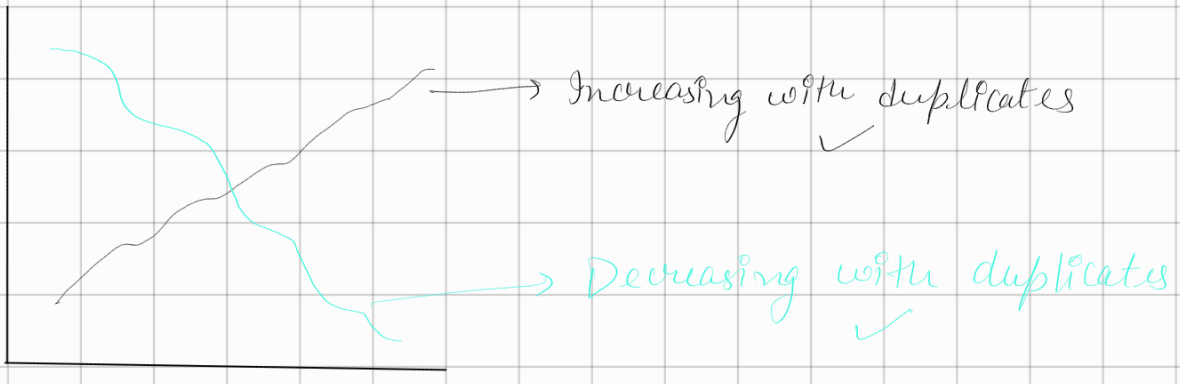
→ Optimises time complexity to $O(\log_2 n)$

→ Constraint : Monotonic (increasing / decreasing)

or

Binary

Duplicate values are also allowed



→ Modified Binary Search works for

- Rotated Sorted array
- Bitonic array
- Infinite array {Unbounded}
- Matrix
- Nearly Sorted array

(K sorted, an element can be $\pm K$ displaced from it's sorted place)

Binary Search

	low ↓				mid = 0+(11-0)/2 ↓					high ↓
arr	-40	-20	-10	0	10	30	30	60	80	90	100
idx	0	1	2	3	4	5	6	7	8	9	10

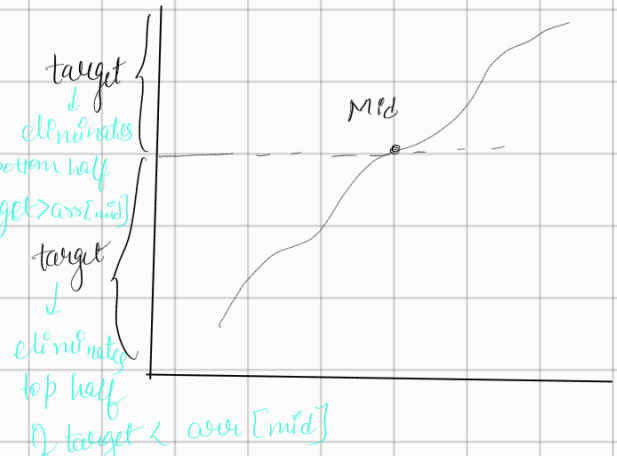
Step ① find mid = low + (high - low)/2

② check if mid value is less / more than target

if less → search in right half

if more → search in left half

if equal → found



To find 60 in above arr,

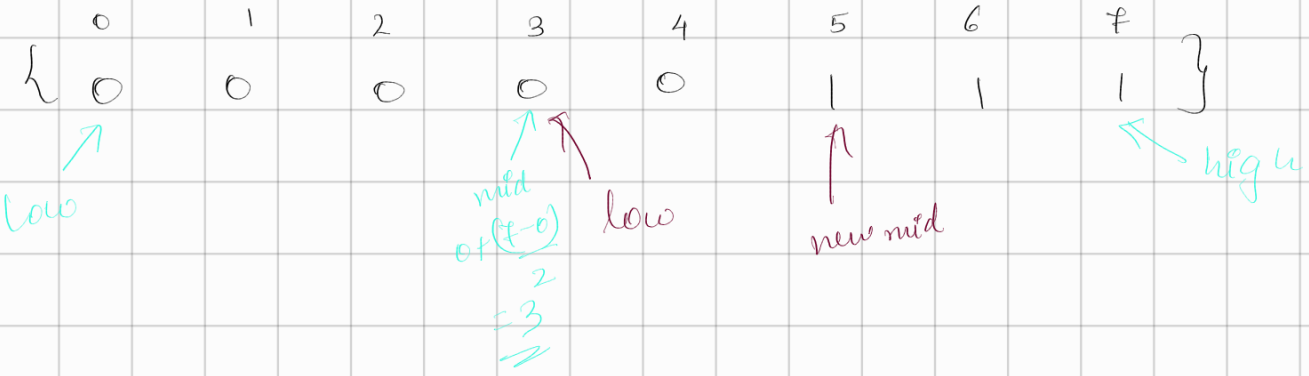
size ⇒ 11 ⟶ 5 ⟶ 2 ⟶ 1 Updated as ⌊n/2⌋ floor value

∴ CODE:

```
int left = 0, right = nums.length - 1;
while( left <= right)
{
    int mid = left + (right - left)/2;
    if( nums[mid] == target){
        return mid;
    } else if (nums[mid] < target){
        left = mid + 1;
    } else {
        right = mid + 1;
    }
}
return -1; // not found.
```

② Find Transition Point.

In a sorted binary array, find the first occurence of 1



at mid = 3 → arr = 0, so not = 1, move to right side

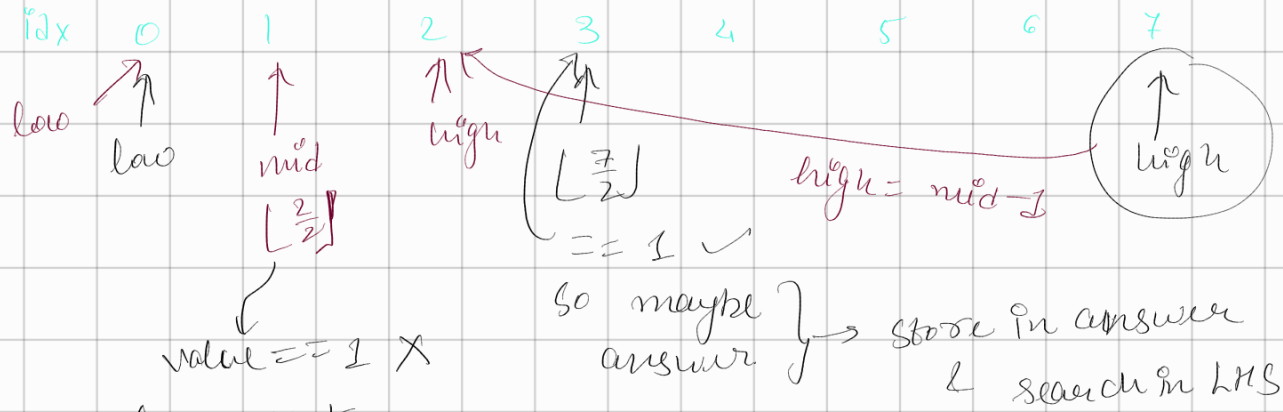
⇒ low = mid + 1

at mid = 5 → high = mid - 1

Now high <= low, so break / loop ends

Now, the index in ans variable is correct answer

Eg: { 0 0 1 1 1 1 1 1 }



value == 1 ✗

So cannot be answer → don't store in answer

So maybe answer → store in answer & search in LHS

In 3rd iteration, low = mid + 1

In 4th iteration, high = mid - 1

∴ high < low → loop breaks

✡ observation, return the low pointer after loop ends because it will point to answer index any way but not if 1 is absent

③ First Bad Version (Similar to First Transition Problem)

	0	0	0	0	1	1	1
	F	F	F	F	T	T	T
idx	1	2	3	4	5	6	7

1 based indexing

using binary search and in condition use the API

left, mid, right

v_1	v_2	v_3	v_4	v_5	v_6	v_7
0	0	0	0	1	1	1
l			mid	left	mid	r / right

mid → not bad → left = mid+1

mid → bad → pot. ans = 6, right = mid − 1

after 3rd iteration, any updation gives (rights = left) → false

So loop breaks

Similar questions can be asked as interactive problem in contests

(4) Guess number is higher or lower.

In a range of 1 to n, find a picked number using given function guess, which returns

→ -1 if pick < num

+1 if pick > num

0 if pick == num

Starting Range [1, n] inclusive

Approach can be similar to First Bad Version, We pick the middle element and check for 3 cases

if (guess(mid) == 0)

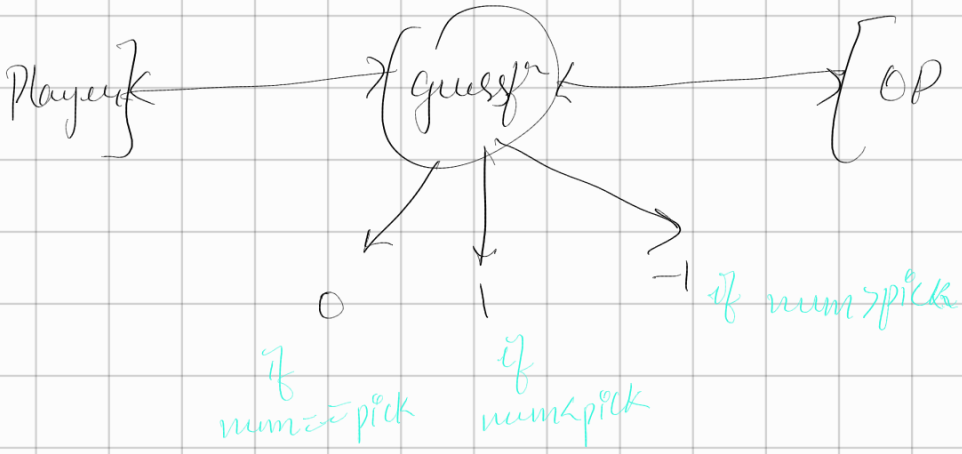
we found pick, return it

else if (guess(mid) == 1)

Search in right half of current range

else

Search in left half of current range



CODE:

```
int left=1, right=n;
while (left<=right){
    int mid = left+(right-left)/2;
    int ans = guess(mid);
    if(ans==0) return mid;
    else if(ans==1) right = mid-1;
    else left = mid+1;
}
return-1;
```

⑤ Broken Economy { floor & ceil }

Question: find floor & ceil value of key element.

idx	0	1	2	3	4	5	6	7
Eg:	5	10	15	22	33	40	42	55

low ↑ (0) high ↑ (7)

for 41 → ceil = 42
floor = 40.

- ☆ floor, largest smaller value
- ☆ ceil, smallest large value

for floor:

① → find mid
② → if (mid < target)
l = mid+1
else
r = mid-1
③ when r < l,
r will be the answer for floor

for ceil

① → find mid
② → if (mid < target)
l = mid+1
else
r = mid-1
③ when r < l,
l will be the answer for ceil

Observation: the right / high pointer points to floor value when loop ends
the left / low pointer points to ceil value when loop ends

☆: If equal elements are found, we need to move both sides from the, so we need to apply 2 different functions / methods to find floor and ceil

☆: for corner case (answer on left or right extremes)
if (right == -1) return value specified in question
Similar for (left == n) ;

⑥ First and Last Position of Element in sorted array

0	1	2	3	4	5	6	7	8	9	10
10	10	20	20	30	30	30	40	40	40	50

target = 40

We cannot find both values in only one binary search

① for upper bound

```
if (mid <= target)
        left = mid+1;
else
        right = mid-1;
if (mid == target)  ans = mid;
```

② for lower bound

```
if (mid >= target)
        right = mid-1
else
        left = mid+1
if (mid == target)  ans = mid;
```

⑦ Count Occurences

Similar to above question and

ans = higher Bound − lower Bound + 1

$[l, r] \Rightarrow r - l + 1$

$[l, r]$ or $[l, r) \Rightarrow r - l$

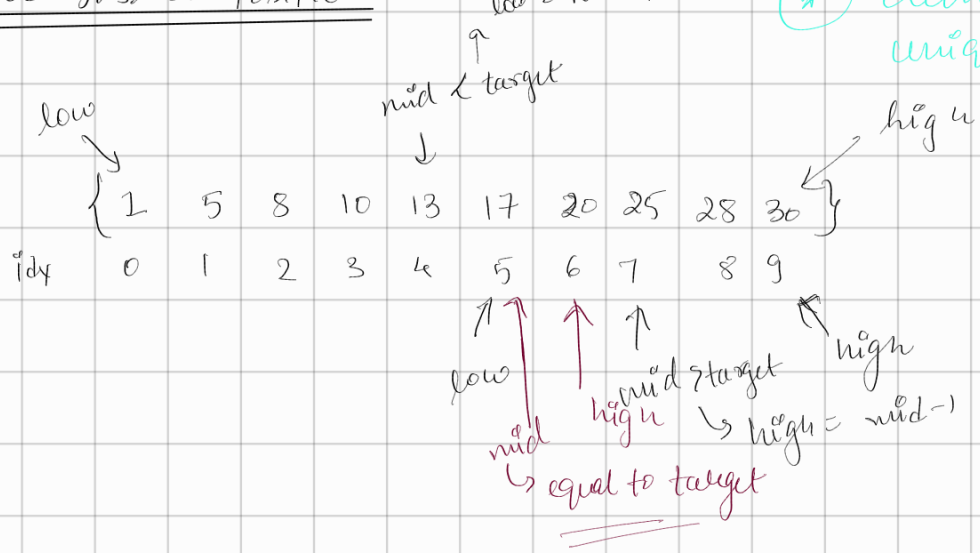
$(l, r) \Rightarrow r - l - 1$

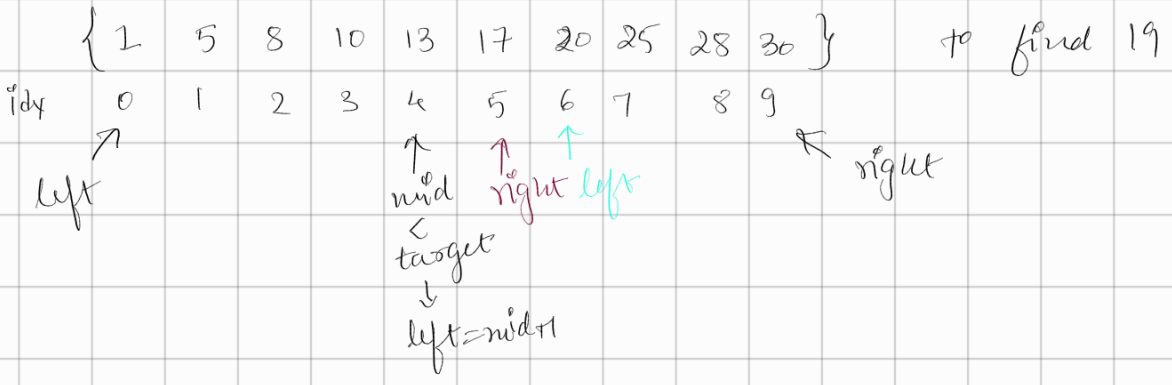
Check if any Bound == −1, element not found

⇒ return 0;

⑧ Search Insert Position

⑦ Elements have to be unique.





If element found, then mid, else left.

⑨ Lower and Upper Bound

Lower: If element found: return first occurrence
If element not found: return ceil

Upper: always return ceil

Eg: 1 1 1 2 2 4 4 4 5 5
0 1 2 3 4 5 6 7 8 9

↳ lower(2) = 3
upper(2) = 5 (not 4)

lower(3) = 5, upper(3) = 5

lower (0) = 0 upper(0) = 0

lower(6) = arr. size() upper(6) = arr. size().

Approach, Lower Bound: Same as Search Insert Position

Upper Bound: Similar to Search Insert Position but change logic for mid == target