

Searching & Sorting

Starting at 9:10

Lecture ①

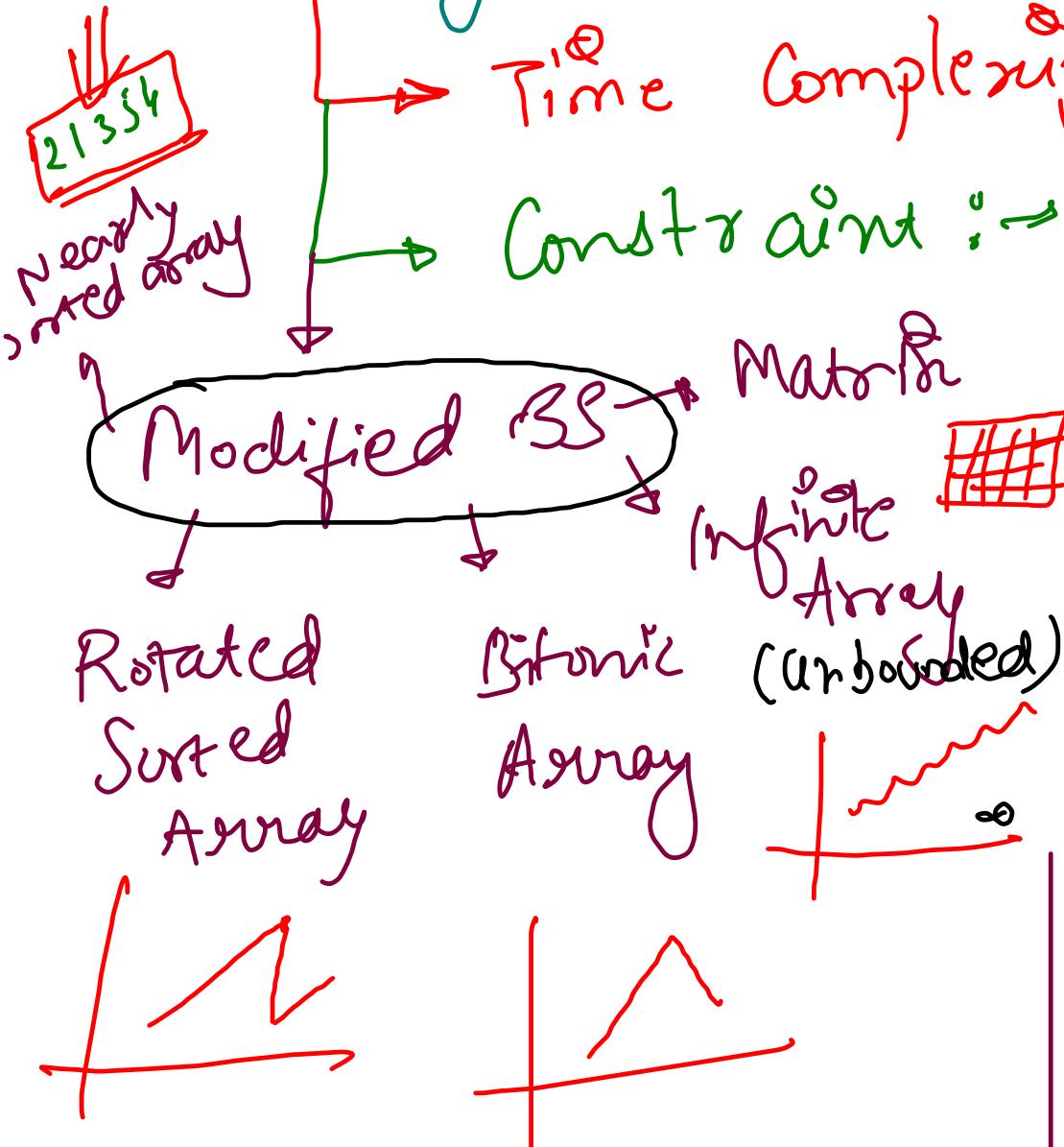
Binary Search Basics

- ① Transition Point
- First Bad Version
- Guess Higher or Lower

- ② Floor & ceil
- First & last Occurrence
- Count Occurrences (HW)

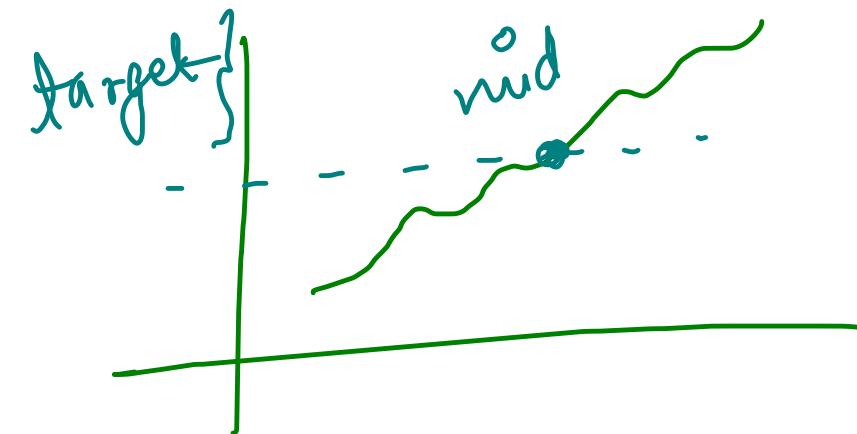
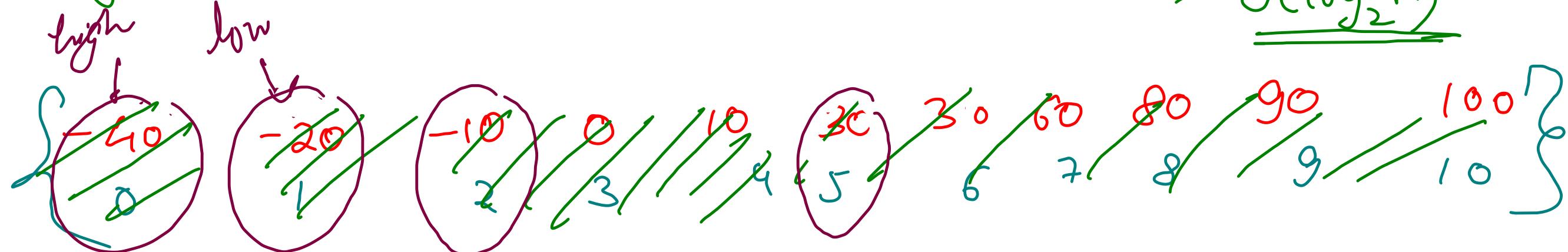
- ③ Upper & lower Bound
- Search Insert Position

Binary Search



$O(N)$ comparisons
linear search

Binary Search

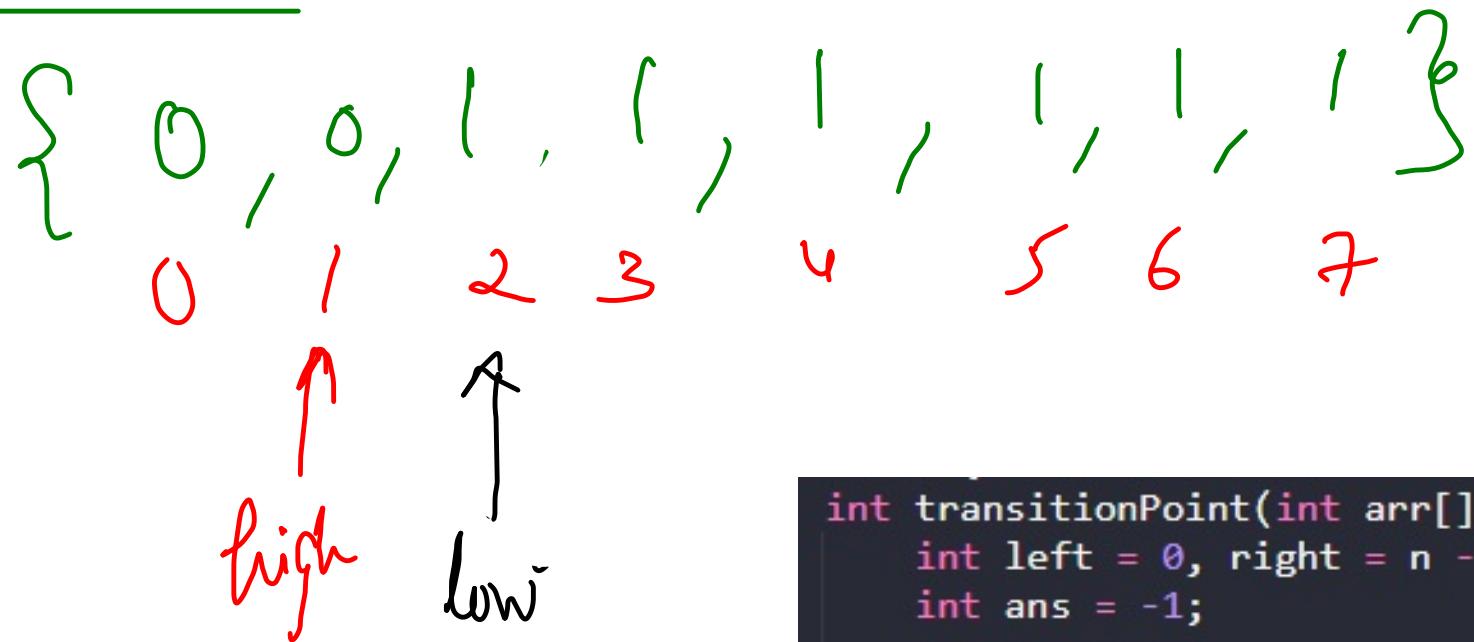


$$T(n) = T(n/2) + k \Rightarrow \underline{\mathcal{O}(\log_2 n)}$$

```
public int search(int[] nums, int target) {  
    int left = 0, right = nums.length - 1;  
  
    while(left <= right){  
        int mid = (left + right) / 2;  
  
        if(nums[mid] == target){  
            return mid; // search successful  
        } else if(nums[mid] < target){  
            left = mid + 1;  
        } else {  
            right = mid - 1;  
        }  
  
    }  
  
    return -1; // search unsuccessful  
}
```

target \Rightarrow -30

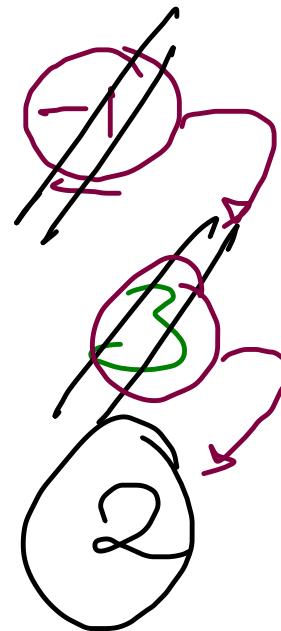
Transition Point



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

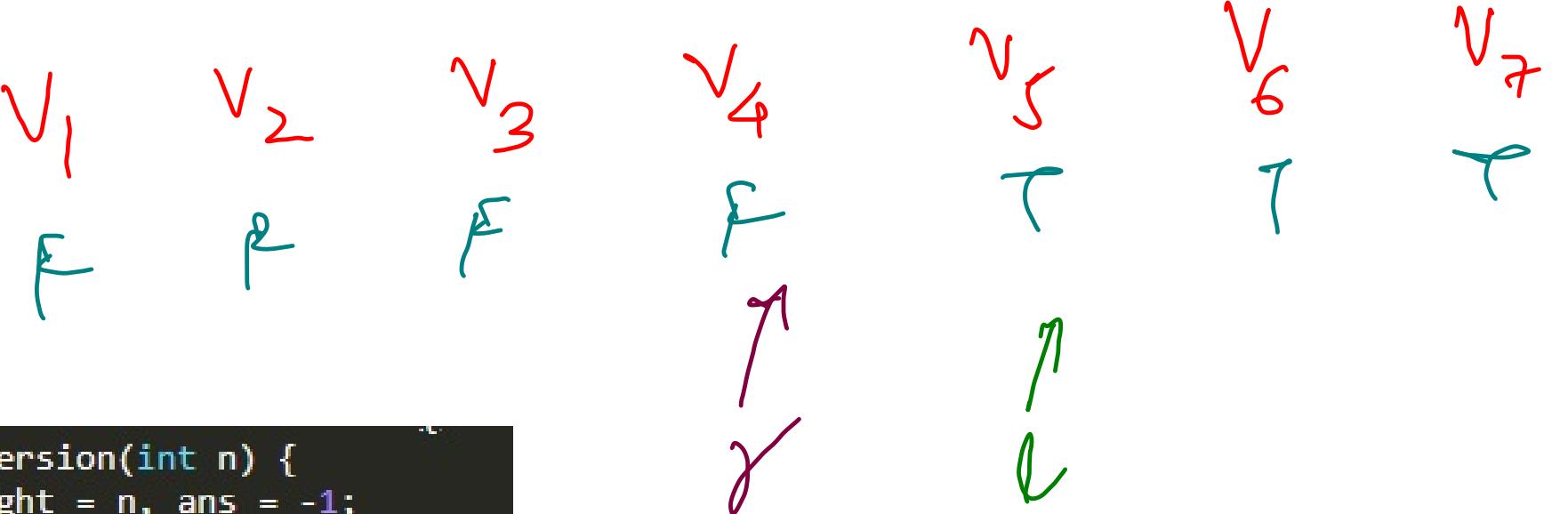
First Occurrence of 0

ans =



First Bad Version

bool isBadVersion(v)



```
public int firstBadVersion(int n) {
    int left = 1, right = n, ans = -1;

    while(left <= right){
        int mid = left + (right - left) / 2;

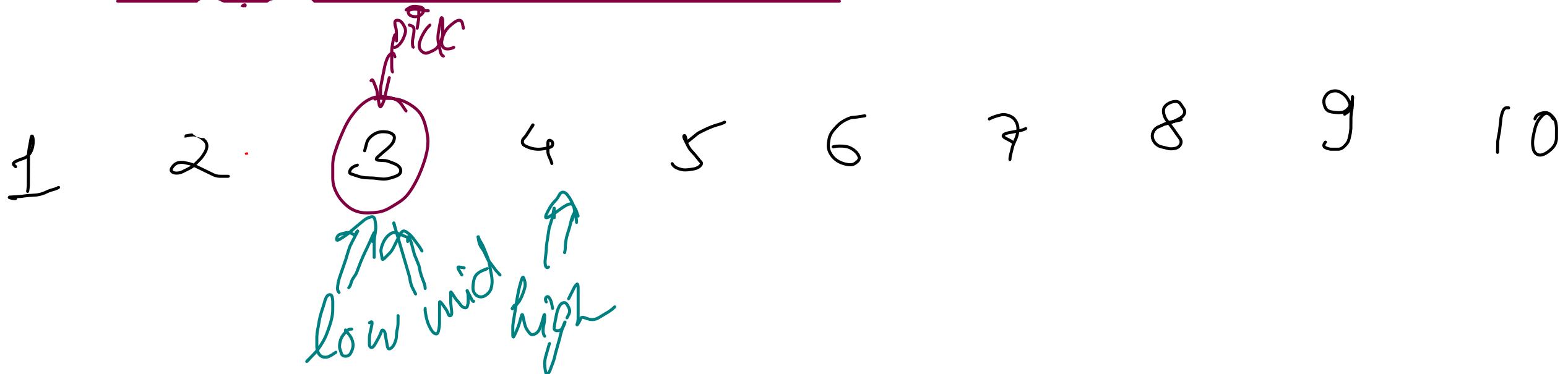
        if(isBadVersion(mid) == false){
            left = mid + 1;
        } else {
            ans = mid;
            right = mid - 1;
        }
    }
    return ans;
}
```

Ans = ~~6~~

~~6~~

5

(Q374) Guess Number or lower



```
public int guessNumber(int n) {
    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;
}
```

guess(5) :- -1
guess(2) :- +1
guess(3) :- 0

Floor & ceil of Broken Economy }

target = 60

	0	1	2	3	4	5	6	7
	[5, 10, 15, 22, 33, 40, 42, 55]							



```
public static int floor(int[] arr, int target){  
    int left = 0, right = arr.length - 1;  
    while(left <= right){  
        int mid = left + (right - left) / 2;  
  
        if(arr[mid] >= target)  
            right = mid - 1;  
        else left = mid + 1;  
    }  
    return arr[right];  
}  
  
public static int ceil(int[] arr, int target){  
    int left = 0, right = arr.length - 1;  
    while(left <= right){  
        int mid = left + (right - left) / 2;  
  
        if(arr[mid] <= target)  
            left = mid + 1;  
        else right = mid - 1;  
    }  
    return arr[left];  
}
```

if $\text{right} \Rightarrow -1$
 \downarrow
floor does not exist

if $\text{left} \Rightarrow n$, ceil does not exist.

(134) First And Last Occurrence

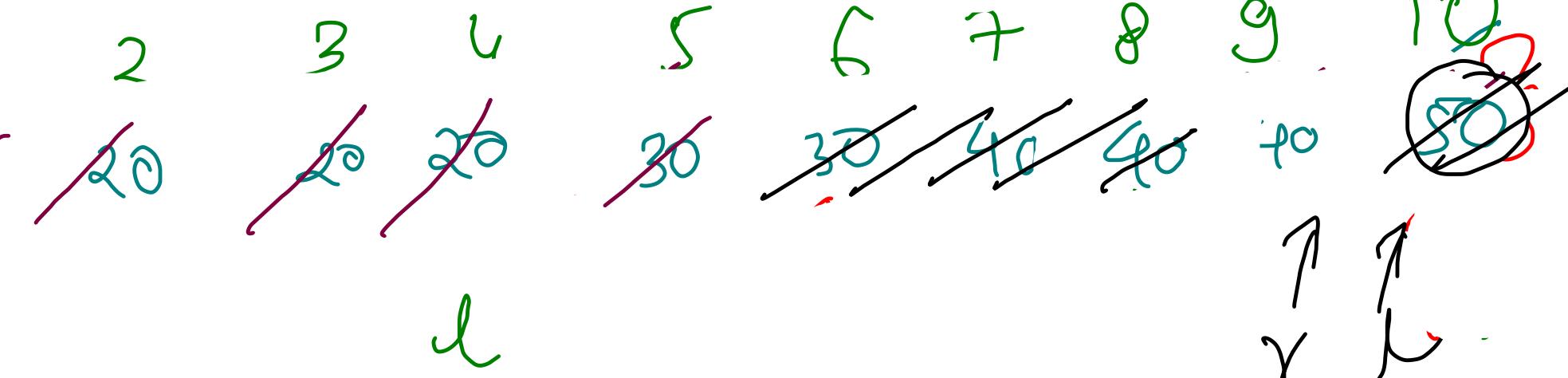
target = 10

~~Count occurrences
if not found return
else return l-f+1~~

f

```
public int firstOcc(int[] nums, int target){
    int left = 0, right = nums.length - 1;
    int ans = -1;
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(nums[mid] == target){
            ans = mid;
            right = mid - 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            // nums[mid] > target
            right = mid - 1;
        }
    }
    return ans;
}
```



l

```
public int lastOcc(int[] nums, int target){
    int left = 0, right = nums.length - 1;
    int ans = -1;
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(nums[mid] == target){
            ans = mid;
            left = mid + 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            // nums[mid] > target
            right = mid - 1;
        }
    }
    return ans;
}
```

first = -1

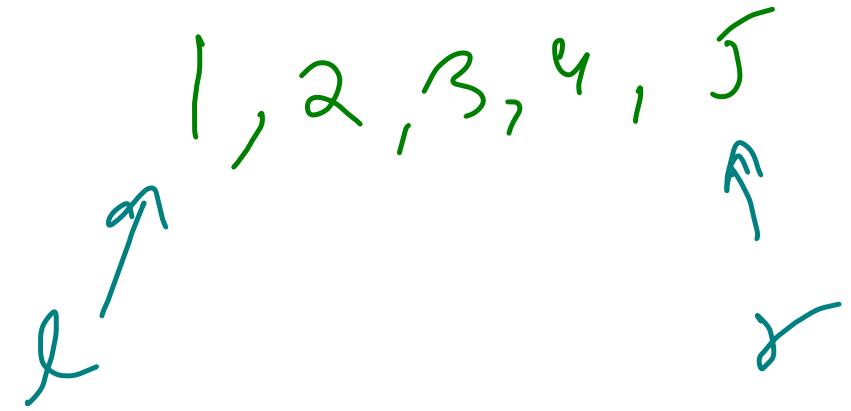
last = -1

7

9

```
public int[] searchRange(int[] nums, int target) {
    int[] ans = {-1, -1};
    if(nums.length == 0) return ans;

    ans[0] = firstOcc(nums, target);
    ans[1] = lastOcc(nums, target);
    return ans;
}
```



$$(1) [l, r] \Rightarrow 5-1+1 \Rightarrow r-l+1$$

$$(2) [l, r] \rightarrow 5-1 \Rightarrow r-l$$

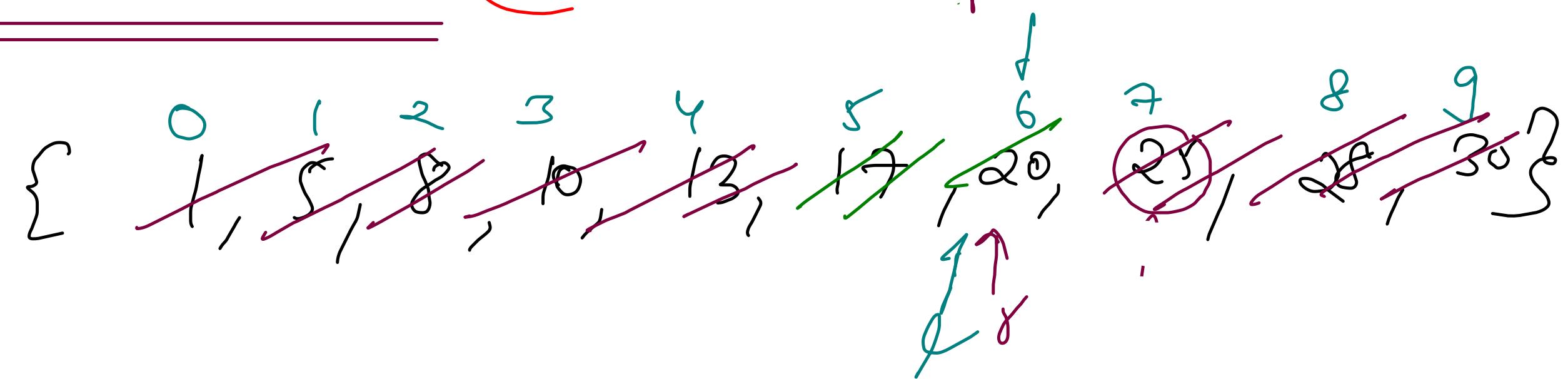
$[l, r)$

$$(3) (l, r) \Rightarrow 5-1-1 \Rightarrow r-l-1$$

Search Insert Position (Q35)

target = 19

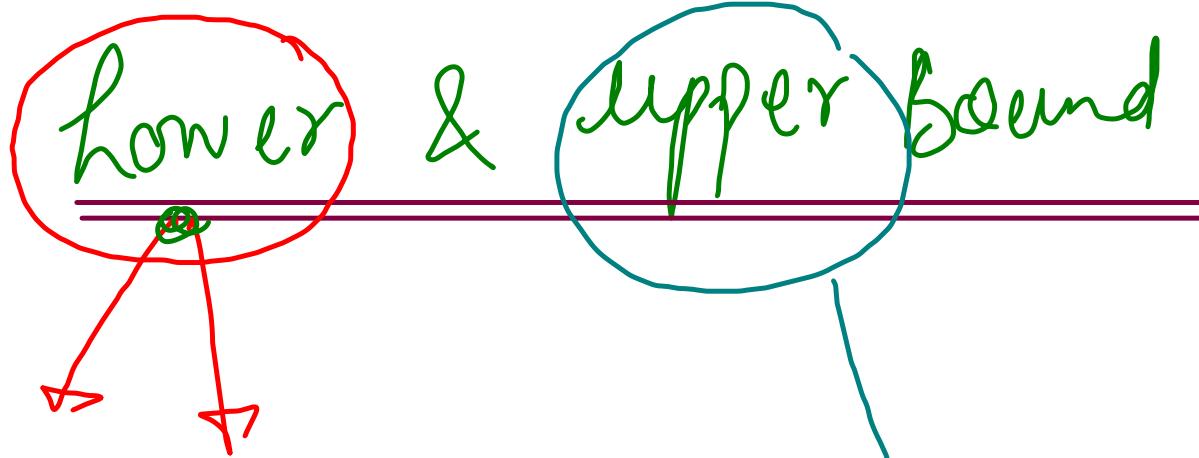
lower bound



on which elements

```
public int searchInsert(int[] nums, int target) {
    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 left;
}
```



if element

found

return

first

occurrence

$\text{arr}[\text{mid}]$
 $\leq \text{target}$

if element

not

found,
then return

just greater
value
(ceil)

$\text{arr}[\text{mid}] > \text{target}$

just greater value

{ceil}

first occurrence of
(ceil)

$$\text{floor} = \text{lb} - 1$$

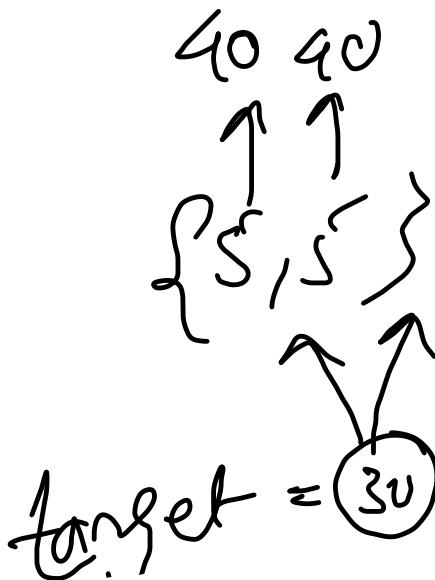
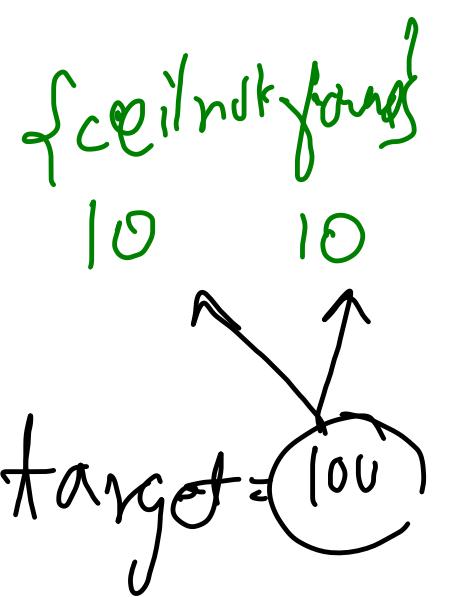
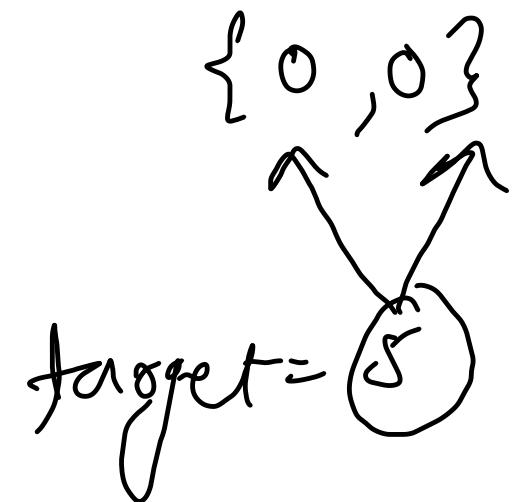
$\{ 10, 10, 10, 20, 20, 40, 40, 40, 50, 50 \}$
 0 1 2 3 4 5 6 7 8 9

```

public static int lowerBound(int[] arr, int target){
    int left = 0, right = arr.length - 1;
    int ans = arr.length;

    while(left <= right){
        int mid = left + (right - left) / 2;

        if(arr[mid] >= target){
            ans = mid;
            right = mid - 1;
        }
        else
            left = mid + 1;
    }
    return ans;
}
  
```



Lecture-②

Starting at 4:10.

① { closest Element
k-closest Elements

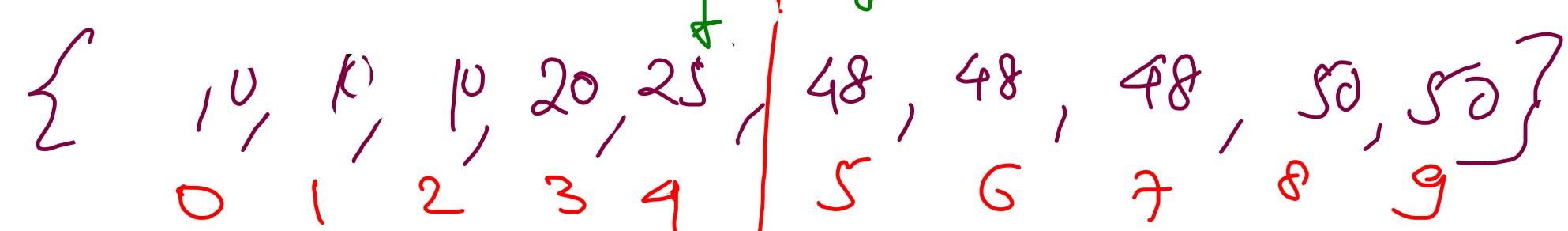
④ } Pivot in Rotated Sorted Array
→ Duplicates not allowed
→ Duplicates allowed
Count Rotations (HW)

② Heaters

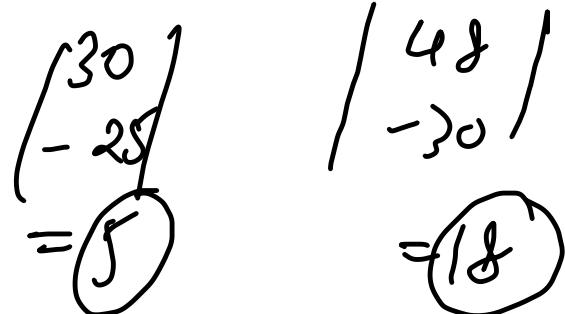
③ { Square Root
Valid Perfect Square
nth Root (HW)

⑤ } Search in Rotated Sorted Array
→ Duplicates not allowed
→ Duplicates allowed (HW)

Closest Element



target = 30

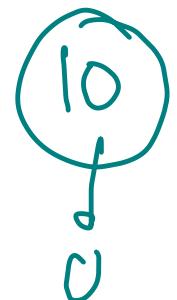


Binary Search on Element

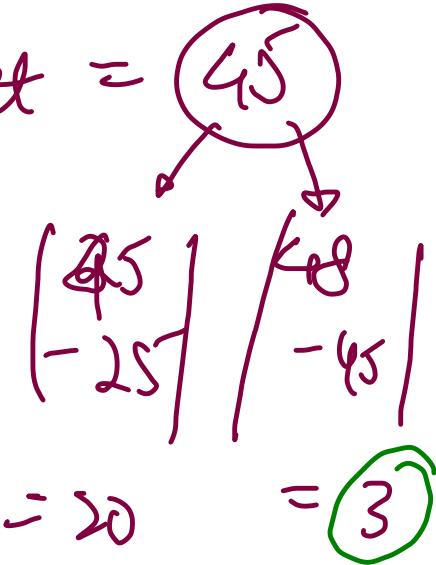
→ if search successful
 ⇒ return ele (diff=0)

→ if search unsuccessful
 ↳ return floor or ceil
 whichever is closer

target =



target = 45



using binary search

```
int second = lowerBound(arr, x);
int first = second - 1;
```

```
List<Integer> res = new ArrayList<>();

while(first >= 0 && second < arr.length && k-- > 0){
    if(Math.abs(arr[first] - x) <= Math.abs(x - arr[second])){
        res.add(arr[first]);
        first--;
    } else {
        res.add(arr[second]);
        second++;
    }
}

while(first >= 0 && k-- > 0){
    res.add(arr[first]);
    first--;
}

while(second < arr.length && k-- > 0){
    res.add(arr[second]);
    second++;
}

Collections.sort(res);
return res;
```

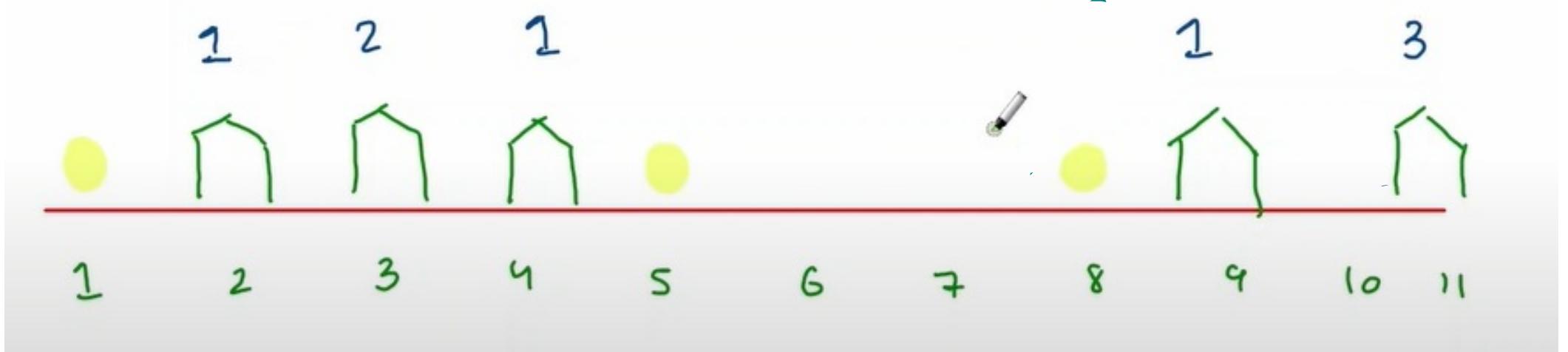
$$\begin{aligned} & \text{lowerbound} \\ & O(\log_2 n) + O(k) \\ & + O(k \log k) \\ & \leq O(\log_2 n + k + k \log k) \end{aligned}$$

↑ two pointers
↑ sorting

Heaters (Q475)

houses = $[3, 9, 2, 4, 11]$

heaters = $[1, 8, 5]$

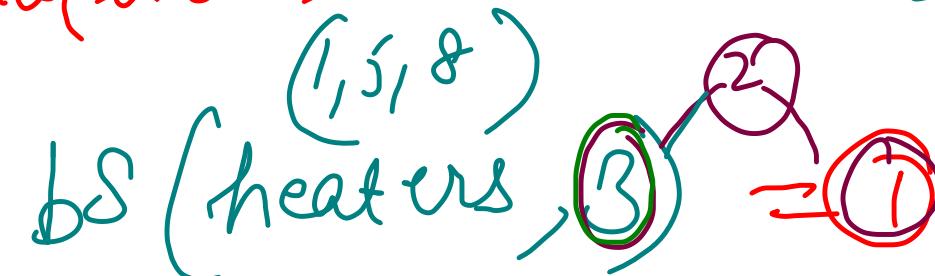


$$h_2 \log h_2 + h_1 * \log h_2 = (h_1 + h_2) \log h_2$$

answer will be maximum of
closest heater from every house

heater \rightarrow sorted

house \rightarrow queries(unsorted)



$$\text{An} = 3$$



```
public static int lowerBound(int[] arr, int target){
    int left = 0, right = arr.length - 1;
    int ans = arr.length;

    while(left <= right){
        int mid = left + (right - left) / 2;

        if(arr[mid] >= target){
            ans = mid;
            right = mid - 1;
        }
        else {
            left = mid + 1;
        }
    }
    return ans;
}
```

```
public static int closest(int[] arr, int target){
    int lb = lowerBound(arr, target);

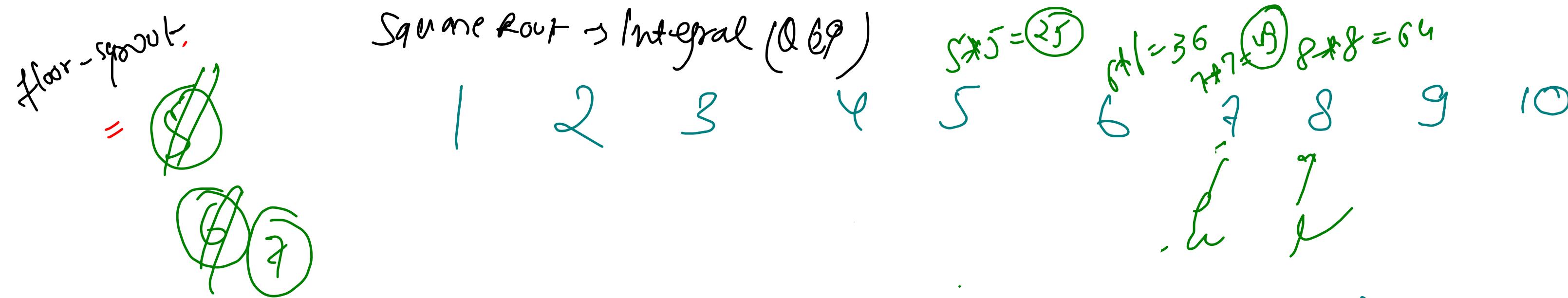
    if(lb == arr.length) return arr[arr.length - 1]; // ceil does not exist
    else if(lb == 0) return arr[0]; // floor not exist

    else if(Math.abs(target - arr[lb]) < Math.abs(target - arr[lb - 1]))
        return arr[lb];
    else return arr[lb - 1];
}

public int findRadius(int[] houses, int[] heaters) {
    Arrays.sort(heaters);

    int max = Integer.MIN_VALUE;
    for(int i=0; i<houses.length; i++){

        int closestHeater = closest(heaters, houses[i]);
        max = Math.max(max, Math.abs(closestHeater - houses[i]));
    }
    return max;
}
```



Square Root \rightarrow Integral (Q69)

1 2 3 4

$5 \times 5 = 25$
 $6 \times 6 = 36$
 $7 \times 7 = 49$
 $8 \times 8 = 64$

9 10

$O(\underline{\log_2 n}) \approx O(1)$

floor sqrt =

$O(\sqrt{n})$

perfect
square

49

11

7

not a
perfect
square

60

11

7

```
public int mySqrt(int x) {
    if(x == 0) return 0;

    long left = 1L, right = x;
    long floorSqrt = 1L;
    while(left <= right){
        long mid = left + (right - left) / 2L;
        long square = mid * mid;

        if(square == x){
            return (int)mid;
        } else if(square < x){
            floorSqrt = mid;
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return (int)floorSqrt;
}
```

Square root \rightarrow Fractional N

precision = 3 p

① floor square root \rightarrow 7

accuracy 0.1

7.0 / 7.1 / 7.2 / 7.3 / 7.4 - - - 7.9

$7+0.1*1$
 $7+0.2 = 7+2*0.1$
 $7+0.3$
 $7+0.1*3$

target = 60

floor + accuracy \uparrow (0 \rightarrow 9)

② 0.01

7.0 / 7.01 / 7.02 / 7.03 - - - 7.09

$7+0.01*1$
 $7+0.01*2$
 $7+0.01*9$

0.001

7.000 / 7.001 / 7.002 - - -

floor

precision

```
double sqroot(int n, int precision)
{
    double ans = floor_sqroot(n);
    double j = 0.1;
    while (precision--)
    {
        while (ans * ans <= n) ans += j;
        ans = ans - j;
        j = j / 10;
    }
    return ans;
}
```

$O(\log_2 N + P * 10)$

Starting
at
8:10

Binary Search lecture - ③

Rotated Sorted Array, Bitonic Array

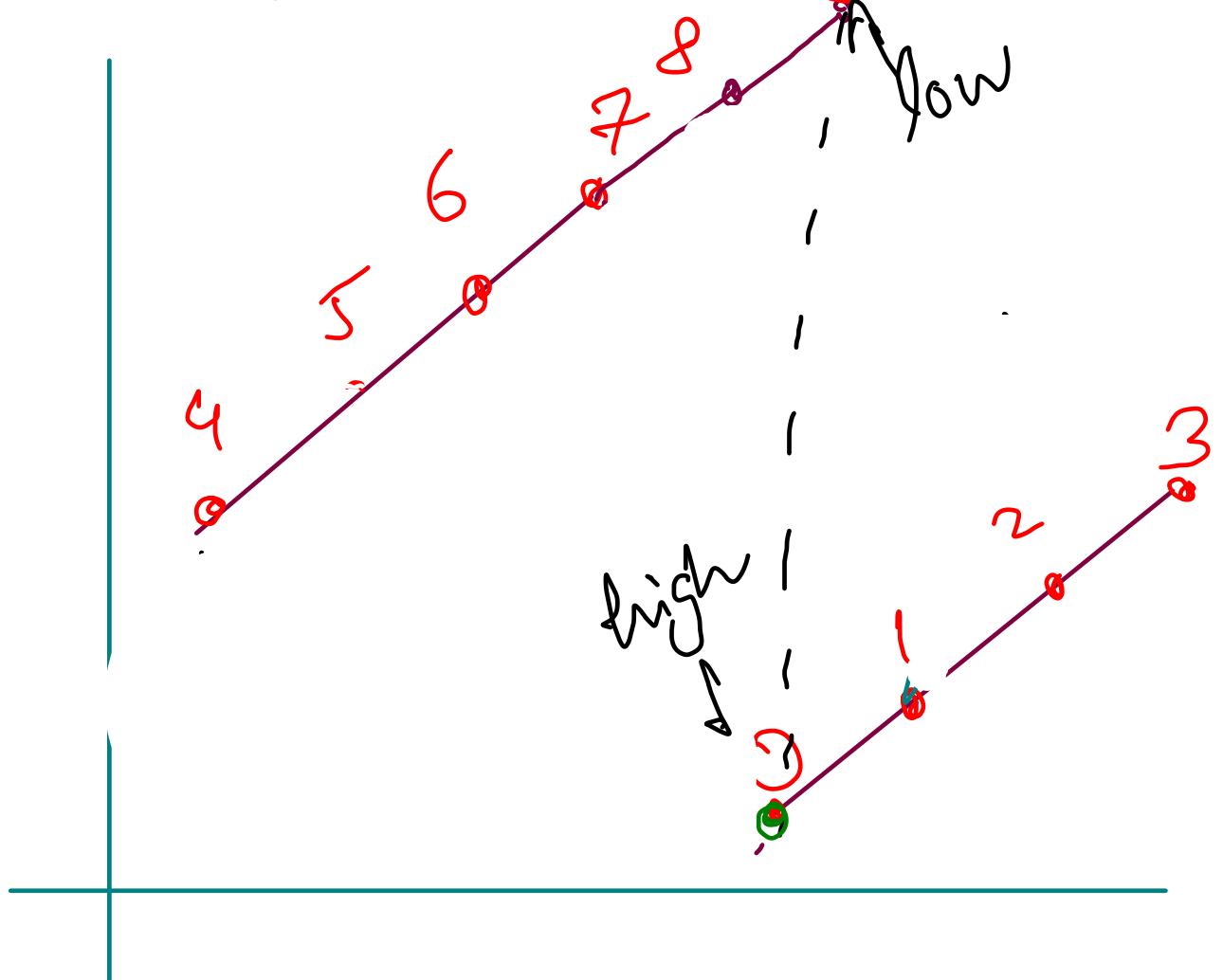
Rotated Sorted Array

- ① {
 - Pivot or Minimum - I (IB)
 - Pivot or Minimum - I
 - Pivot or Minimum - II
- ② {
 - Find Rotation Count
 - Find Rotation Count
- ③ {
 - Searching - I (IB)
 - Searching - I
 - Searching - II

Bitonic Array

- ④ {
 - Maximum Element (IB)
 - Maximum Element
- ⑤ {
 - Search Element
 - Search Element
- ⑥ {
 - Peak Element
 - Peak Element

Modified Binary Search



Time Complexity: $O(\log n)$

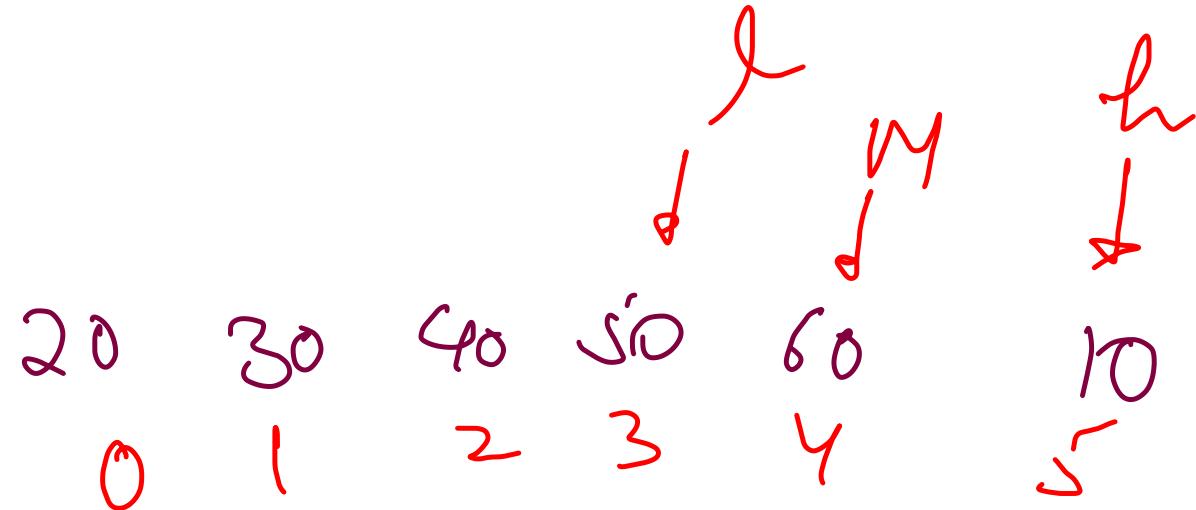
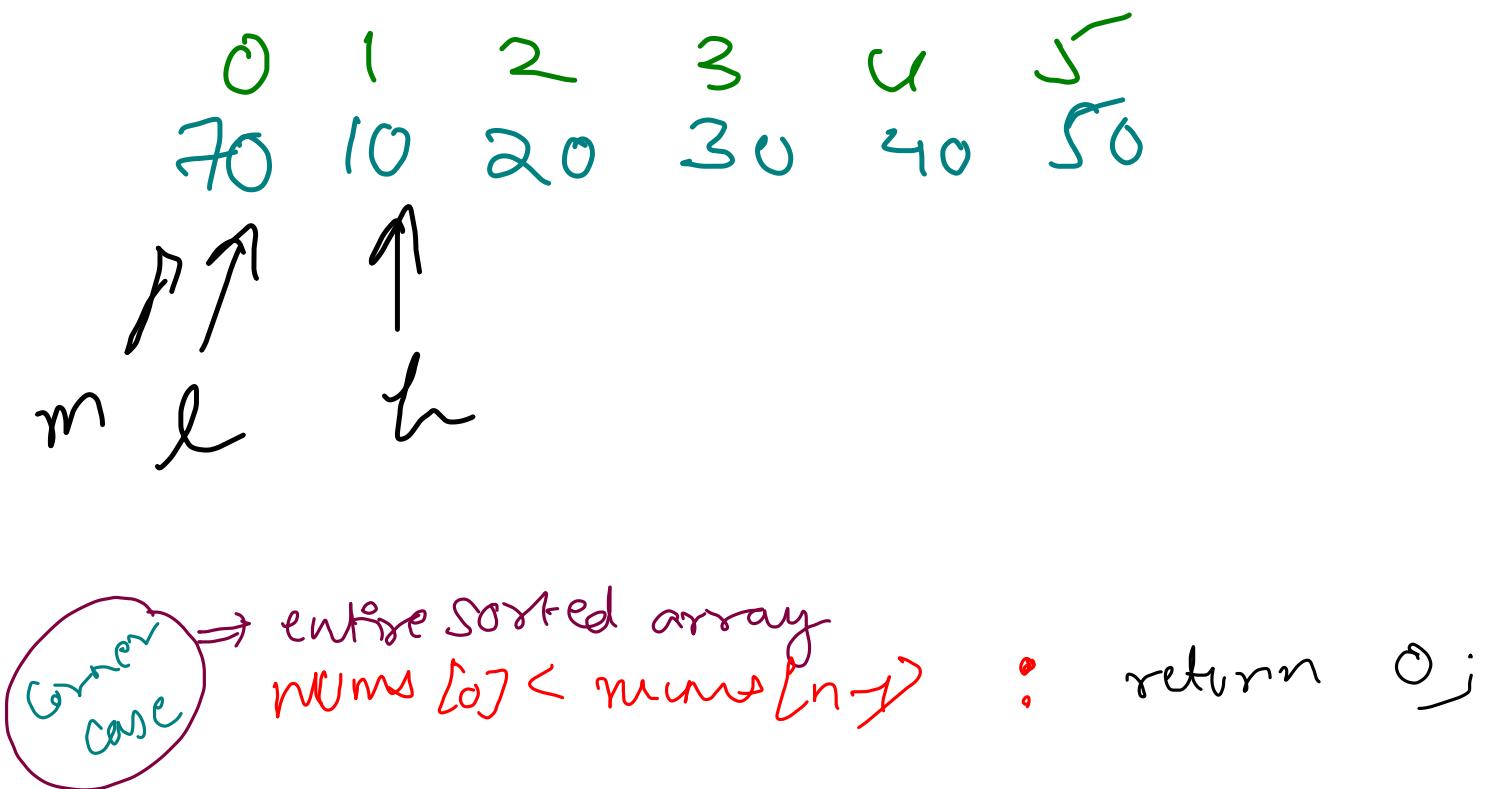
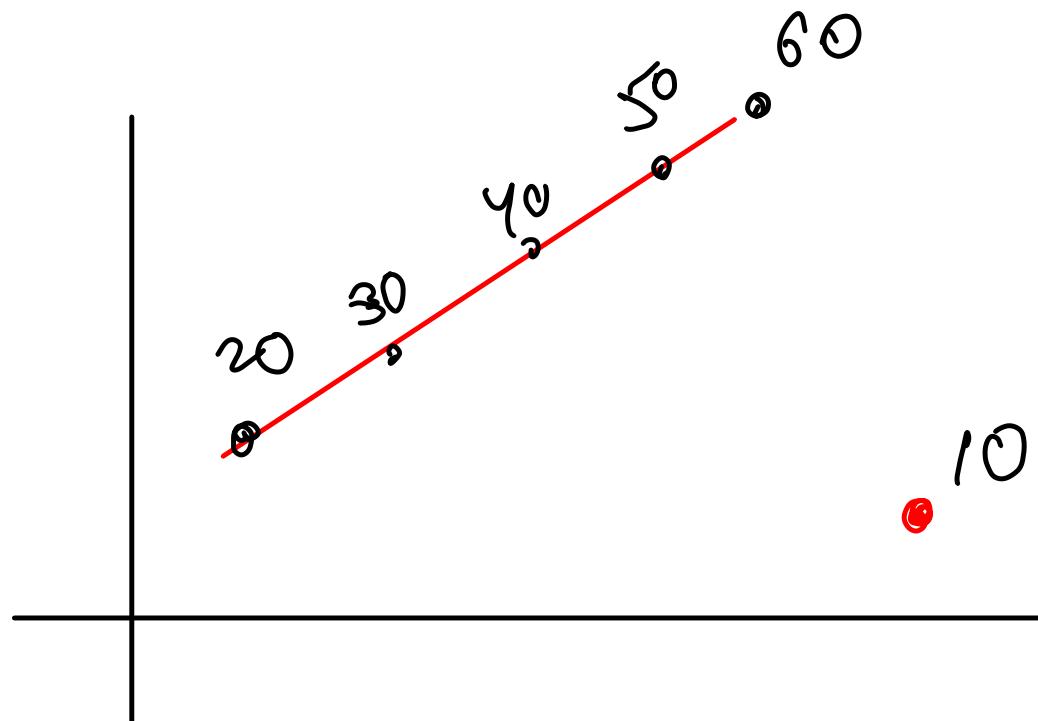
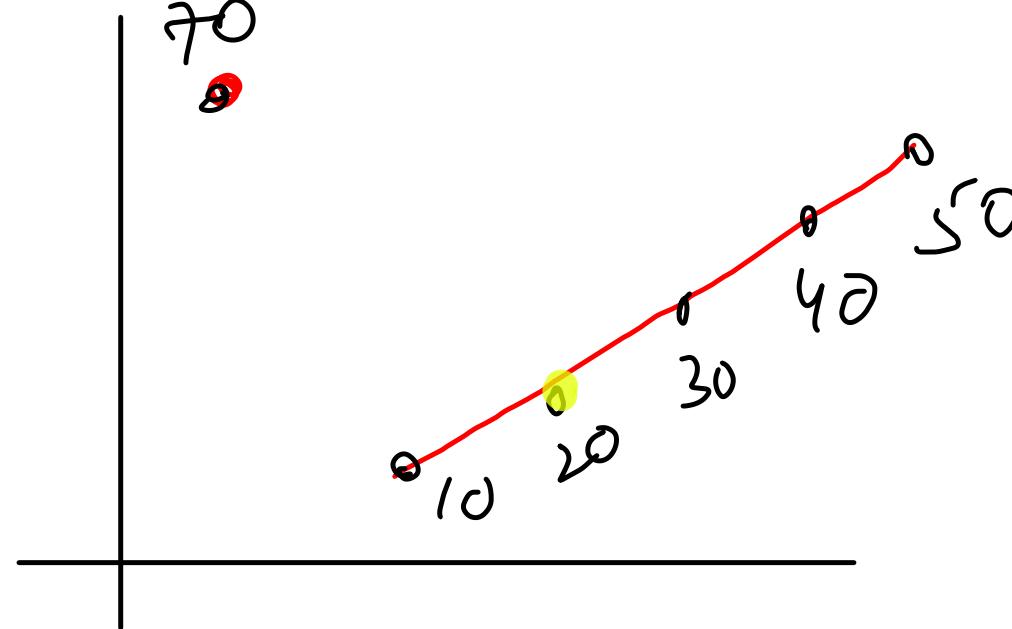
if ($\text{mid} == \text{target}$)
 return mid ;

if ($\text{mid} == \text{left}$)
 return mid ;

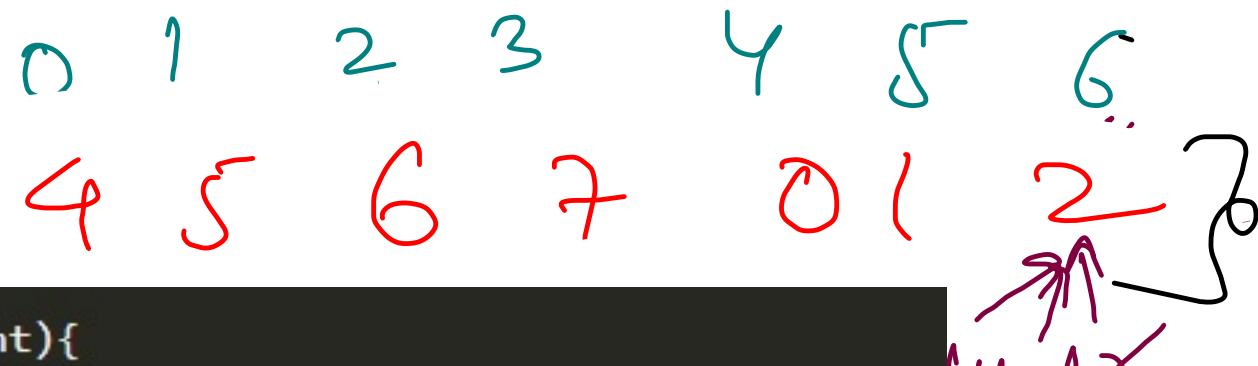
if ($\text{nums}[\text{low}] < \text{nums}[\text{mid}]$)
 \Rightarrow left range is sorted
 $\Rightarrow l = m + 1$

else
 \Rightarrow left range is unsorted
 $\Rightarrow r = m - 1$

$\text{mid} > \text{left}$
 $\text{mid} > \text{right}$

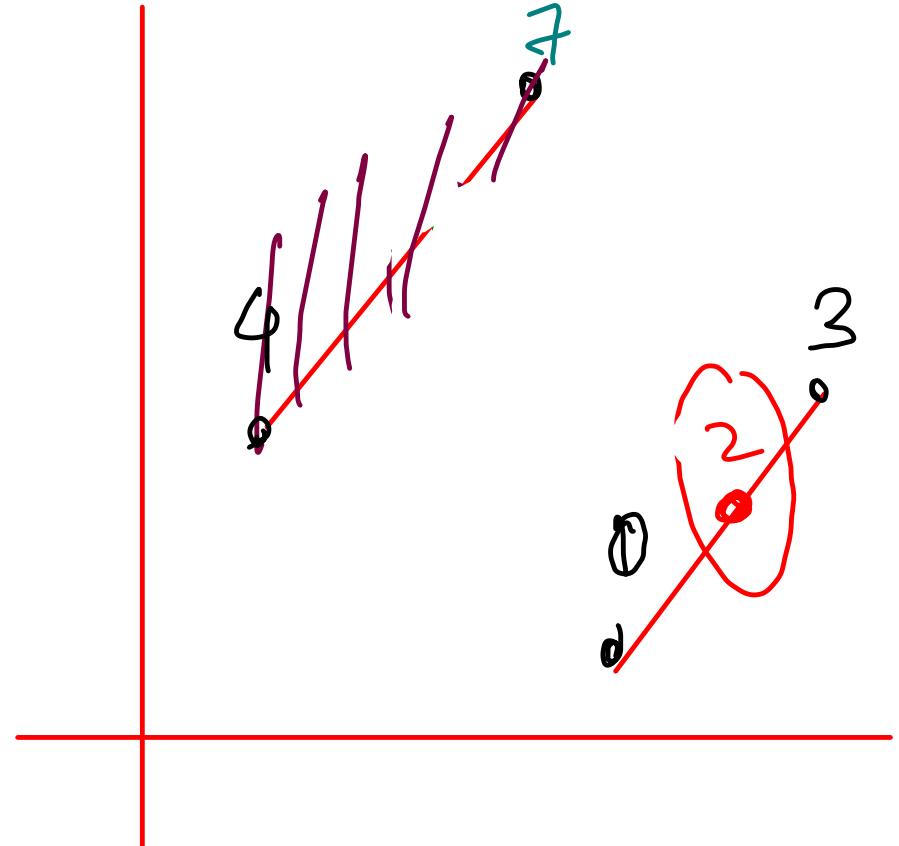


Searching in Rotated Sorted Array



```
while(left <= right){  
    int mid = left + (right - left) / 2;  
  
    if(arr[mid] == target) return mid;  
  
    if(arr[left] <= arr[mid]){  
        // left range is sorted  
        if(arr[left] <= target && target < arr[mid]){  
            right = mid - 1;  
        } else {  
            left = mid + 1;  
        }  
    }  
  
    else if(arr[mid] < arr[right]){  
        // right range is sorted  
        if(arr[mid] < target && target <= arr[right]){  
            left = mid + 1;  
        } else {  
            right = mid - 1;  
        }  
    }  
}
```

target = 3



	0	1	2	3	4	5	6	
0	0	0	0	0	0	0	1	$fp = 6$
1	0	0	0	0	1	1	1	$fp = 4$
2	0	0	0	0	0	1	1	
3	0	0	0	1	1	1	1	$fp = 3$
4	0	1	1	1	1	1	1	$fp = 1$
f	0	0	0	0	0	0	0	

$$n = 6$$

$$m = 7$$

$$\text{no of ones} = m - fp$$

$$\text{ans} = \emptyset / \{3\}$$

6

left case \Rightarrow logn

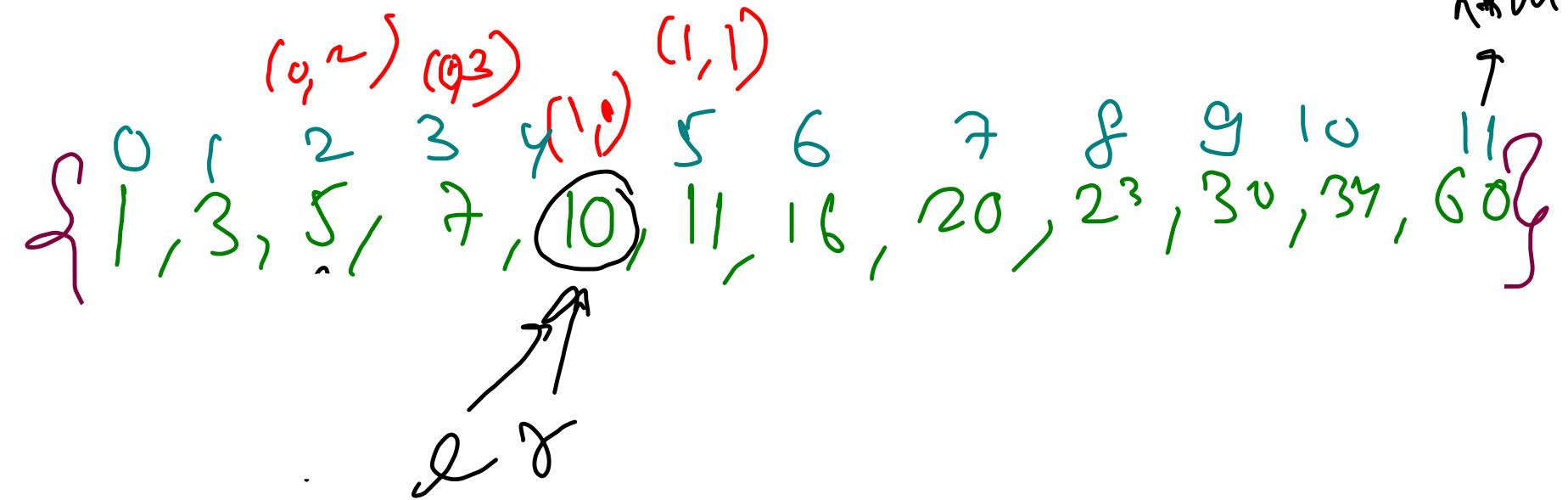
worst case $\Rightarrow n \log m$

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

```
int rowWithMax1s(int arr[][][], int n, int m) {  
    int right = m - 1;  
    int ans = -1, noOfOnes = 0;  
    for(int i=0; i<n; i++){  
        int transitionPt = transitionPoint(arr[i], 0, right);  
        if(transitionPt != -1 && m - transitionPt > noOfOnes){  
            ans = i;  
            noOfOnes = m - transitionPt;  
            right = transitionPt - 1;  
        }  
    }  
    return ans;  
}
```

target = 10

0	1	2	3
1	3	5	7
4	5	6	7
10	11	16	20
8	9	10	11
23	30	34	60



$\mathcal{O}(\log(n \cdot m))$

(c) cell $\xrightarrow{\text{row } (c/m)}$ arr[r][c]
 $\xrightarrow{\text{col } (c \% m)}$

```
public boolean searchMatrix1(int[][] matrix, int target){
    int n = matrix.length, m = matrix[0].length;
    int left = 0, right = n * m - 1;

    while(left <= right){
        int midCell = left + (right - left) / 2;

        int rowIdx = midCell / m;
        int colIdx = midCell % m;

        if(matrix[rowIdx][colIdx] == target){
            return true;
        }

        if(matrix[rowIdx][colIdx] < target){
            left = midCell + 1;
        }

        else right = midCell - 1;
    }
    return false;
}
```