Binary Search

- Linear Search {unsorted}

10, 5, 7, 9, 2, 6

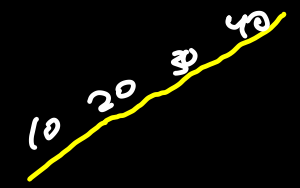
target = 9

Worst case ⇒ O(n)
time

Space complexity ⇒ O(1)
inplace

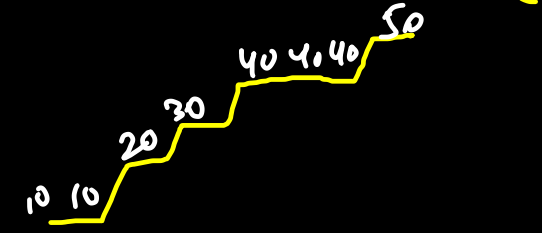
- monotonic array {sorted} optimizatn

→ strictly increasing

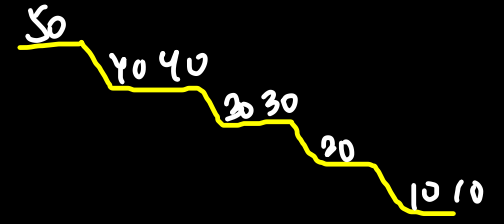


→ strictly decreasing

→ increasing

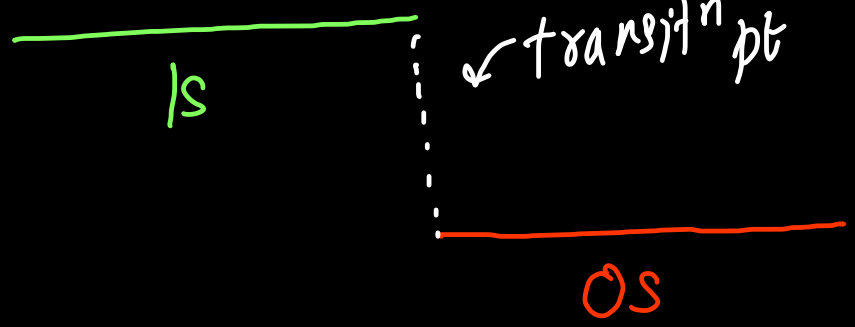


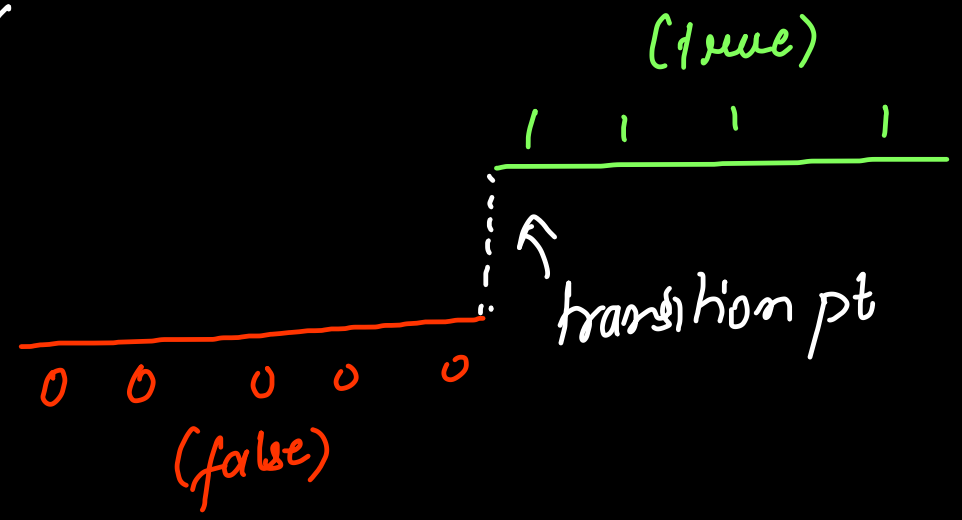
→ decreasing



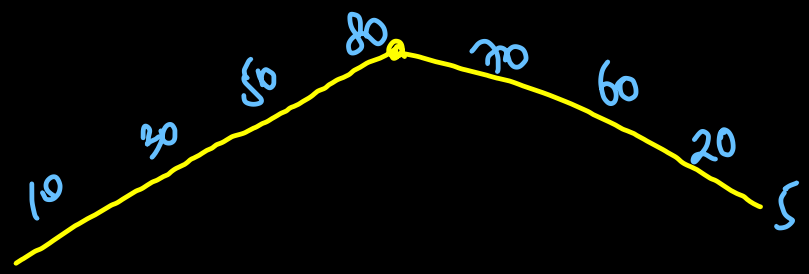
- Variations of monotonic array

① Binary / boolean (sorted)

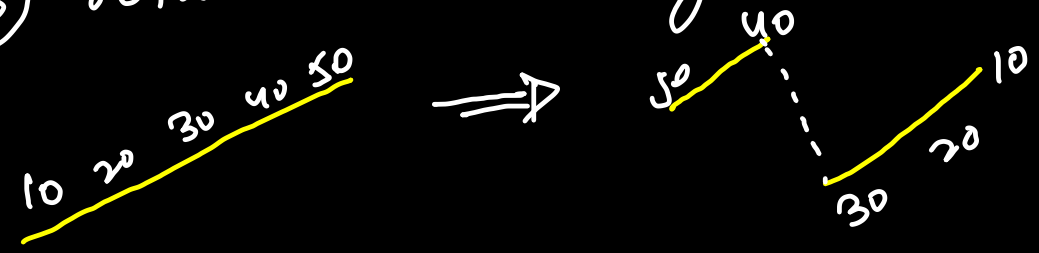




② Bitonic array / peak array



③ rotated sorted array



④ Sorted Matrix

⑤ Infinite Sorted

⑥ Nearly Sorted

20 10 40 30 60 50

Binary Search (LC 704) Strictly increasing

eg 1

10	20	40	60	70	90	100	120	140	150
0	1	2	3	4	5	6	7	8	9

target = 100

Search space ⇒ from left to right

l = 0, r = 9, m = (0+9)/2 = 4

70 < 100 ⇒ left = mid+1

arr[m] < target

l = 5, r = 9, m = (5+9)/2 = 7

120 > 100 ⇒ right = mid-1

arr[m] > target

l = 5, r = 6, m = (5+6)/2 = 5

90 < 100 ⇒ left = mid+1

arr[m] < target

l = 6, r = 6, m = (6+6)/2 = 6

100 = 100 ⇒ return mid;

eg2

10	20	40	60	70	90	100	120	140	150
0	1	2	3	4	5	6	7	8	9

right

left

target : 30

$l = 0,\ r = 9,\ m = (0+9)/2 = 4$

$70 > 30 \Rightarrow$ discard right

right = mid-1

$l = 0,\ r = 3,\ m = (0+3)/2 = 1$

$20 < 30 \Rightarrow$ discard left

left = mid+1

$l = 2,\ r = 3,\ m = (2+3)/2 = 2$

$40 > 30$

$\Rightarrow$ discard right

right = mid-1

target search unsuccessful

$\Rightarrow$ **left > right**

```java
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;
        } else if(nums[mid] < target){
            left = mid + 1;
            // discard left to mid region
        } else {
            right = mid - 1;
            // discard mid to right region
        }
    }

    return -1; // search unsuccessful
}
```

Space = O(1) constant / inplace

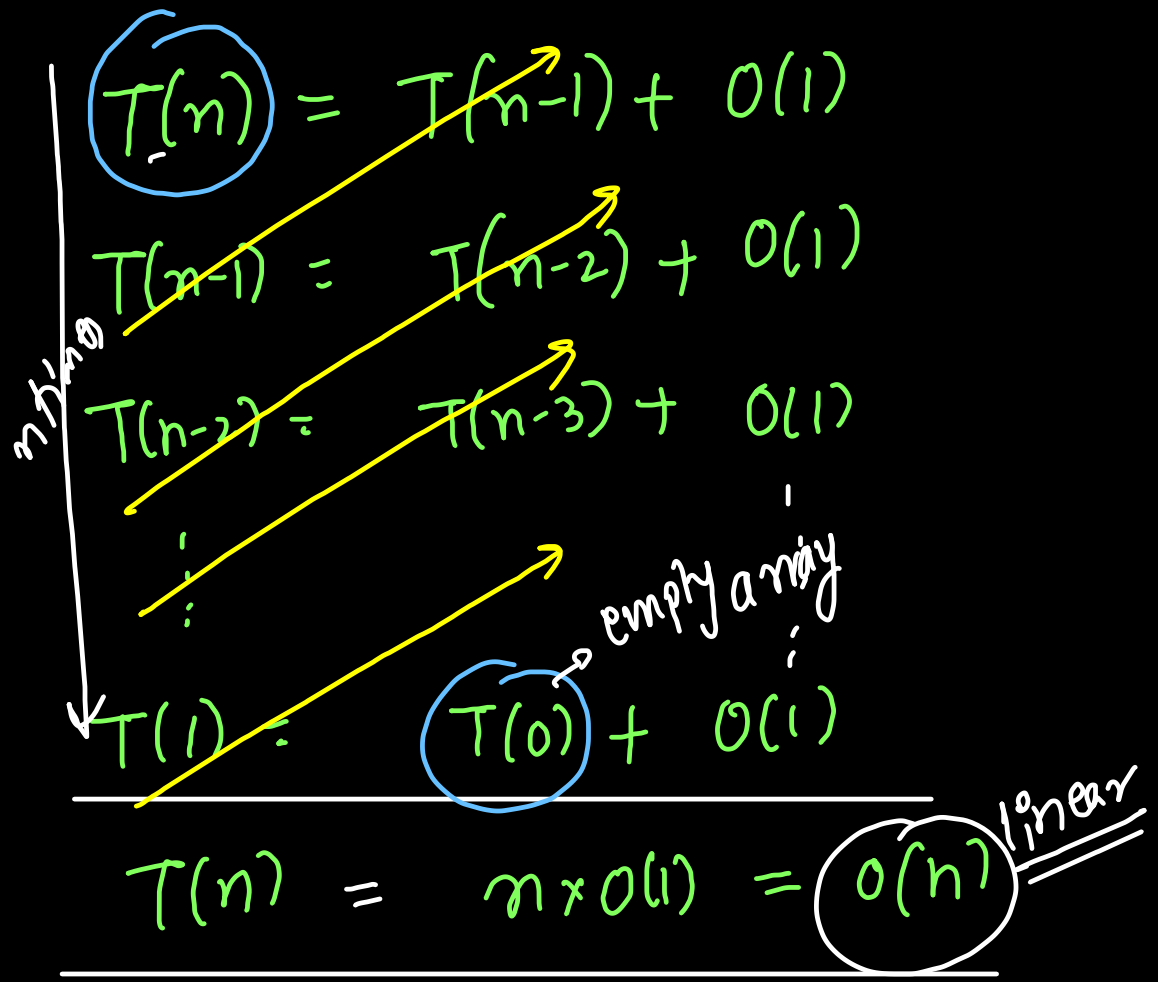
Time <= O(log n) logarithmic

constant

RC 704)

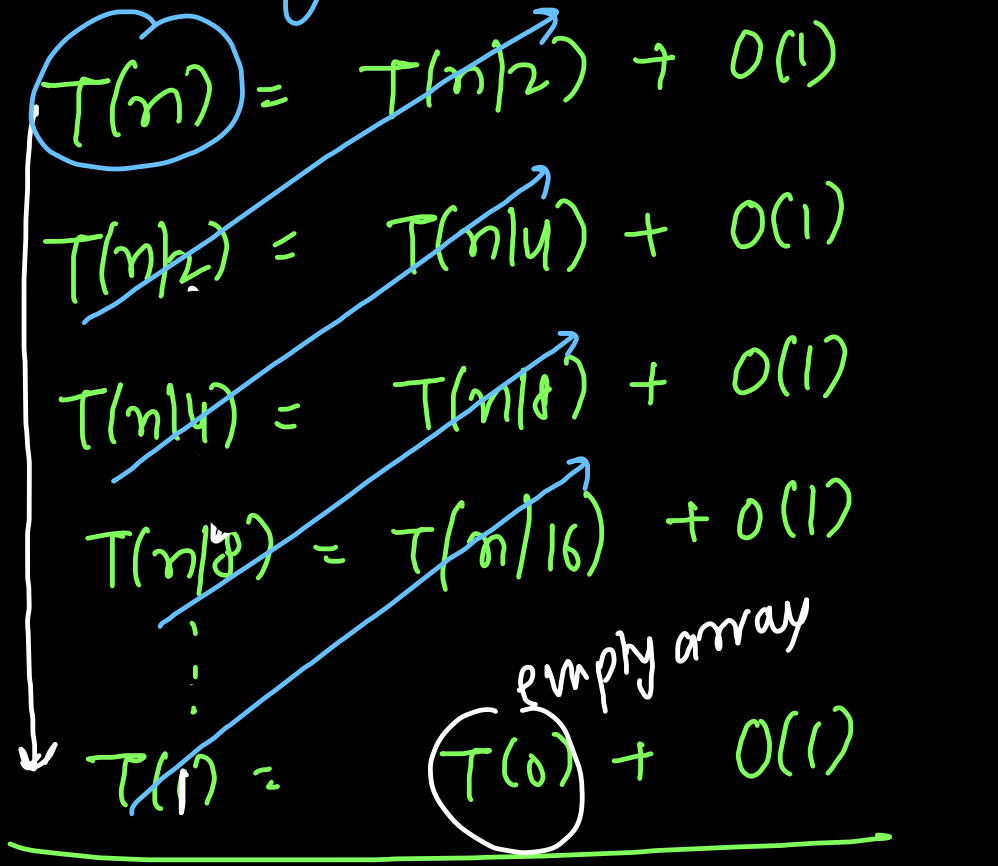
Recurrence relation

Linear search



$T(n) = n \times O(1) = O(n)$ linear

Binary search



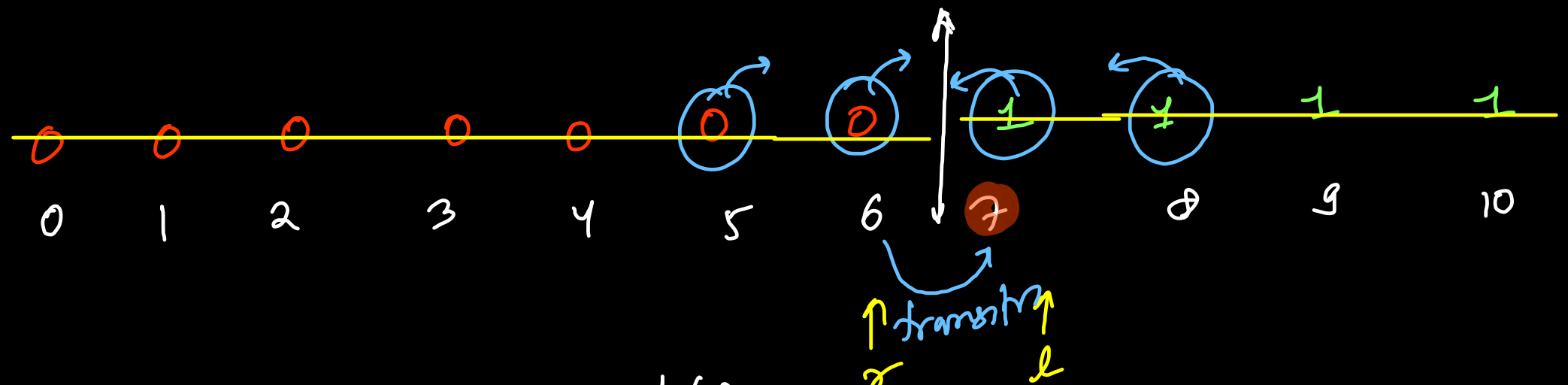
$T(n) = \log_2 n \times O(1) = O(\log_2 n)$

(almost constant)

input size (N)	logarithmic $O(\log_2 N)$	<<<	linear $O(N)$
$n = 2$	$\log_2 2 = 1$		2
$n = 2^2 = 4$	$\log_2 4 = 2$		4
$n = 2^3 = 8$	$\log_2 8 = 3$		8
$n = 2^{10}$	$\log 2^{10} = 10$		1024
$n = 2^{31}$	$\log 2^{31} = 31$		$2^{31} = 2 \times 10^9$

Binary Sorted Array

0's 1's Array Transition Point



$l = 0, r = 10, m = 5 \Rightarrow$ left discard (0)

$l = 6, r = 10, m = 8 \Rightarrow$ right discard (1)

$l = 6, r = 7, m = 6 \Rightarrow$ left discard (0)

$l = 7, r = 7, m = 7 \Rightarrow$ left discard (1)

return left;

```
int transitionPoint(int arr[], int n) {
    if(arr[n - 1] == 0) return -1;
    // no 1s in entire array -> no transition point

    int left = 0, right = n - 1;
    while(left <= right){
        int mid = (left + right) / 2;

        if(arr[mid] == 0){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return left;
}
```

Time ⇒ $O(\log_2 N)$

$N \rightarrow N/2 \rightarrow N/4 \rightarrow N/8 \rightarrow \cdots 1$

Space ⇒ $O(1)$

Constant extra/inplace

Google interview

Lc 278) First Bad Version

$n = 11$

logical (binary sorted array)

brute force ⇒ linear search $\{1 \rightarrow N\}$

worst API calls ⇒ $O(N)$

binary search ⇒ optimized

worst API calls ⇒ $O(\log_2 N)$

Time: $O(\log_2 n)$

Space: $O(1)$ inplace

$(left + right)/2$ ⇒ overflow → left = 1, right = $2^{31} - 1 = \infty$

$(l + r)/2 = (1 + 2^{31} - 1)/2$

$= 2^{31}/2$

$= -2^{31}/2$

$= -2^{30}$ garbage

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

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

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

    return left;
}
```

$left + (right - left)/2$ ⇒ it will never overflow

$1 + (2^{31} - 1 - 1)/2$

$= 1 + (2^{31} - 2)/2 = 1 + 2^{30} - 1$

$= 2^{30}$

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

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

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

    return left;
}
```

approach ①

```java
public int firstBadVersion(int n) {
    long left = 1, right = n;

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

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

    return (int)left;
}
```

approach ②

LC 35) Search Insert Position

"Lower Bound"

10	20	20	30	30	30	40	50	50	50	50	60	
0	1	2	3	4	5	6	7	8	9	10	11	12

target = 75 ⇒ 12

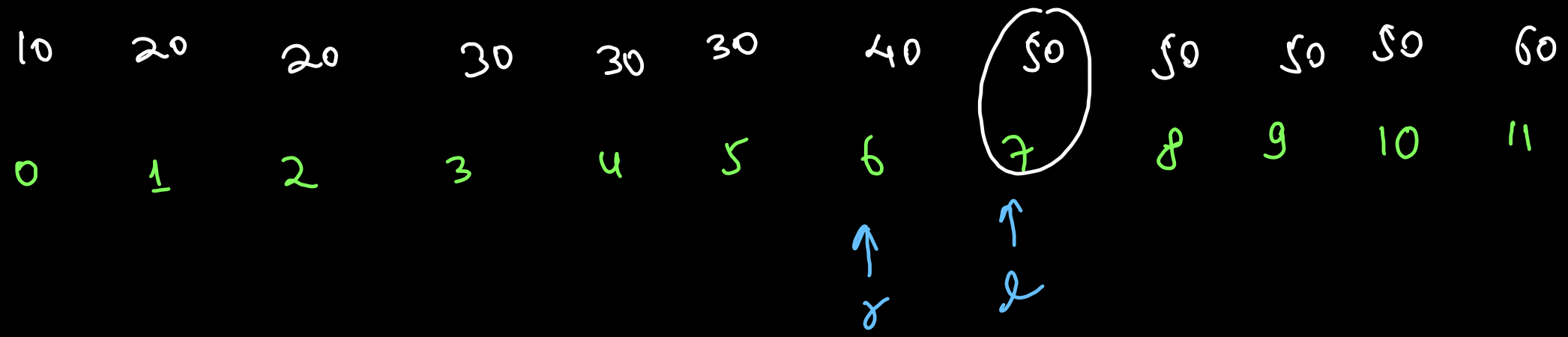
target = 5 ⇒ 0

target = 45 ⇒ 7

target = 40 ⇒ 6

target = 20 ⇒ 1

target = 70 ⇒ 7



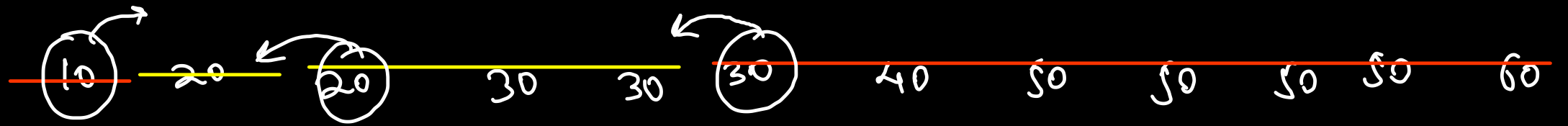
target = 45

l = 0, r = 11, m = 5

l = 6, r = 11, m = 8

l = 6, r = 7, m = 6

l, 7, r = 7, m = 7



LowerBound

arr(m) == t

⇒ r = m - 1

target = 20

l = 0, r = 11, m = 5

l = 0, r = 4, m = 2

l = 0, r = 1, m = 0

l = 1, r = 1, m = 1

l = 1, r = 0

arr(m) < t

⇒ l = m + 1

arr(m) > t

⇒ r = m - 1

```java
public int searchInsert(int[] nums, int target) {
    int left = 0, right = nums.length - 1;

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

        if(nums[mid] == target){
            // first occurence
            right = mid - 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return left;
}
```

Time :- $O(\log_2 n)$ {divide & conquer}

Space :- $O(1)$ {constant / inplace}

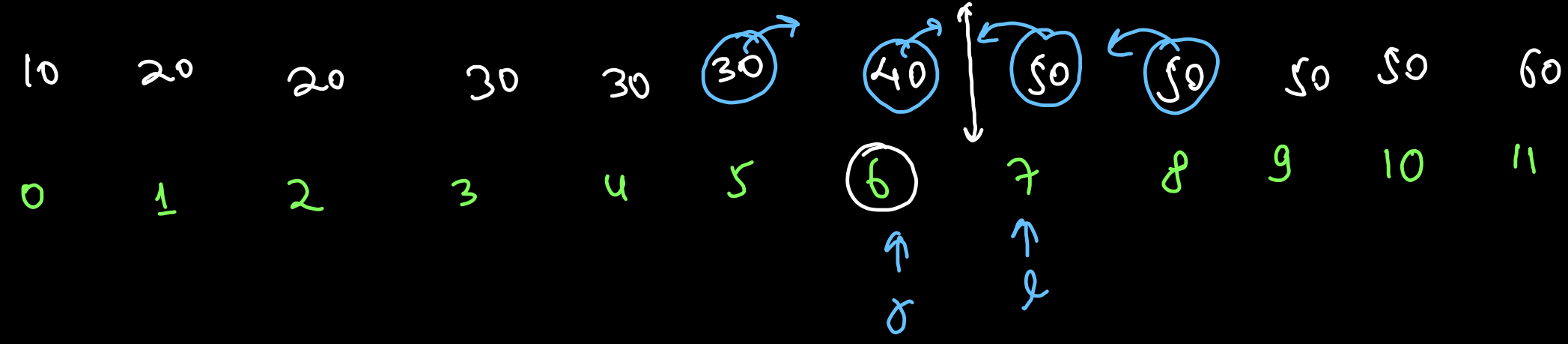
Lower Bound

target is found

⇒ return first occurrence

target is not found

⇒ just greater value's first occ (ceil)



floor(45) = 6

floor(30) = 2

floor(80) = 11

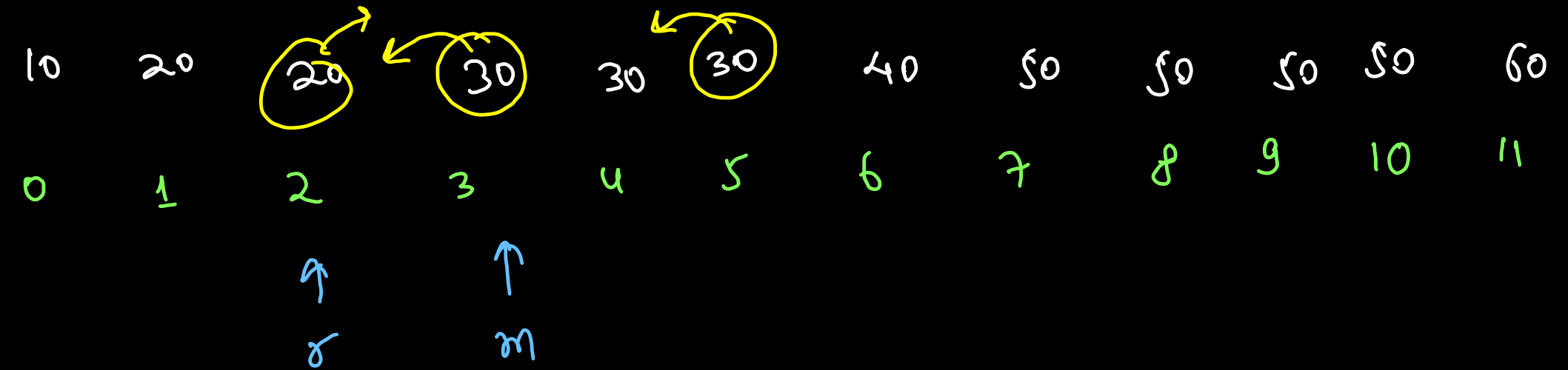
floor(10) = -1

$30 < 45 \Rightarrow l = m+1$

$50 > 45 \Rightarrow r = m-1$

$40 < 45 \Rightarrow l = m+1$

$50 > 45 \Rightarrow r = m-1$



target (30) = floor ⇒ right (2)

l = 0, r = 11, m = 5 ⇒ 30 = 30

l = 0, r = 4, m = 2, ⇒ 20 < 30

floor

```java
public int searchInsert(int[] nums, int target) {
    int left = 0, right = nums.length - 1;

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

        if(nums[mid] == target){
            // first occurence
            right = mid - 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return left;
}
```

right

ceil or upper bound ⇒ just greater value

ceil ⇓ value

upper bound ⇓ index

10	20	20	30	30	30	40	50	50	50	50	60
0	1	2	3	4	5	6	7	8	9	10	11

ceil(45) = 50

ub(45) = 7th

ceil(30) = 40

ub(30) = 6th

ceil(5) = 10

ub(5) = 0th

ceil(70) = -1

ub(70) = -1

eg: 10 20 20 30 30 30 40 50 50 50 50 60

0 1 2 3 4 5 6 7 8 9 10 11

cell

r

l

m

target = 45

eg: 10 20 20 30 30 30 40 50 50 50 50 60

0 1 2 3 4 5 6 7 8 9 10 11

cell

r

l

m

target = 30

ceil / upper bound

```java
public static int ceilingInSortedArray(int n, int target, int[] nums) {
    int left = 0, right = nums.length - 1;

    // ceil or upper bound
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(nums[mid] == target){
            left = mid + 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return left;
}
```

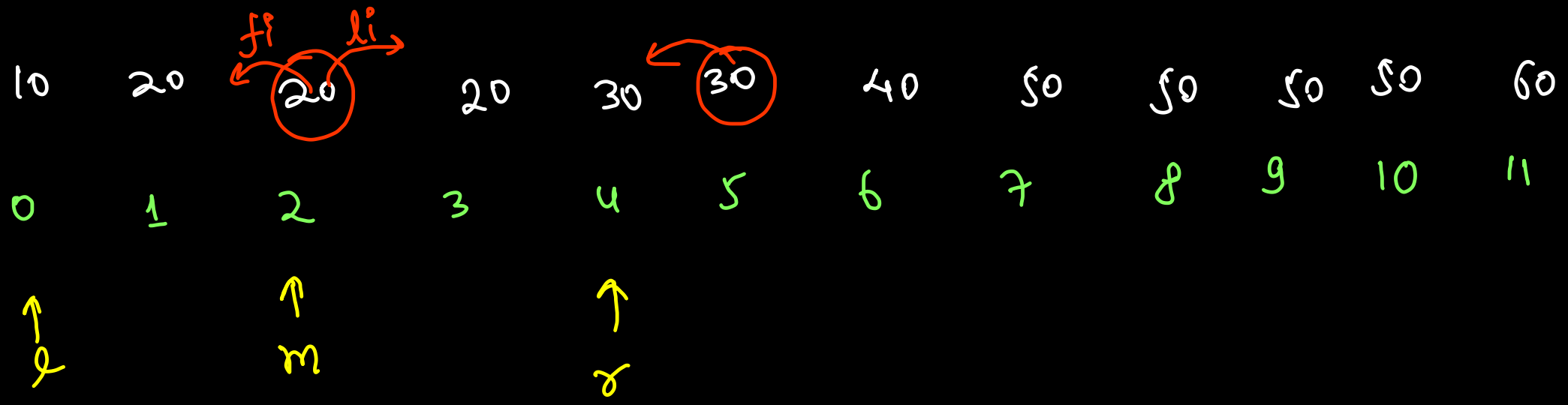
#first occurrence & last occurrence
(LC 34)

10	20	20	30	30	30	40	50	50	50	50	60
0	1	2	3	4	5	6	7	8	9	10	11

target ~~multiple~~ 30 $\Rightarrow$ $fi = 3$, $li = 5$ $[3,5]$

~~Single~~ 40 $\Rightarrow$ $fi = 6$, $li = 6$ $[6,6]$

~~zero~~ 45 $\Rightarrow$ $fi = -1$, $li = -1$ $[-1,-1]$

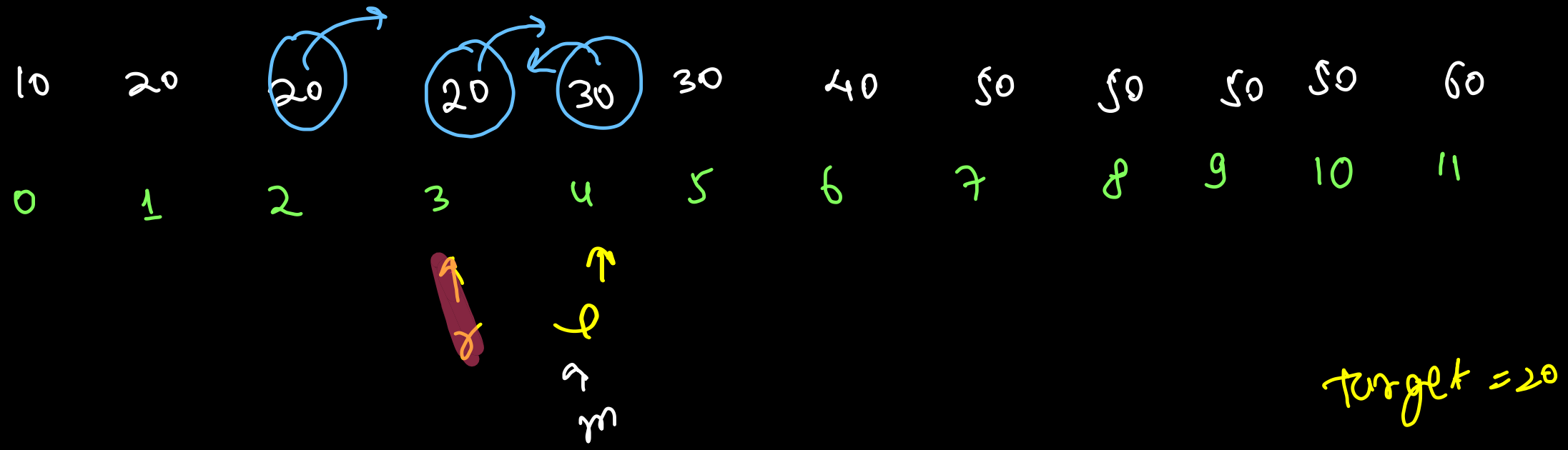


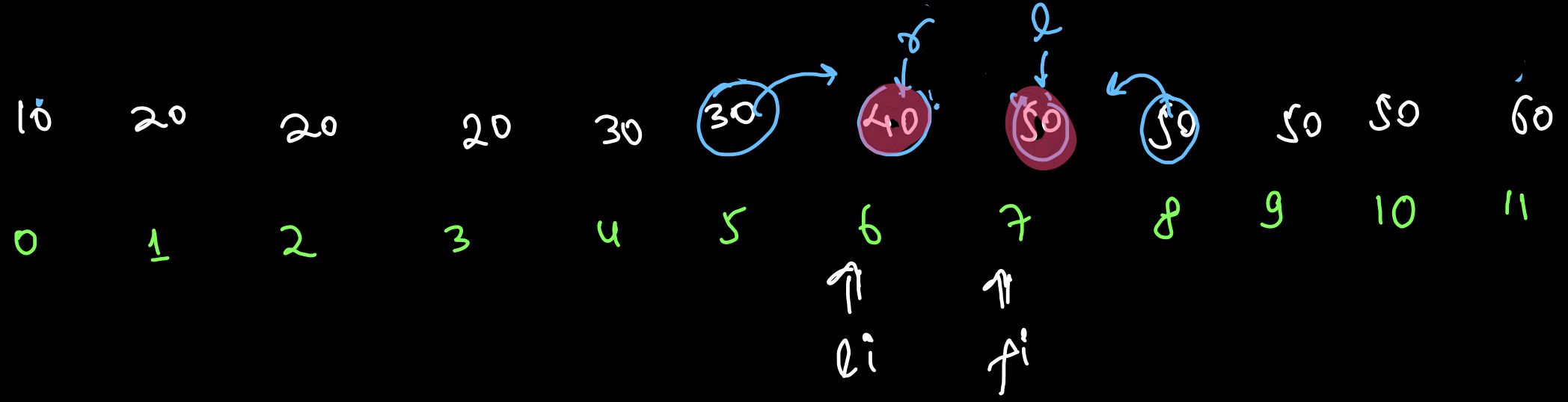
target = 20

$30 > 20 \Rightarrow r = m-1$

$20 = 20 \Rightarrow$ fi $\Rightarrow r = m-1 \Rightarrow$

li $\Rightarrow l = m+1$


10 20 20 20 30 30 40 50 50 50 50 60
0 1 2 3 4 5 6 7 8 9 10 11
target = 20
last occurence



target = 45

```java
public int firstOccurence(int[] nums, int target){
    int left = 0, right = nums.length - 1;

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

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

    return left;
}
```

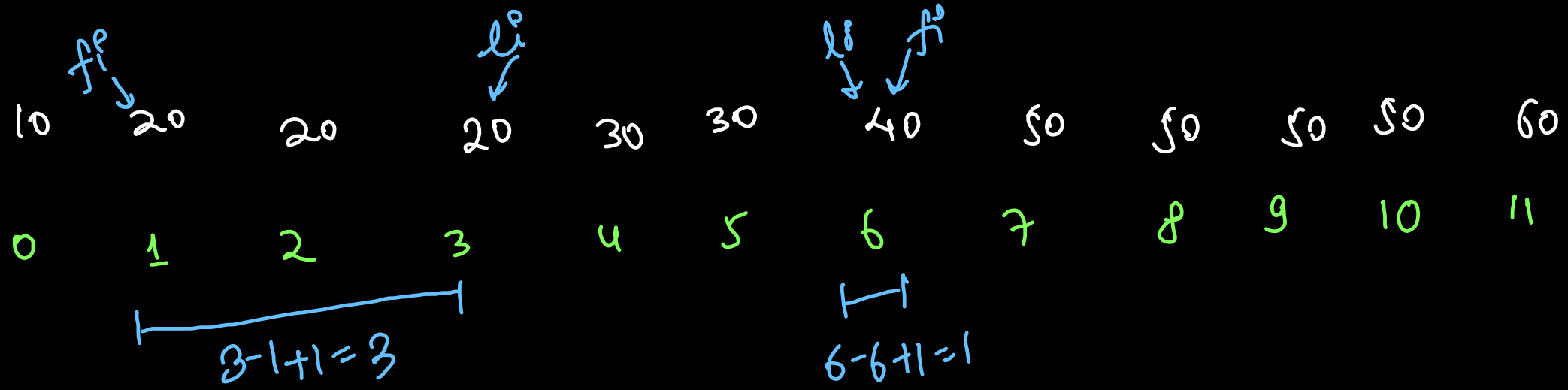
```java
public int lastOccurence(int[] nums, int target){
    int left = 0, right = nums.length - 1;

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

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

    return right;
}
```

```java
public int[] searchRange(int[] nums, int target) {
    int fi = firstOccurence(nums, target);
    int li = lastOccurence(nums, target);

    if(fi > li) return new int[]{-1, -1};
    return new int[]{fi, li};
}
```

→ $\log n$

→ $\log n$

$O(2 \log n)$ Time, $O(1)$ space



Count (20) = 3

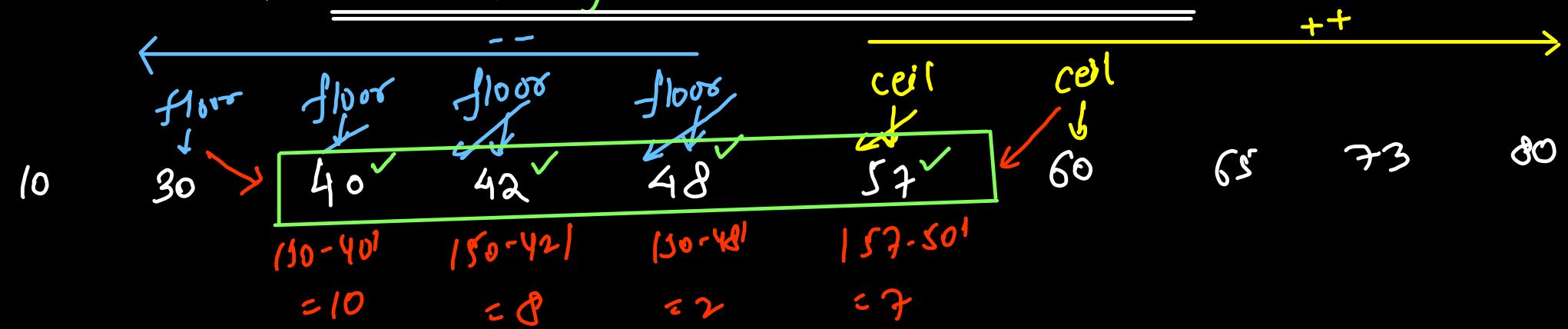
Count (50) = 4

Count (40) = 1

Count (45) = 0 $\Rightarrow$ corner case: $f_i = l_i = -1$ $\Rightarrow$ count = 0

answer $\Rightarrow$ $l_i - f_i + 1$

RC 658) find k closest elements



target = 50

40 42 48 57

k = 4

closest element

- floor
- floor + 1 → ceil, target

```java
public int floorInArray(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 if(arr[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return right;
}
```

```java
public List<Integer> findClosestElements(int[] arr, int k, int target)
    int floor = floorInArray(arr, target);
    int ceil = floor + 1;

    for(int count = 0; count < k; count++){
        int floorDist = (floor == -1)
                ? Integer.MAX_VALUE : (target - arr[floor]);
        int ceilDist = (ceil == arr.length)
                ? Integer.MAX_VALUE : (arr[ceil] - target);

        if(floorDist <= ceilDist){
            floor--; // floor closer
        } else {
            ceil++; // ceil closer
        }
    }

    List<Integer> res = new ArrayList<>();
    for(int idx = floor + 1; idx < ceil; idx++){
        res.add(arr[idx]);
    }
    return res;
}
```

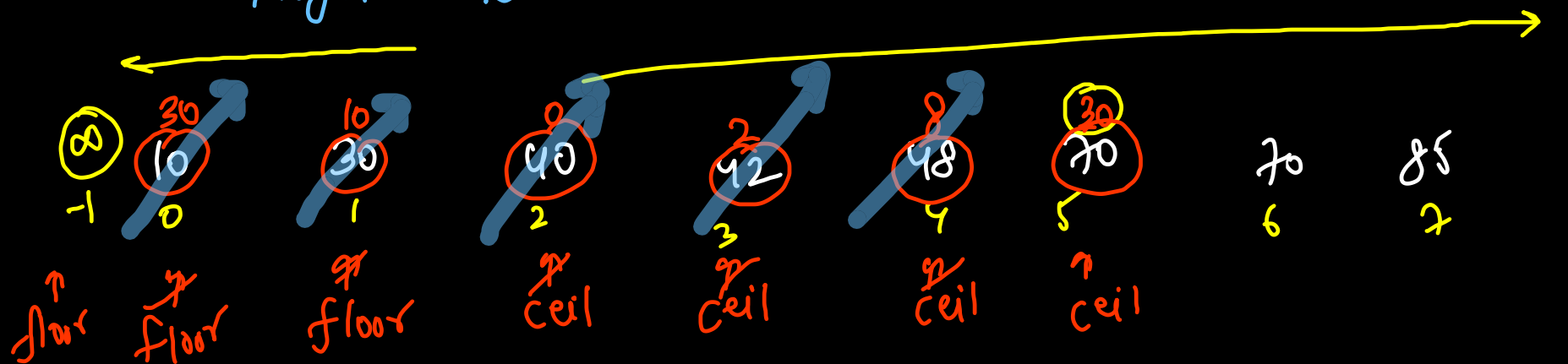
O(logN)

O(K)

output

O(K)

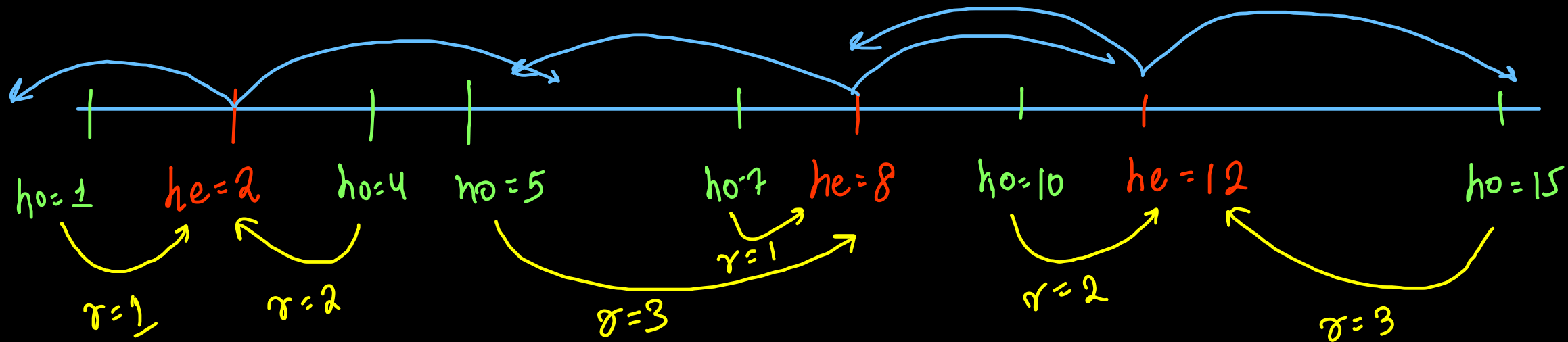
target = 40 , k = 8



time = O(logN + K)

Space = O(1)

LC 475) heaters



minradius = (3)

closest heater to every house

(sorted heaters) (unsorted houses)

heaters = {8, 2, 12} ⇒ {2, 8, 12}

houses = {7, 1, 4, 5, 15, 10}

logN

```java
public int floorInArray(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 if(arr[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return right;
}
```

time = O(n log n)

space = O(1)

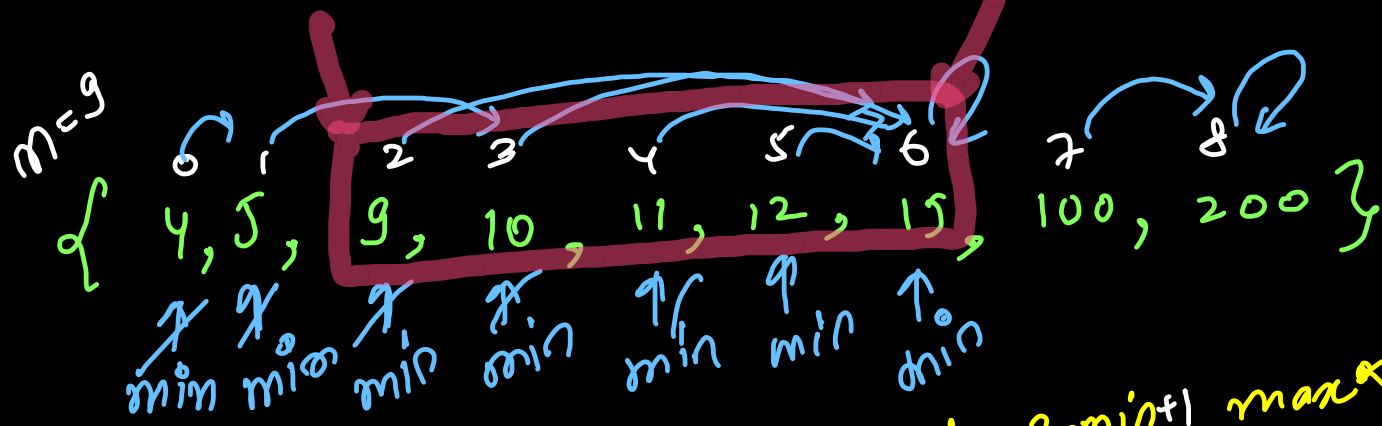
```java
public int findClosest(int[] heaters, int house){
    int floor = floorInArray(heaters, house);
    int ceil = floor + 1;

    int floorDist = (floor == -1)
        ? Integer.MAX_VALUE : (house - heaters[floor]);
    int ceilDist = (ceil == heaters.length)
        ? Integer.MAX_VALUE : (heaters[ceil] - house);

    return Math.min(floorDist, ceilDist);
}

public int findRadius(int[] houses, int[] heaters) {
    Arrays.sort(heaters); // closest heater queries → NlogN

    int minRadius = 0;
    for(int house: houses){ → O(N)
        int radius = findClosest(heaters, house); → O(logN)
        minRadius = Math.max(minRadius, radius);
    }

    return minRadius;
}
```

→ NlogN

Remove Min Elements

Binary Search + Two Pointer

{ 4, 5, 100, 9, 10, 11, 12, 15, 200 }



min removals

2 x min >= max

min 2xmin+1 max

4 → 8 → 9 → 5

5 → 10 → 11 → 10

9 → 18 → 19 → 15

10 → 20 → 21 → 15

11 → 22 → 23 → 15

2xmin → floor

[4, 5 / 0, 1] 2*4 >= 5

=> removals = 7

[5, 10 / 1, 3] 2*5 >= 10

=> removals = 6

[9 — 15 / 2, 6] 2*9 >= 15

=> removals = 4

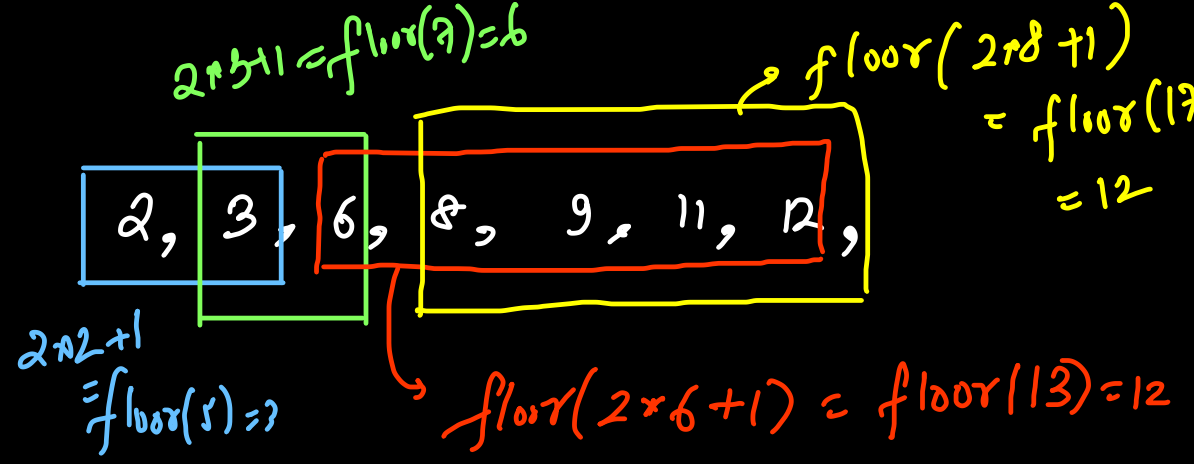
[10 - 15 / 3, 6] 2*10 >= 15

=> removals = 5

```java
int findFloor(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 if(arr[mid] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return right;
}

int minRemoval(int arr[], int n) {
    Arrays.sort(arr); ~ nlogn

    int minRemovals = n;
    for(int min = 0; min < n; min++){ → n
        int max = findFloor(arr, 2 * arr[min] + 1); → logn
        int removals = n - (max - min + 1);
        minRemovals = Math.min(removals, minRemovals);
    }

    return minRemovals;
}
```

Time ⇒ O(nlogn)

Space ⇒ O(1) constant

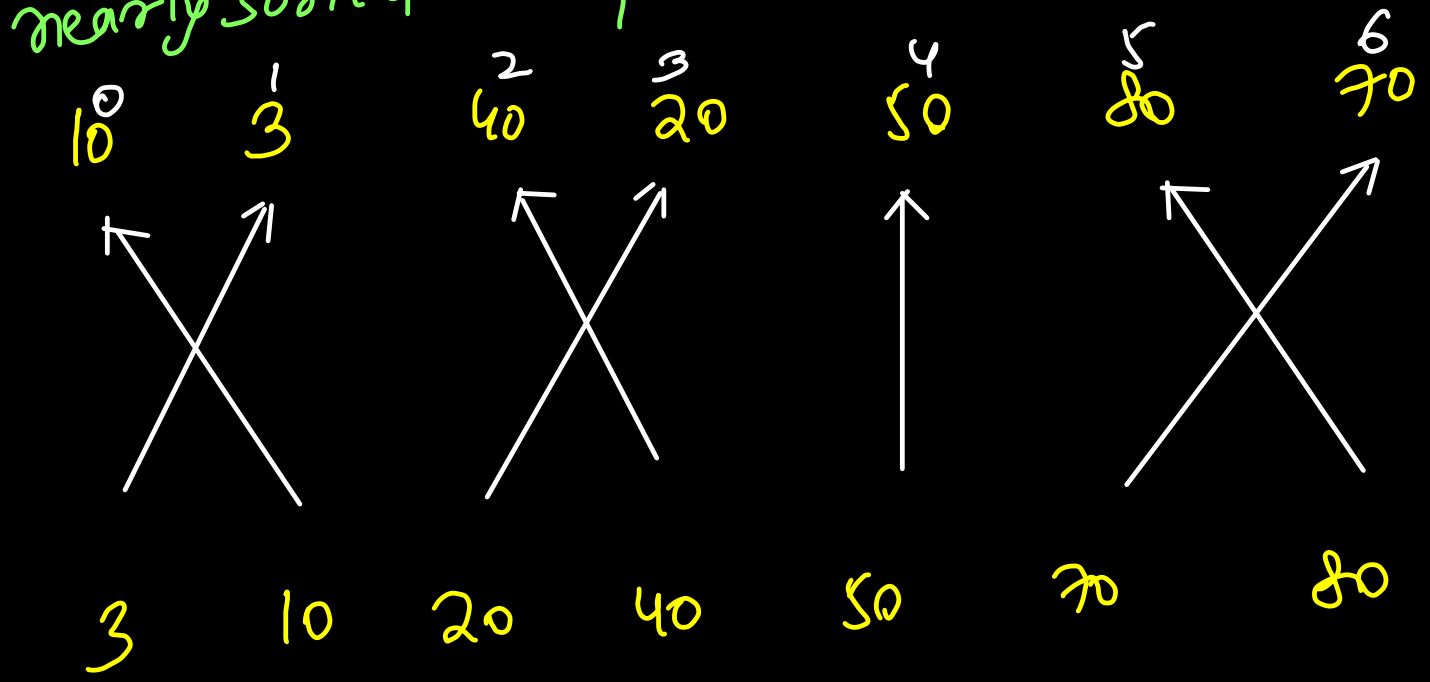


2 * min ≥ max

2 * 6 ≥ 12

Search in Nearly Sorted Array

nearly sorted array



sorted array

target = 40

brute force

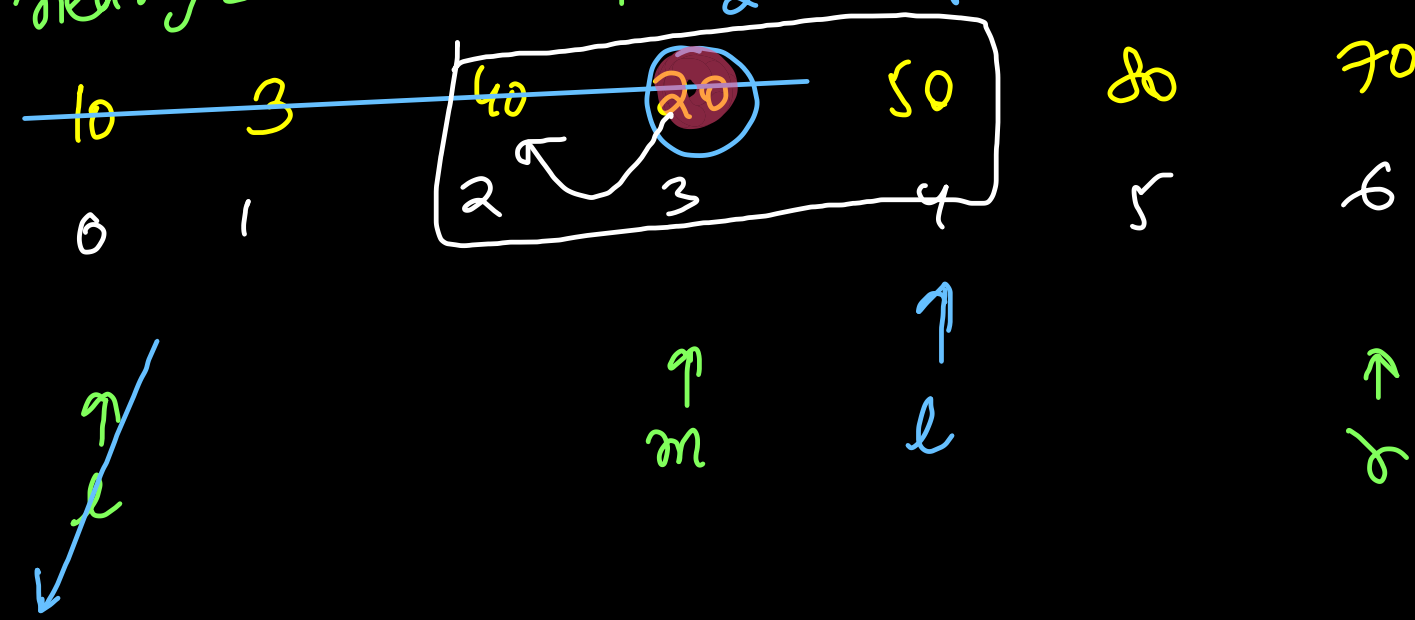
linear search

O(n)

binary search

O(log n)

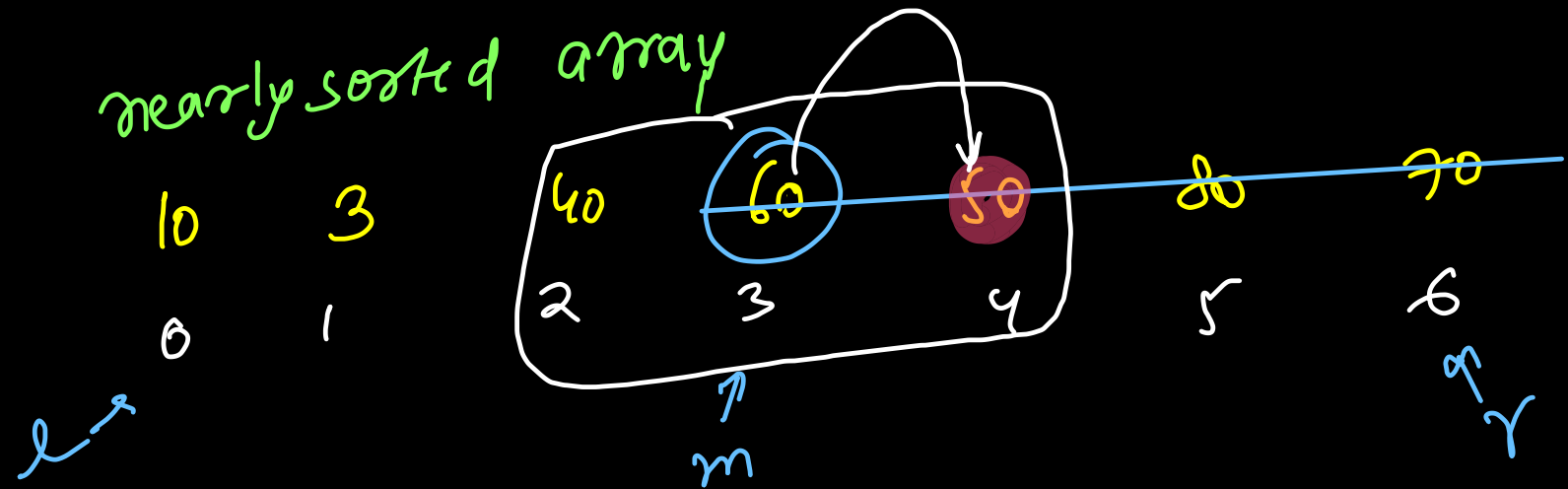
nearly sorted array



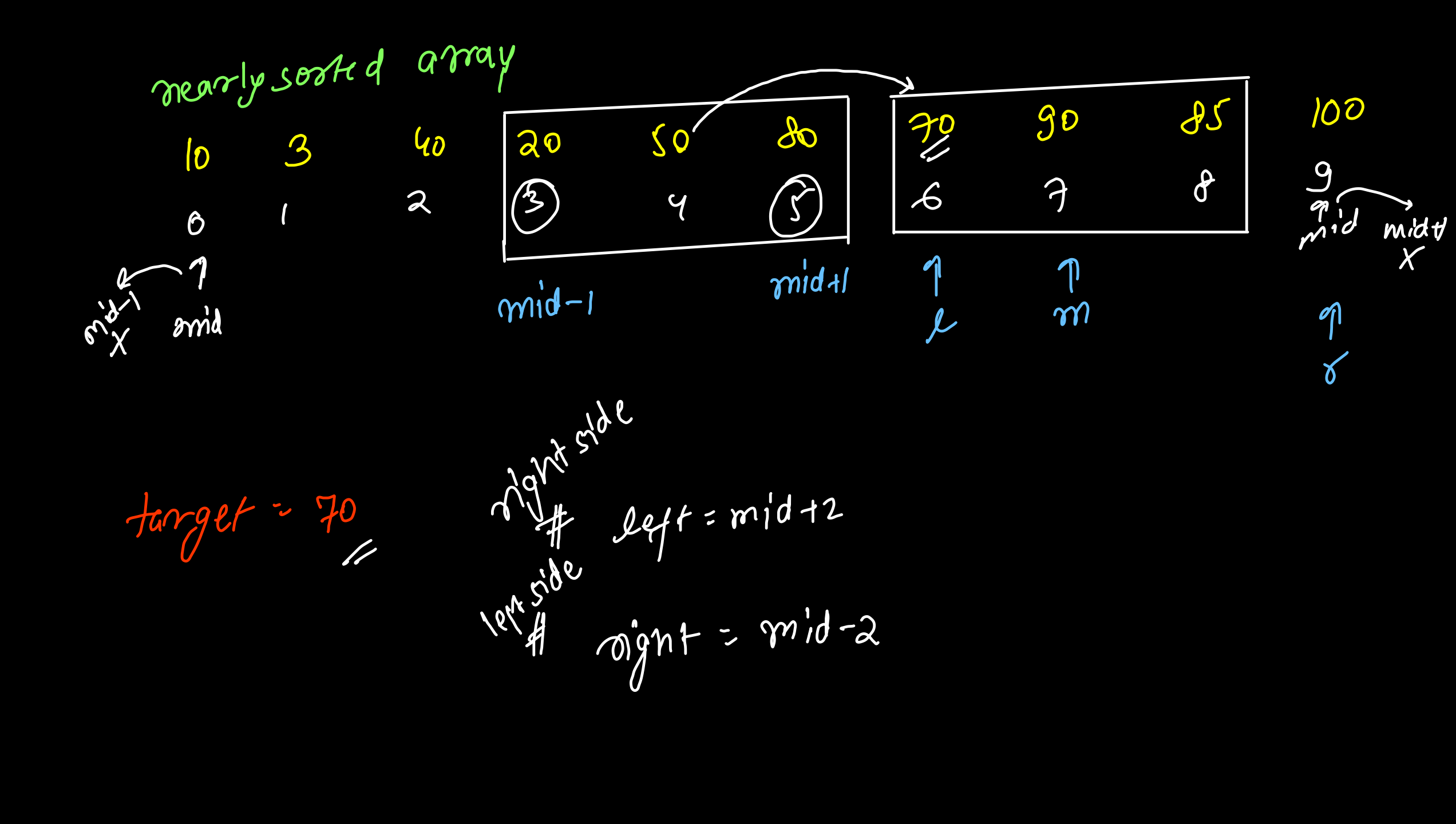
target = 40

Simple binary search will fail (arr[m] comparison)

nearly sorted array



target = 50


nearly sorted array
10 3 40
0 1 2
mid-1
mid
20 50 80
3 4 5
mid-1
mid+1
70 90 85
6 7 8
l
m
100
9
mid
mid+1
r
target = 70
right side
left = mid+2
left side
right = mid-2

```java
public static int binarySearch(int arr[], int l, int r, int x)
{
  if (r >= l)
  {
    int mid = l + (r - l) / 2;

    // If the element is present at
    // one of the middle 3 positions
    if (arr[mid] == x)
      return mid;
    if (mid > l && arr[mid - 1] == x)
      return (mid - 1);
    if (mid < r && arr[mid + 1] == x)
      return (mid + 1);

    // If element is smaller than mid, then
    // it can only be present in left subarray
    if (arr[mid] > x)
      return binarySearch(arr, l, mid - 2, x);

    // Else the element can only be present
    // in right subarray
    return binarySearch(arr, mid + 2, r, x);
  }

  // We reach here when element is
  // not present in array
  return -1;
}
```

(*) index out of bound

mid-1 / mid / mid+1

l=0, r = n-1;

while (l <= r) {

m = (l+r)/2;

(#)

if (arr[m] < target)
  l = m+2;
else
  r = m-2;
}

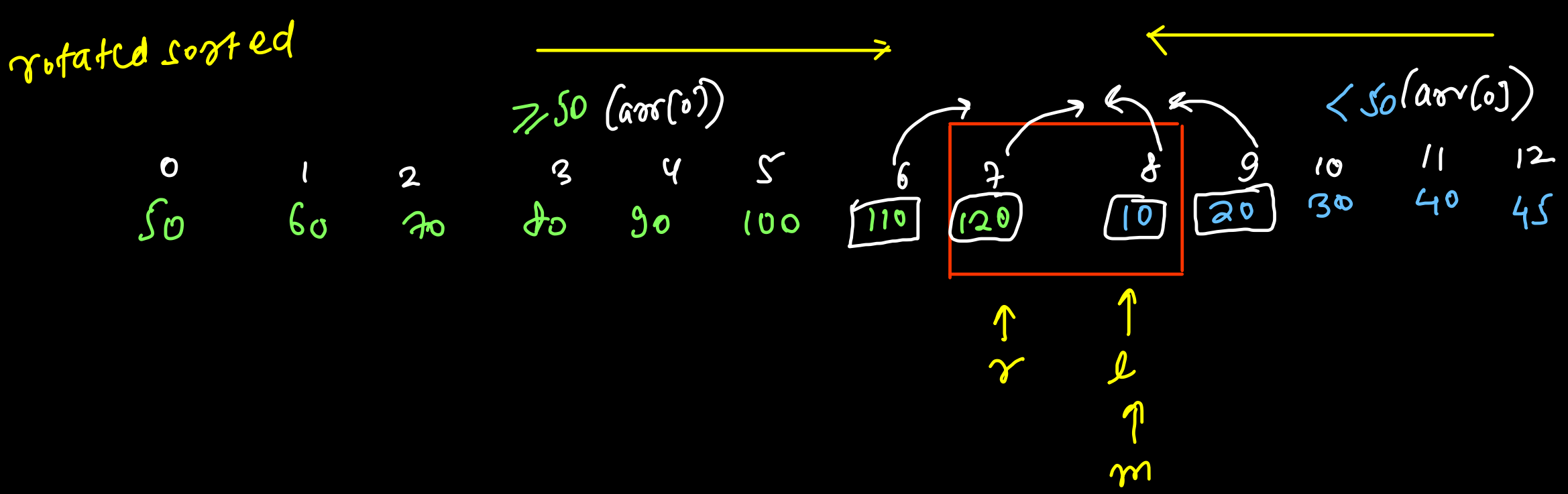
time = O(logN)

space = O(1) (iterative)

Minimum in Rotated Sorted Array

(Pivot → Minimum)

- Linear search ⟹ O(n) time, O(1) extra space
- Binary search (variation)



$110 \geq 50$ ($arr[m] \geq arr[0]$) $\Rightarrow l = m+1$

$20 < 50$ ($arr[m] < arr[0]$) $\Rightarrow r = m-1$

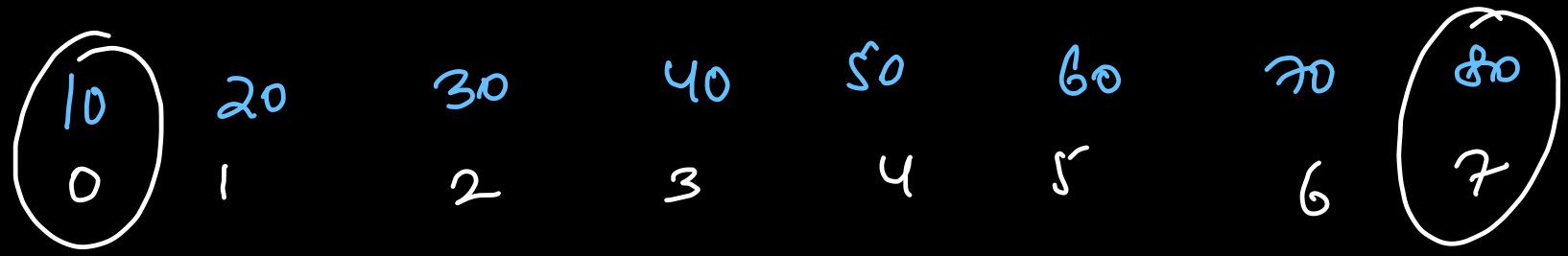
$120 \geq 50$ ($arr[m] \geq arr[0]$) $\Rightarrow l = m+1$

$10 < 50$ ($arr[m] < arr[0]$) $\Rightarrow r = m-1$

$max^m \rightarrow$ right

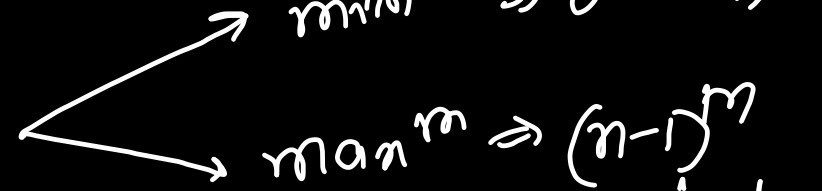
$min^m \rightarrow$ left

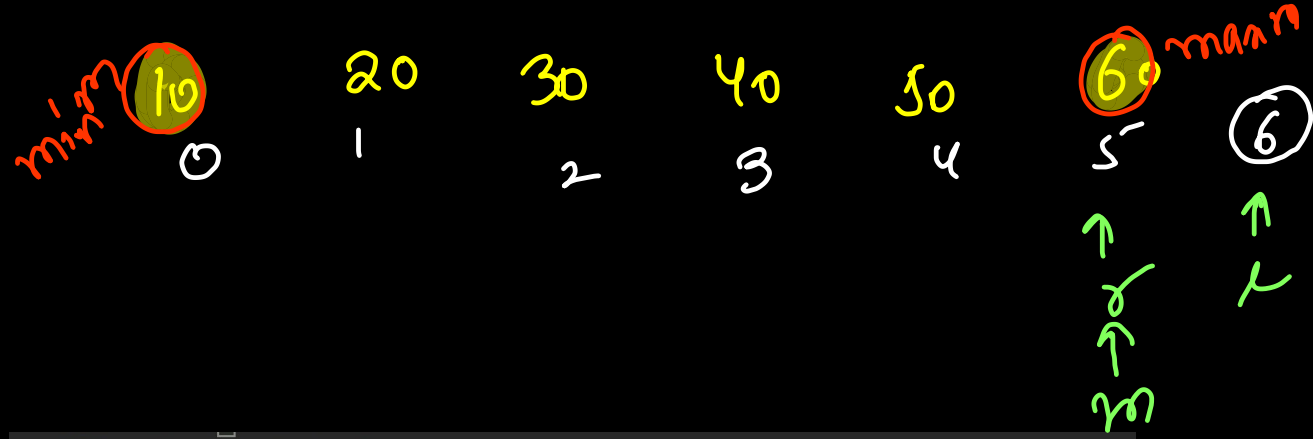
cornercase



completely sorted
(not rotated)

arr(0) ≤ arr[n-1]





Sorted + rotated = O/n time

```java
public int findMin(int[] nums) {
    int left = 0, right = nums.length - 1;
    if(nums[left] <= nums[right]){
        // corner case: completely sorted (not rotated at all)
        return nums[0];
    }

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

        if(nums[mid] >= nums[0]){
            // greater region (unrotated)
            left = mid + 1;
        } else {
            // lesser region (rotated)
            right = mid - 1;
        }
    }

    return nums[left];
}
```

LC 153

comparison with mid-1, mid+1 will fail

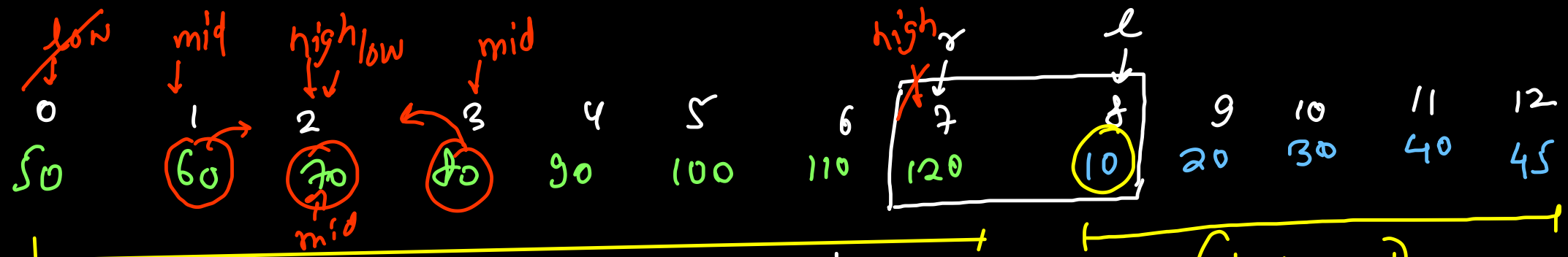
Variation of Binary Search

Time = $O(\log_2 n)$

Space = O(1) constant

LC (33) Search in Rotated Sorted Array

Linear Search :→ O(n) time, O(1) space , Binary Search :→ O(log₂n) time, O(1) space



target = 70

① Find transition pt ⇒ O(log₂n)

② Check target ← rotated / unrotated ⇒ O(1)

③ Binary search ⇒ O(log₂n)

```java
public int findMin(int[] nums) {
    int left = 0, right = nums.length - 1;
    if(nums[left] <= nums[right]){
        // corner case: completely sorted (not rotated at all)
        return 0;
    }

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

        if(nums[mid] >= nums[0]){
            // greater region (unrotated)
            left = mid + 1;
        } else {
            // lesser region (rotated)
            right = mid - 1;
        }
    }

    return left;
}
```

```java
public int search(int[] nums, int target) {
    int pivot = findMin(nums);

    if(pivot == 0){ // not rotated
        return binarySearch(nums, target, 0, nums.length - 1);
    } else if(target < nums[0]){
        return binarySearch(nums, target, pivot, nums.length - 1);
    } else {
        return binarySearch(nums, target, 0, pivot - 1);
    }
}
```

```java
public int binarySearch(int[] nums, int target, int left, int right) {
    while(left <= right){
        int mid = (left + right) / 2;
        if(nums[mid] == target){
            return mid;
        } else if(nums[mid] < target){
            left = mid + 1;
            // discard left to mid region
        } else {
            right = mid - 1;
            // discard mid to right region
        }
    }

    return -1; // search unsuccessful
}
```

↳ this fn will only work if array is sorted (not rotated)

R133

eg1

0	1	2	3	4	5
40	50	60	10	20	30

pivot

target = 50 ≥ 40 ⇒ [0,2]

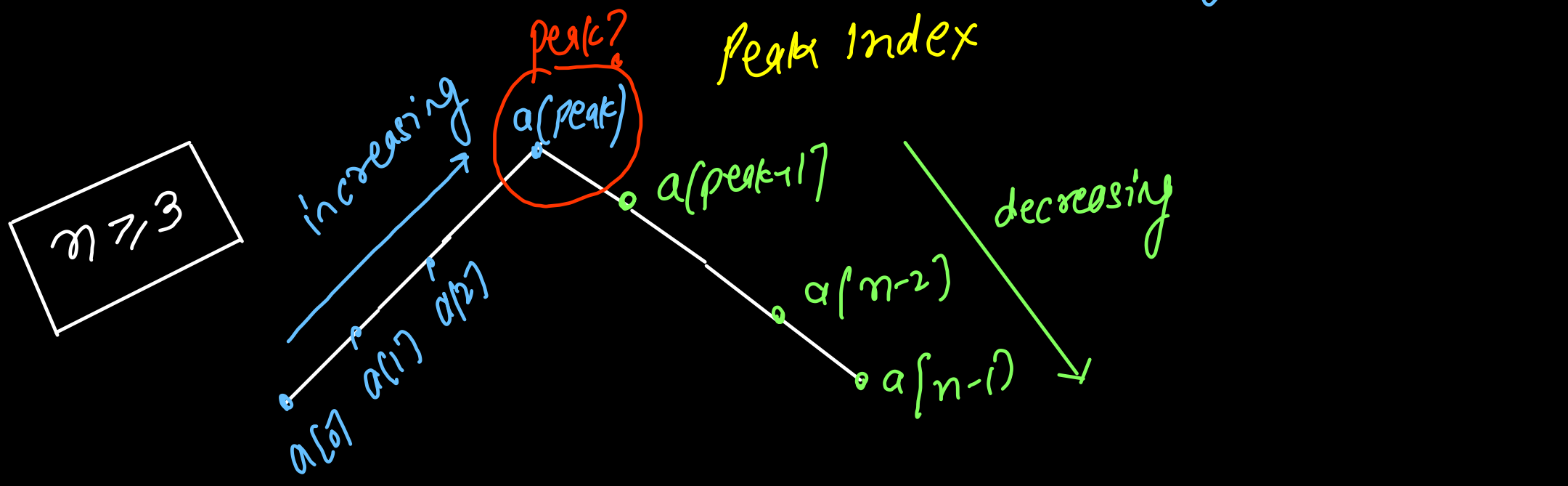
target: 25 < 40 ⇒ [3,5]

eg2

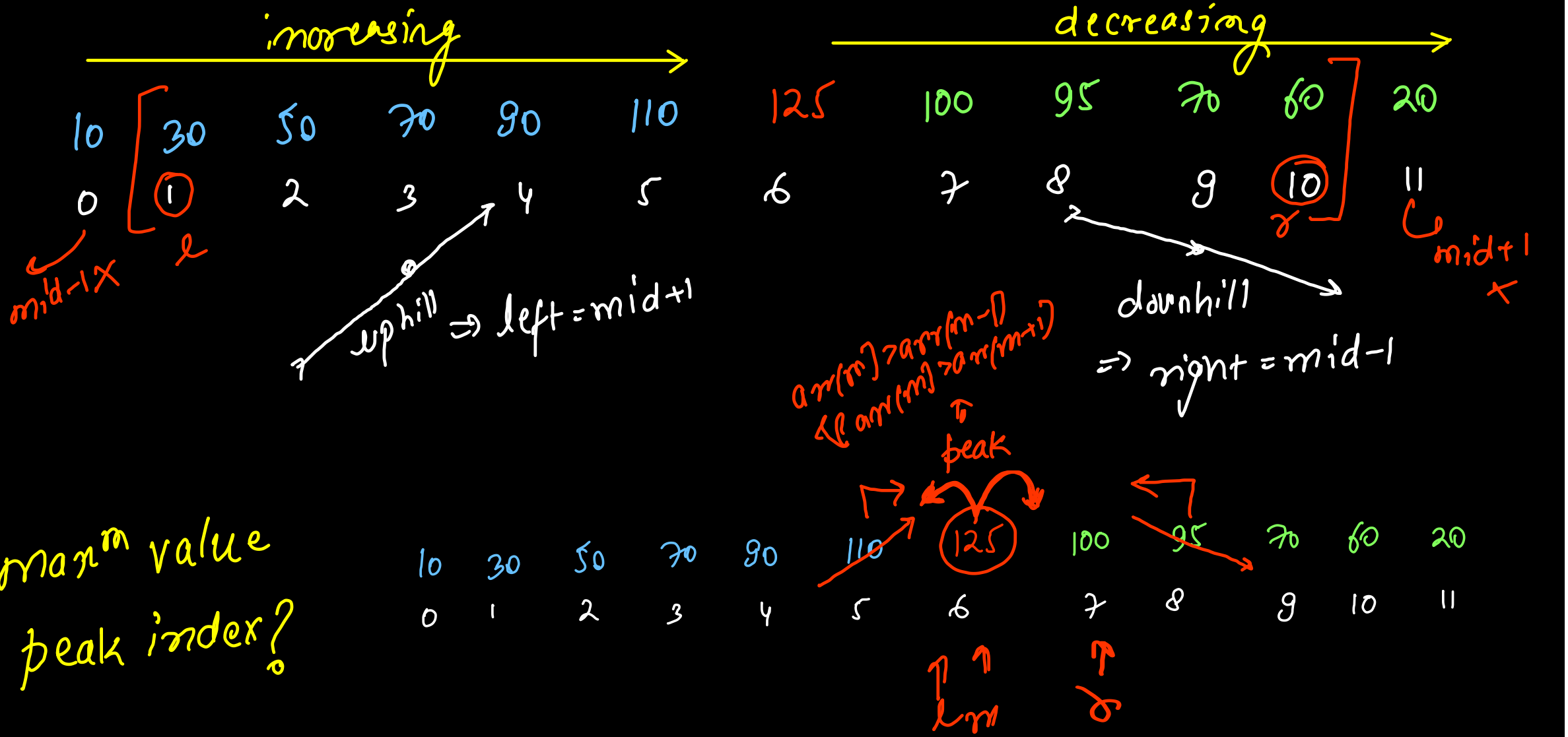
10	20	30	40
0	1	2	3

pivot → search

LC 852) Mountain Array / Bitonic Array



10 30 50 70 90 110 125 100 95 70 60 20


increasing
10 30 50 70 90 110
0 1 2 3 4 5
mid-1 X
l
uphill ⇒ left = mid+1
decreasing
125 100 95 70 60 20
6 7 8 9 10 11
r
mid+1
X
downhill
⇒ right = mid-1
arr[m] > arr[m-1]
&& arr[m] > arr[m+1]
peak
max^m value
peak index?
10 30 50 70 90 110 125 100 95 70 60 20
0 1 2 3 4 5 6 7 8 9 10 11
l_m
r_m

```java
public int peakIndexInMountainArray(int[] arr) {
    int left = 1, right = arr.length - 2;

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

        if(arr[mid] > arr[mid - 1] && arr[mid] > arr[mid + 1]){
            // peak (greater than both neighbours)
            return mid;
        } else if(arr[mid] > arr[mid - 1]){
            // uphill
            left = mid + 1;
        } else {
            // downhill
            right = mid - 1;
        }
    }

    return -1;
}
```

Peak cannot be 0 or last index (array is valid mountain)

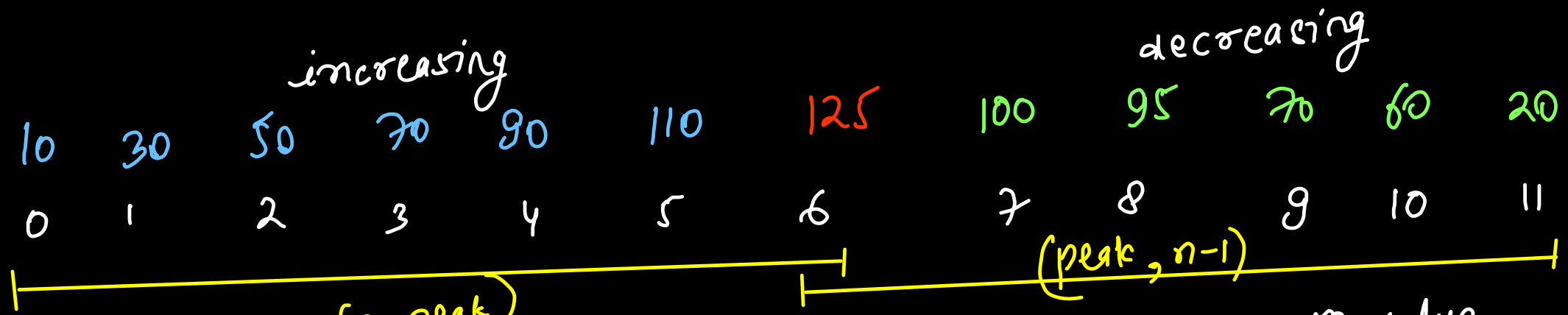
this line will never execute (if peak exists)

Time = $O(\log_2 n)$

Space = $O(1)$

LC 852

(Interviewbit) Search in Bitonic / Mountain Array



target = 90 ⇒ (4)

target = 60 ⇒ (10)

target = 45 ⇒ (-1)

① Peak index → Maam value
↳ $O(\log_2 n)$

+

② find target in uphill → $O(\log_2 n)$

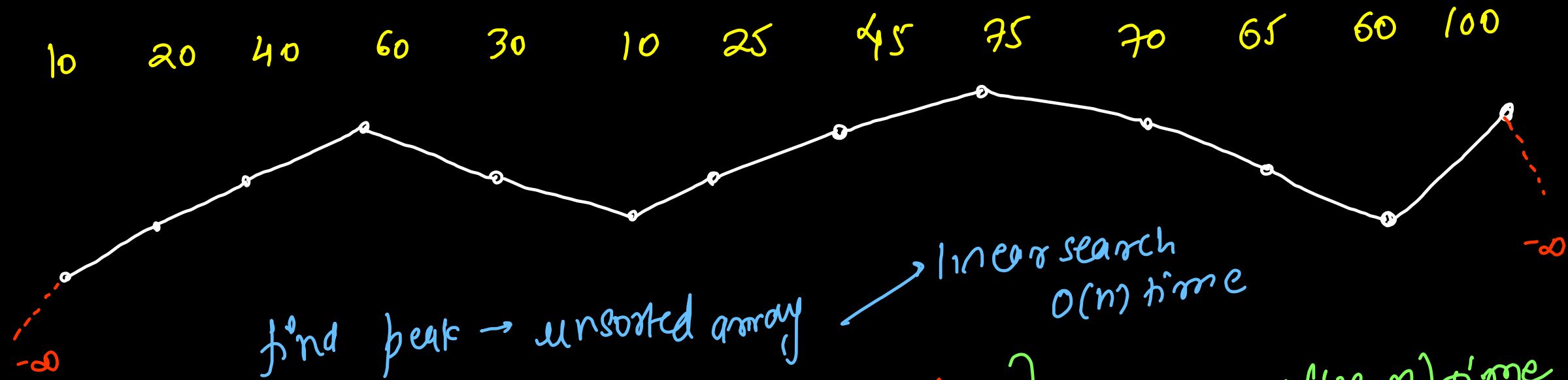
or/and

+

③ find target in downhill
↳ $O(\log_2 n)$

Lc 162) Find Peak Element

multiple peaks



find peak → unsorted array → linear search O(n) time

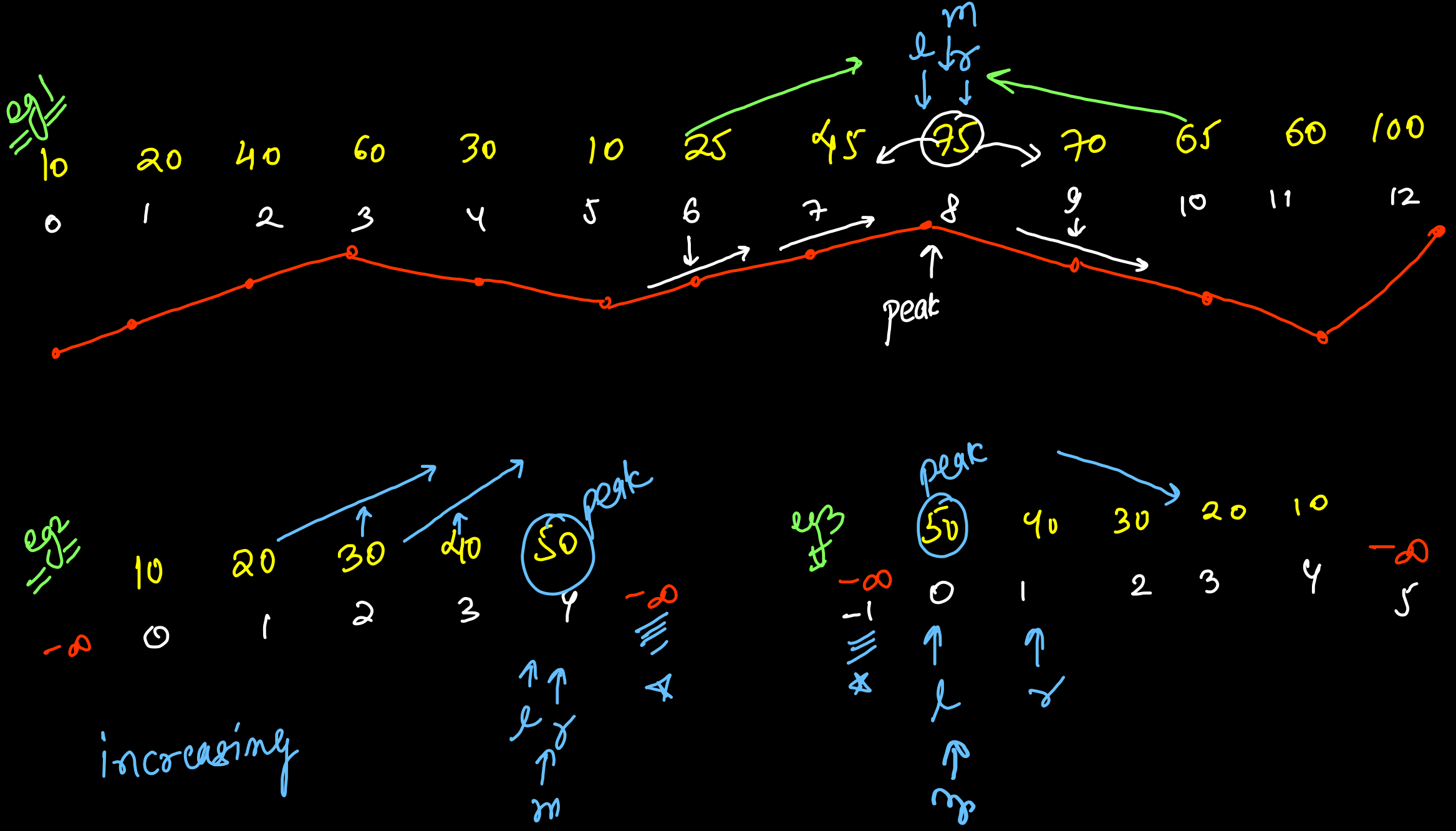
+

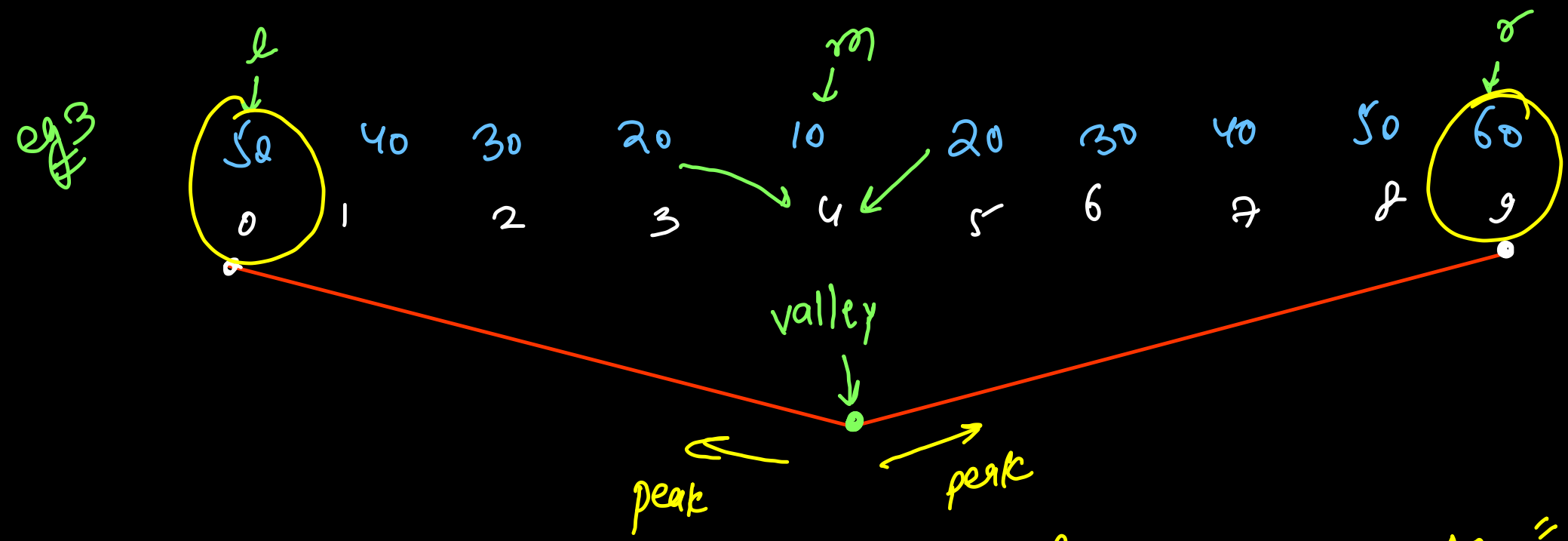
peak (any) < global ✗ / local ✓

+

guarantee ⇒ peak always exist ⇐ out of bound elements (-∞)

} binary search ⇒ $O(\log_2 n)$ time ⇒ O(1) space


eg!
10 20 40 60 30 10 25 45 75 70 65 60 100
0 1 2 3 4 5 6 7 8 9 10 11 12
l
m
peak
eg
10 20 30 40 50 -∞
-∞ 0 1 2 3 4 -∞
peak
increasing
l
r
m
eg
-∞ 50 40 30 20 10 -∞
-1 0 1 2 3 4 5
peak
l
r
m



"peak will be in both directions from a valley ie"

```java
public int findPeakElement(int[] nums) {
    int n = nums.length;
    if(n == 1) return 0;
    if(nums[0] > nums[1]) return 0;
    if(nums[n - 1] > nums[n - 2]) return n - 1;

    int left = 1, right = n - 2;
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(nums[mid] > nums[mid - 1] && nums[mid] > nums[mid + 1]){
            // peak (greater than both neighbours)
            return mid;
        } else if(nums[mid] > nums[mid - 1]){
            // uphill
            left = mid + 1;
        } else {
            // downhill
            right = mid - 1;
        }
    }

    return -1;
}
```

only one element

0th index is peak

last index is peak

Corner cases

LC 162

find peak in bitonic array

LC 69) Square Root of Integer

(Floor → Integer Ans)

floor

perfect square → $\sqrt{3} = 1$

$\sqrt{4} = 2$

→ $\sqrt{5} = 2$

$\sqrt{9} = 3$

→ $\sqrt{10} = 3$

$\sqrt{16} = 4$

→ $\sqrt{20} = 4$

Java Inbuilt Fn

⇒ Math.sqrt(x)

or

Math.pow(x, 0.5)

Approach (1)

$\sqrt{16}$ ⇒

$1 \times 1 \leq 16$ ✓

$2 \times 2 \leq 16$ ✓

$3 \times 3 \leq 16$ ✓

$4 \times 4 \leq 16$ ✓

$5 \times 5 \not\leq 16$ ×

(linear search)

Time ⇒ $O(\sqrt{n})$

Square root

approach ① linearsearch

```java
public int mySqrt(int x) {
    long answer = 0;

    for(long idx = 1; idx <= x; idx++){
        if(idx * idx <= x){
            answer = idx;
        } else break;
    }

    return (int)answer;
}
```

$n = 20$

$1^2 \leq 20$ ✓

$2^2 \leq 20$ ✓

$3^2 \leq 20$ ✓

$4^2 \leq 20$ ✓ ⇒ answer = 4

$5^2 \not\leq 20$ X

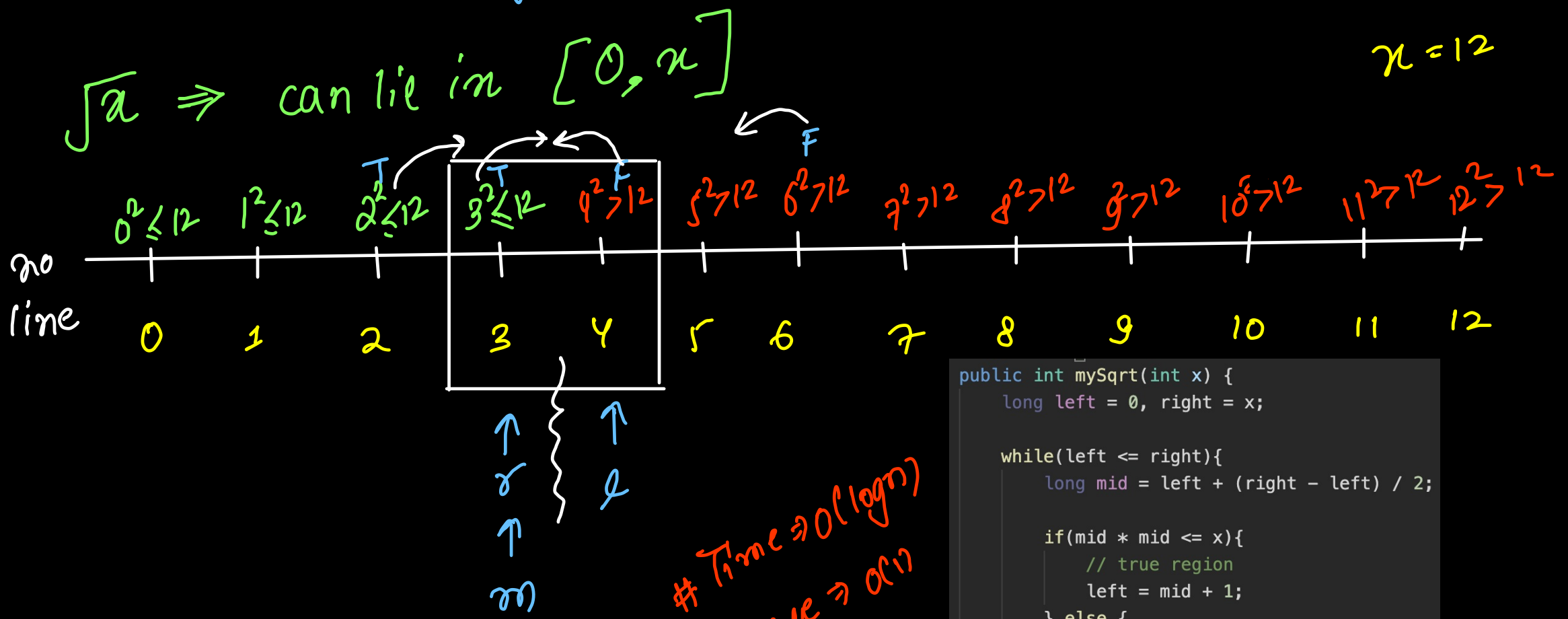
$O(\sqrt{n})$ time

$O(1)$ space

approach ② binary search / Divide & Conquer

$\sqrt{x}$ ⇒ can lie in $[0, x]$

$x = 12$



Time ⇒ O(log n)

Space ⇒ O(1)

```java
public int mySqrt(int x) {
    long left = 0, right = x;

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

        if(mid * mid <= x){
            // true region
            left = mid + 1;
        } else {
            // false region
            right = mid - 1;
        }
    }

    return (int)right;
}
```

LC 367) valid perfect square

1, 4, 9, 16, 25 ⟶ true

anything else ⟶ false

no inbuilt library.

Approach ① O(√n)

Count factors

odd ✓ even ✗

Approach ② O(log n)

binary search ⟶ sqrt

(9) ⇒ 3 * 3 = 9 ✓

(12) ⇒ 3 * 3 ≠ 12 ✗

```java
public int mySqrt(int x) {
    long left = 0, right = x;

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

        if(mid * mid <= x){
            // true region
            left = mid + 1;
        } else {
            // false region
            right = mid - 1;
        }
    }

    return (int)right;
}

public boolean isPerfectSquare(int num) {
    int sqrt = mySqrt(num);
    return sqrt * sqrt == num;
}
```

Searching in 2D Matrix

unsorted ?

(1) Brute force :→ Linear search → O(m × n) quadratic time
                                 → O(1) space constant

↳ ordering of matrix ✗

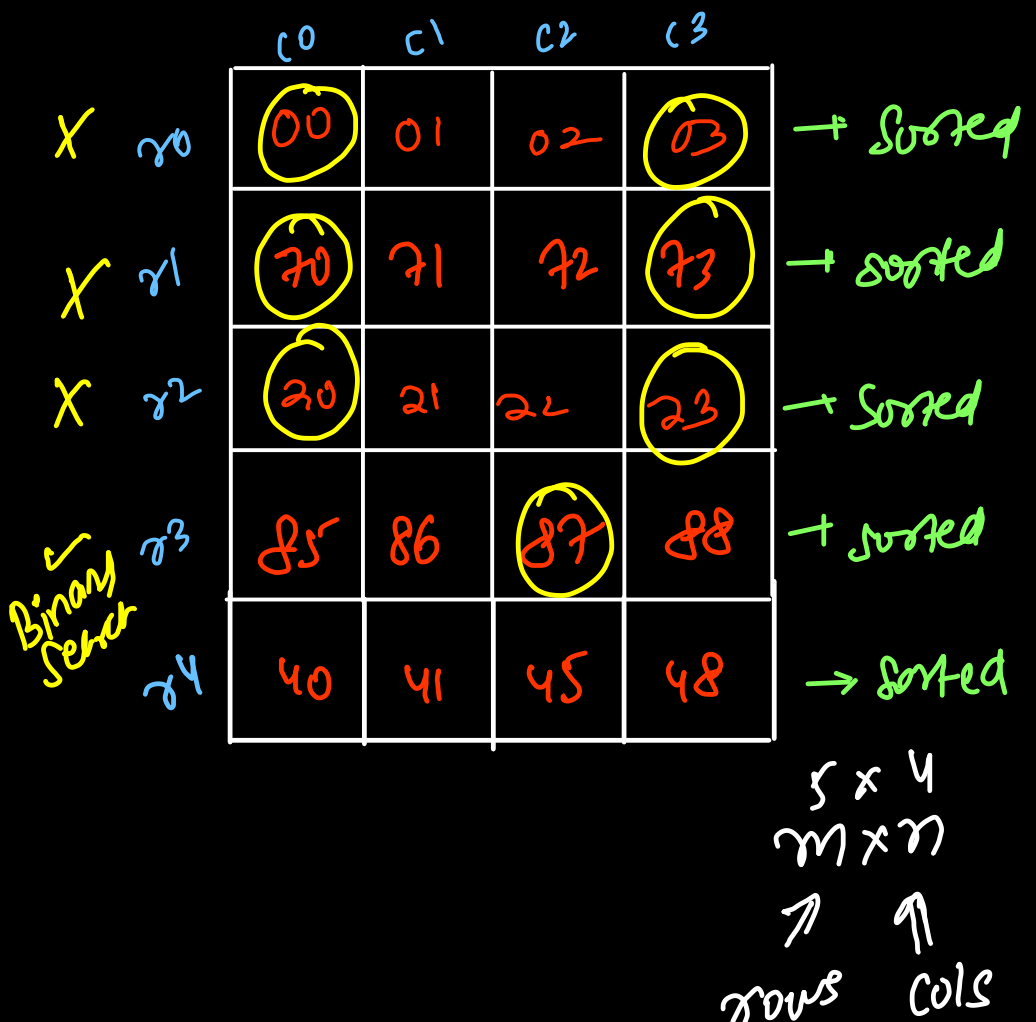
(2) All rows are sorted individually

(3) All cols are sorted individually

(4) All rows & cols are sorted individually

(5) All rows & cols are sorted continuously

All rows individually sorted



Target = 87 ⇒ search successful

43 ⇒ search unsuccessful

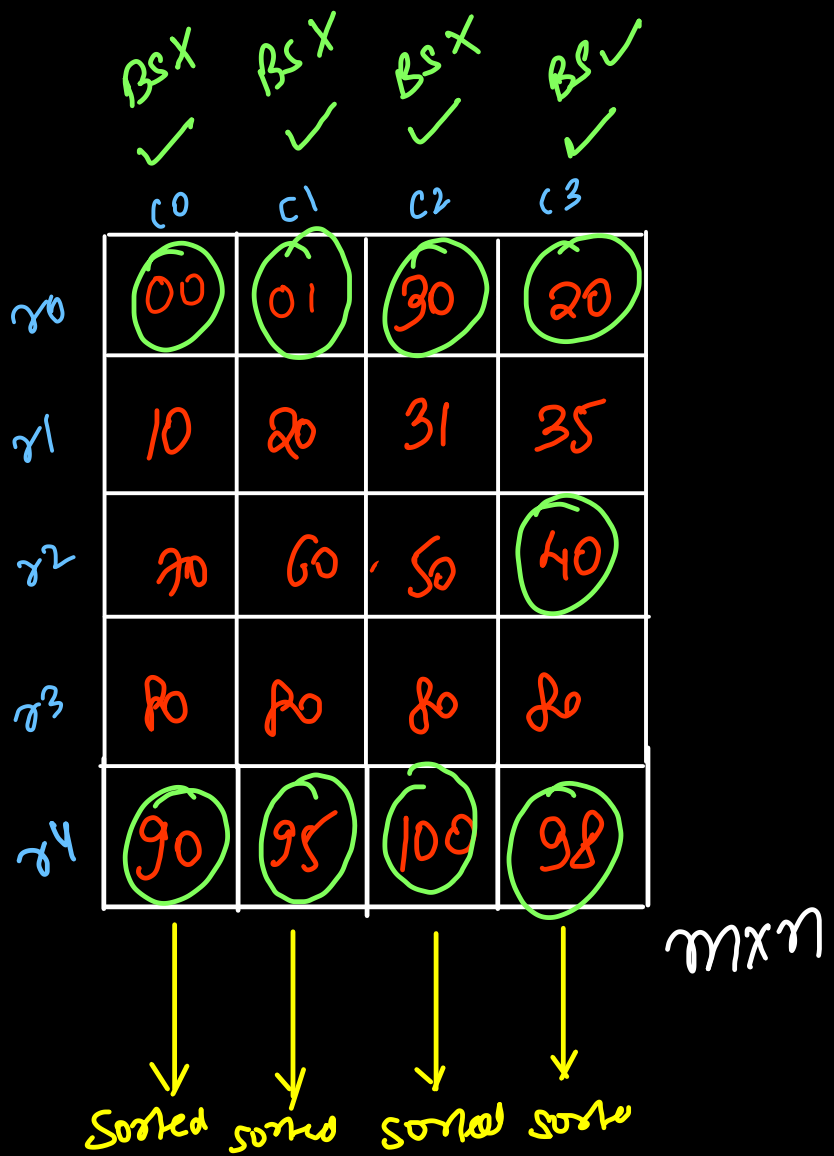
Worst case

BS on each row

$O(\log n) * m$

$\Rightarrow O(m \log n)$

All cols individually sorted



target = 40

BS on each col

⇓

O(log m) * n

O(n log m)

R 240

All rows & cols are sorted individually

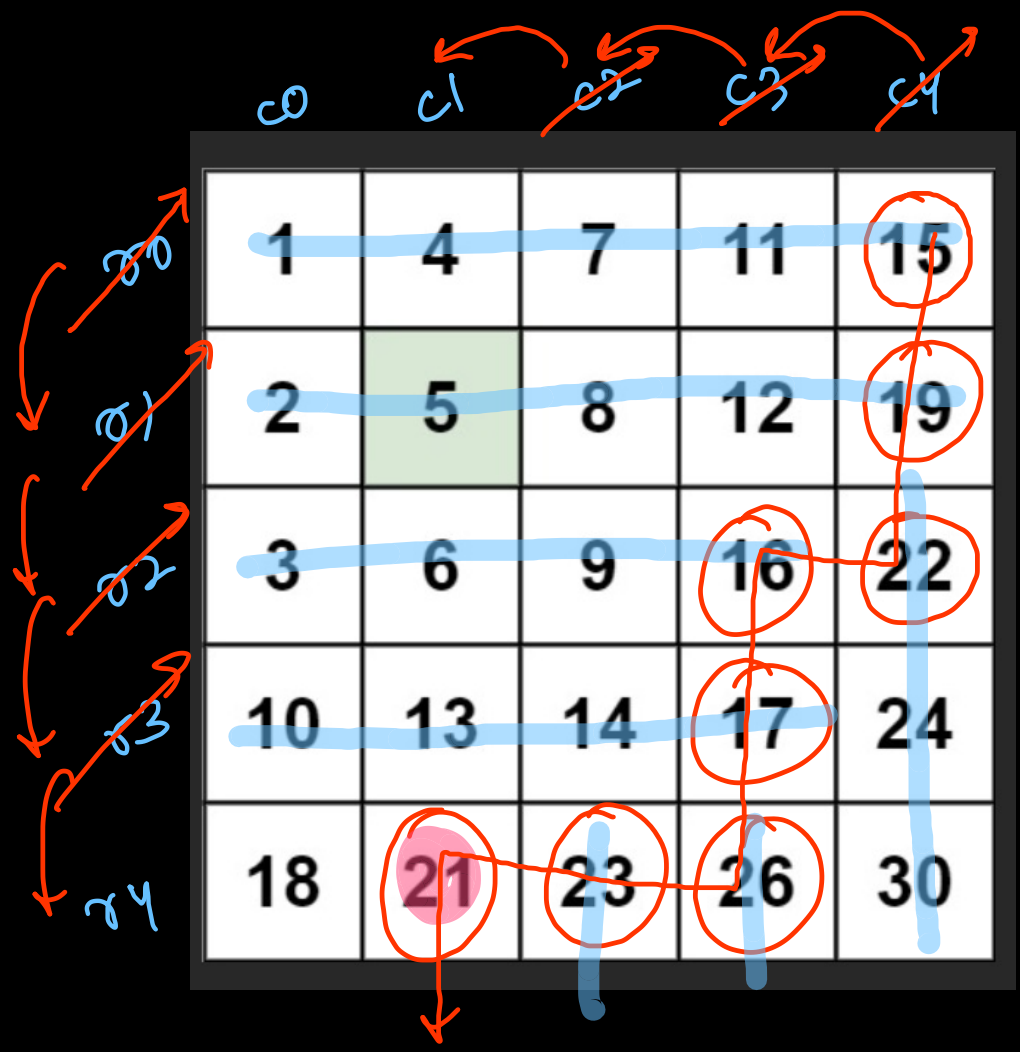
target = 21

	c0	c1	c2	c3	c4
r0	1	4	7	11	15
r1	2	5	8	12	19
r2	3	6	9	16	22
r3	10	13	14	17	24
r4	18	21	23	26	30

App1 Row by row Binary search
O(m log n)

App2 Col by col Binary search
O(n log m)

App3 Staircase Search
(Divide & Conquer)



discard → row / col

search space

target → 21

row = 0, col = n-1 = 4

15 < 21

↓

row = 1, col = 4

19 < 21

↓

row = 2, col = 4

22 > 21

↓

row = 2, col = 3

16 < 21

row = 3, col = 3

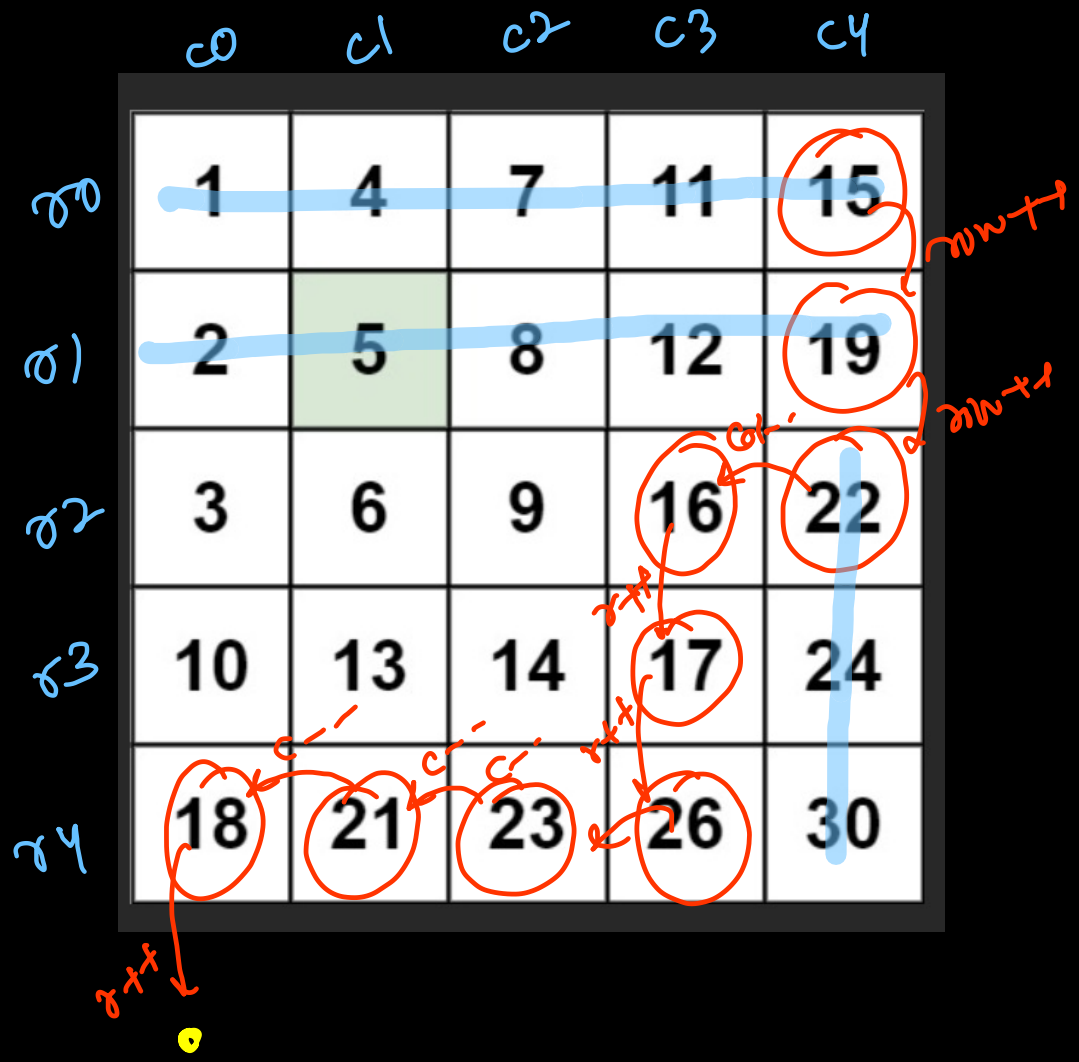
17 < 21

↓

row = 4, col = 3

26 > 21

(search unsuccessful)
target = 20



top right

row++ ↓ bottom wall ↓ row == m

col-- ↓ left wall ↓ col == -1

Leetcode ★

```java
public boolean searchMatrix(int[][] mat, int target) {
    int row = 0, col = mat[0].length - 1;

    // staircase or stepcase search
    while(row < mat.length && col >= 0){
        if(mat[row][col] == target){
            return true; // search successful
        } else if(mat[row][col] < target){
            row++; // discard row
        }
        else {
            col--; // discard col
        }
    }

    return false; // search unsuccessful
}
```

O(m) + O(n)

Time ⇒ O(m+n) linear

Space ⇒ O(1) space

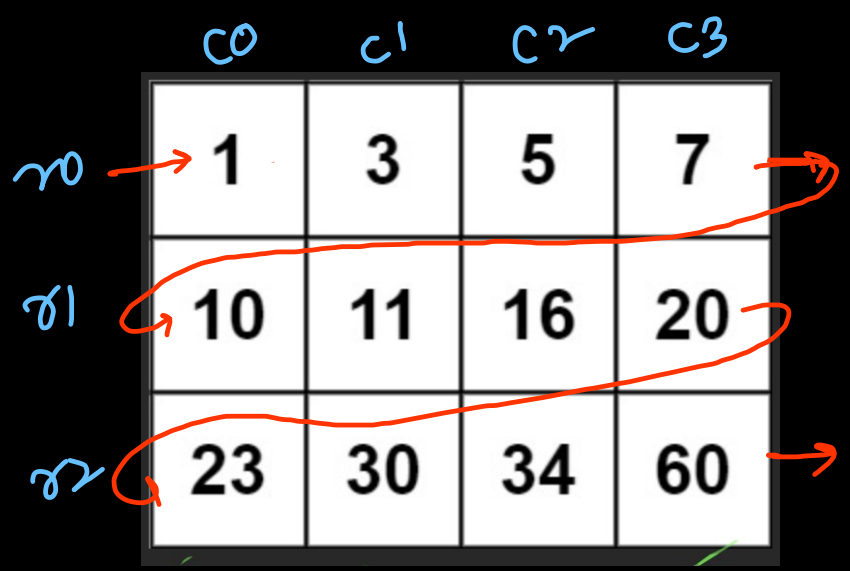
	c0	c1	c2	c3	c4
r0	1	4	7	11	15
r1	2	5	8	12	19
r2	3	6	9	16	22
r3	10	13	14	17	24
r4	18	21	23	26	30

target = 18

"faang" ★

RC34

Rows & cols (All 1s) are Sorted Continuously *



target = 34

App ① BS on each row → O(nlogm)

App ② BS on each col → O(mlogn)

App ③ Skpeare Search → O(n+m)

App ④

3x4
mxn

target: 16

	c0	c1	c2	c3
r0	1 (0)	3 (1)	5 (2)	7 (3)
r1	10 (4)	11 (5)	16 (6)	20 (7)
r2	23 (8)	30 (9)	34 (10)	60 (11)

Completely Sorted

BS ⟹ O(log(mxn))

left=0, mid=5, right = 11 (mxn-1)

↓

row = 5/4, col = 5%4 ⇒ (1,1)

11 < 16

left=6, mid=8, right=11

↓

row = 8/4, col = 8%4 ⇒ [2,0]

23 > 16

left=6, mid=6, right=7

↓

row = 6/4 = 1, col = 6%4 [1,2]

16 = 16

Convert 1D to 2D

idx → row → idx / cols

idx → col → idx % cols

```java
public boolean searchMatrix(int[][] mat, int target) {
    int rows = mat.length, cols = mat[0].length;
    int left = 0, right = rows * cols - 1;

    while(left <= right){
        int mid = left + (right - left) / 2;
        int row = mid / cols, col = mid % cols;
        if(mat[row][col] == target){
            return true;
        } else if(mat[row][col] < target){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return false;
}
```

Convert 1d to 2d

LC 74

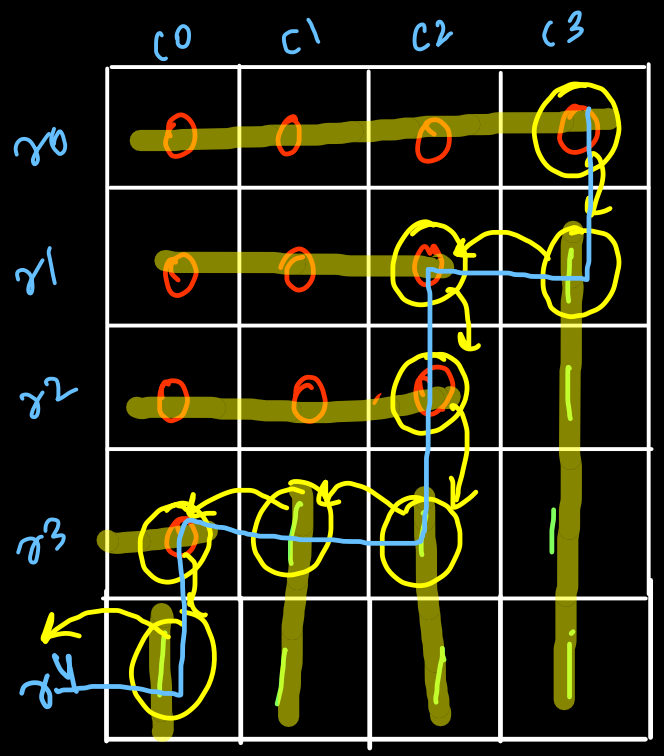
BS on 2D array

(assuming as 1d array)

Time ⇒ O(log(mxn))

Space ⇒ O(1)

GFG Count Zeros in sorted matrix



{ All rows are sorted (0s left, 1s right)
  { All cols are sorted (0s top, 1s bottom)

- Brute force :→ linear search ⇒ $O(N^2)$
- Optimized :→ BS on each row → $O(N \log N)$
- Most optimal :→ Staircase search
  $O(2n)$ linear

row, col
- 0 → row++ ; count += (col+1);
- 1 → col--

Count = 0+4+3+3+1

Imp

```
int countZeros(int A[][], int N)
{
    int zeros = 0;
    int row = 0, col = N - 1;

    while(col >= 0 && row < N){
        if(A[row][col] == 0){
            row++;
            zeros += (col + 1);
        } else {
            col--;
        }
    }

    return zeros;
}
```

ones = 0

leftwall

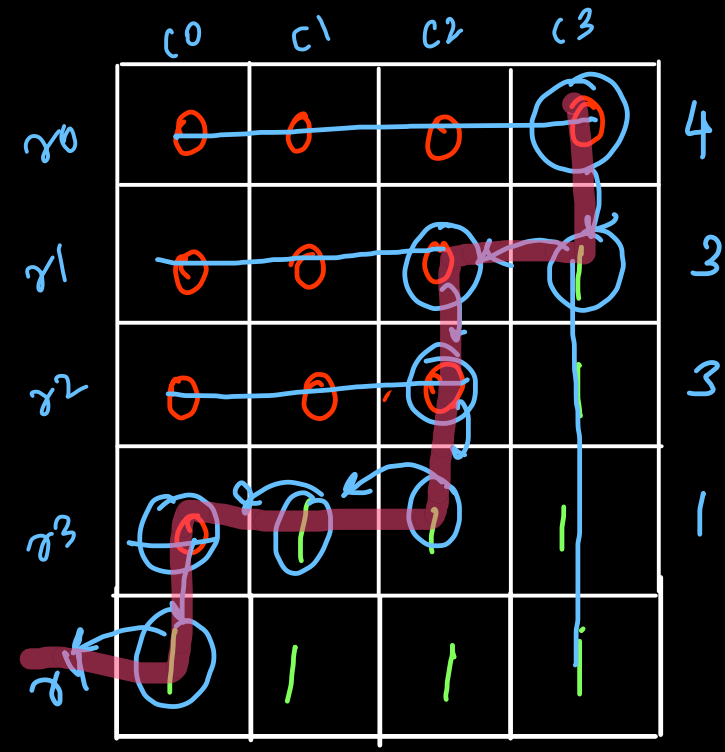
bottomwall

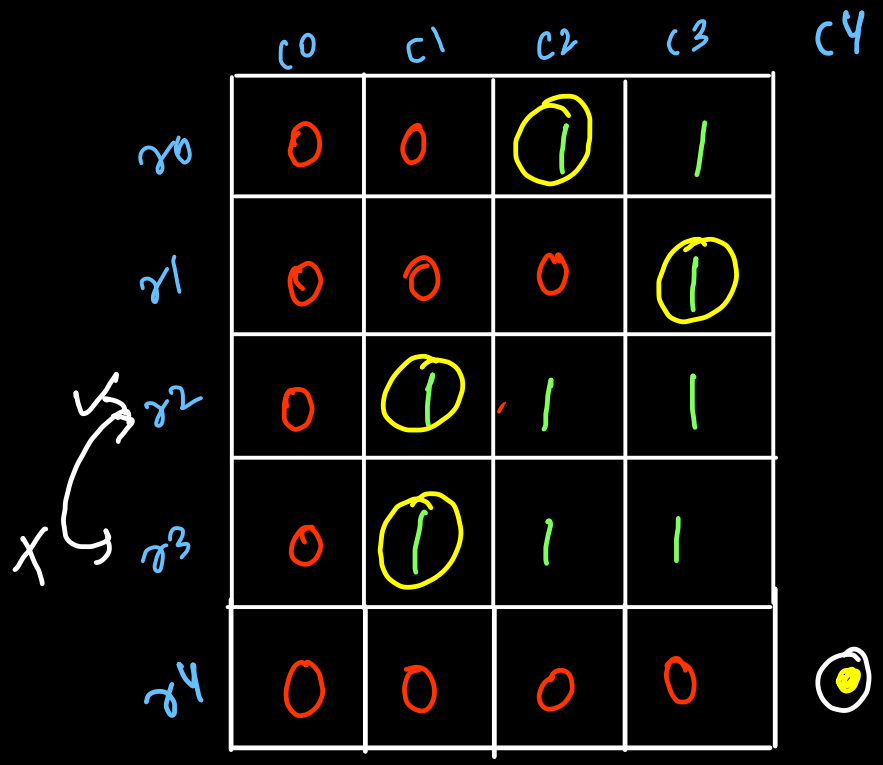
O(n) + O(n)

ones += (N - row);

Time ⇒ O(2n) ⇒ linear

Space ⇒ O(1)





given row ⇒ no of 1s

⇒ transition pt = tp

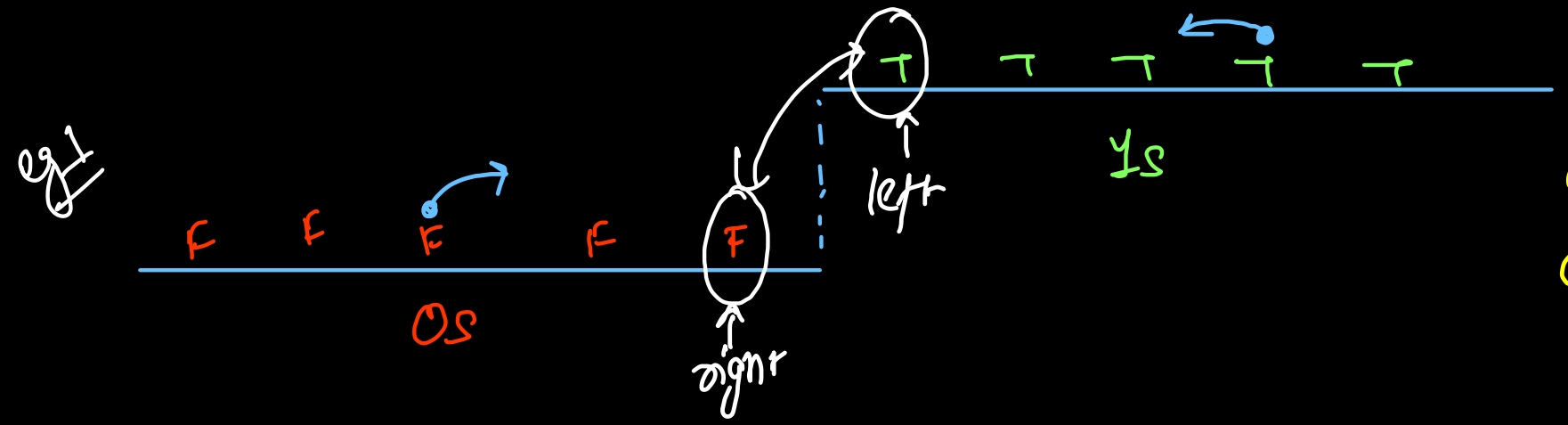
⊛ Homework ques

answer Row = ~~0~~ ~~2nd~~ 2nd

tp = ~~2nd~~ 1st

Transition pt → Sorted

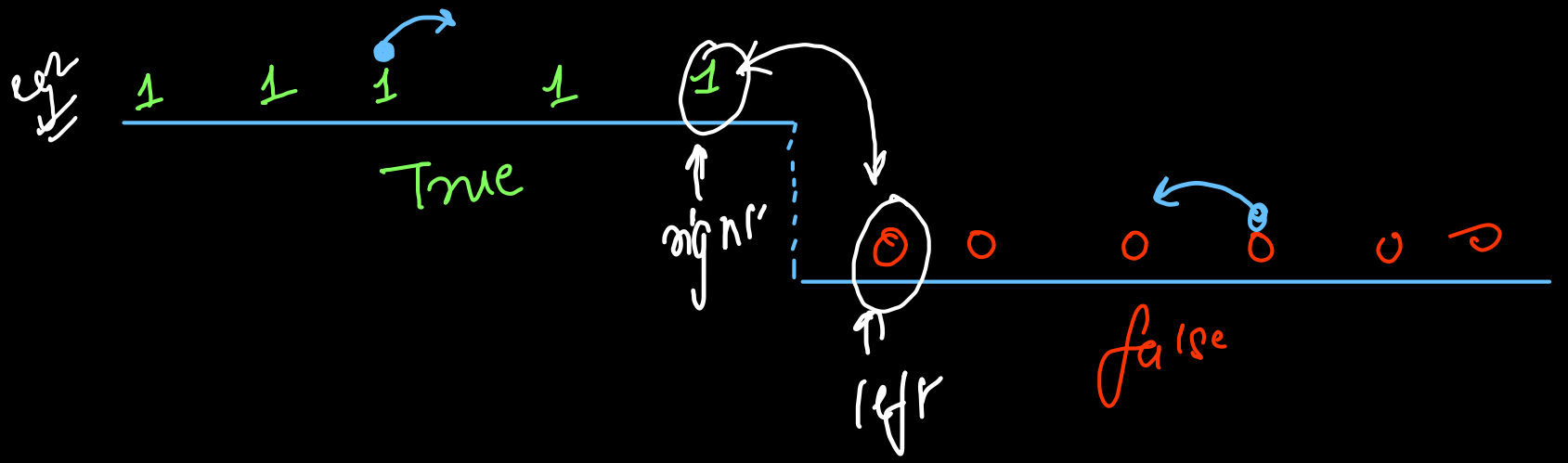
(Binary Array / Boolean Array)



"minimizing the max"

arr[m] == T right = mid-1

arr[m] == F left = mid+1



"maximizing the min"

arr[m] == T left = mid+1

arr[m] == F right = mid-1

Binary Search on Answer

transitn pt → fn → boolean
- true
- false

Keywords
- maximize the minm
- minimize the maxm

LC 875) Koko Eating Bananas

piles = { 3, 7, 6, 11 }
indices: 0, 1, 2, 3

hours = 8

minspeed = ~~1~~ ~~2~~ ~~3~~ 4

is Possible	(27) 3+7+6+11 > 8	(15) 2+4+3+6 > 8	(10) 1+3+2+4 > 8	(8) 1+2+2+3 ≤ 8	(8) 1+2+2+3 ≤ 8	(6) 1+2+1+2 ≤ 8	(5) 1+1+1+2 ≤ 8
Speed	1	2	3	4	5	6	7
	X	X	X	✓	✓	✓	✓

linear search

```java
// true if it is possible to eat all bananas
// with the given speed in the given time else false
public boolean isPossible(int[] piles, int hours, int speed){
    long sum = 0;

    for(long bananas: piles){
        sum += bananas / speed;
        if(bananas % speed != 0) sum++; // ceil = floor + 1
    }

    return (sum <= hours);
}

public int minEatingSpeed(int[] piles, int hours) {
    long max = (int)1e9;

    // linear traversal: slowest speed to finish all bananas
    for(int speed = 1; speed <= max; speed++){
        if(isPossible(piles, hours, speed) == true)
            return speed;
    }

    return -1;
}
```

→ O(n)

O(n × max)

$10^4 \times 10^9 = 10^{13}$ TLE

→ O(max) * O(n)

piles = { 3, 7, 6, 11 }
0 1 2 3

hours = 8

{ BS on answer :→ speed }

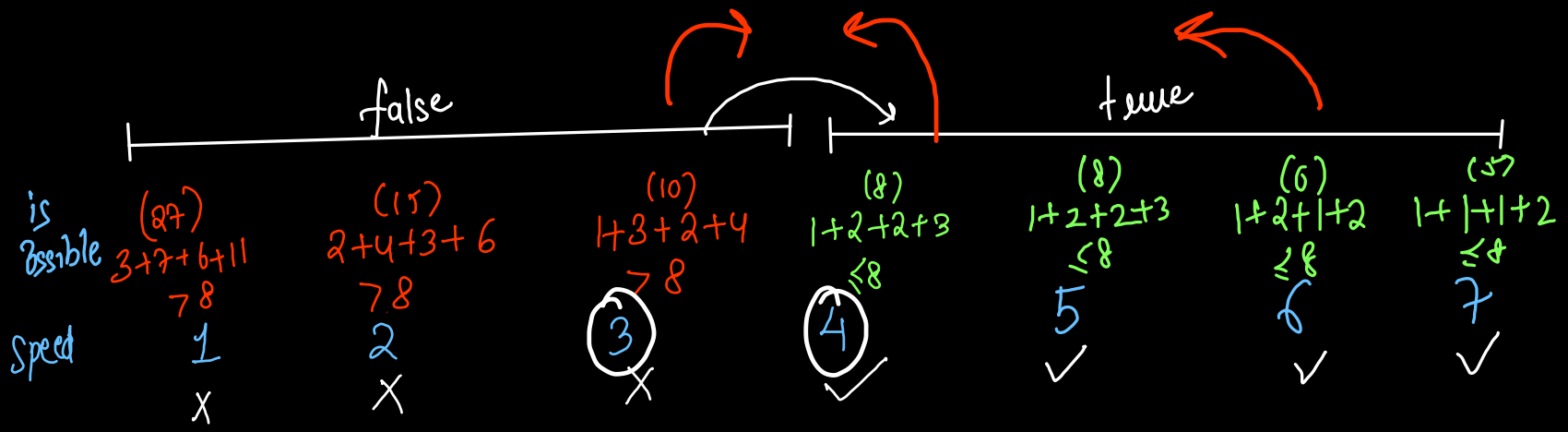
left = 1, right = 11, mid = (6) isPossible = true ⇒ r = m − 1

left = 1, right = 5, mid = (3) isPossible = false ⇒ l = m + 1

left = 4, right = 5 mid = (4) isPossible = true ⇒ r = m − 1

l = (4), r = (3)

return left;



```java
// true if it is possible to eat all bananas
// with the given speed in the given time else false
public boolean isPossible(int[] piles, int hours, int speed){
    long sum = 0;

    for(long bananas: piles){
        sum += bananas / speed;
        if(bananas % speed != 0) sum++; // ceil = floor + 1
    }

    return (sum <= hours);
}

public int minEatingSpeed(int[] piles, int hours) {
    long left = 1, right = (long)1e9;

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

        if(isPossible(piles, hours, (int)mid) == true){
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return (int)left;
}
```

O(n)

max

↳ O(n)

O(log max)

Time ⇒ O(n × log max)

$10^4 * \log 10^9$

$= 10^4 \times 27 \leq 10^8$

Accepted!

```java
// true if it is possible to eat all bananas
// with the given speed in the given time else false
public boolean isPossible(int[] piles, int hours, int speed){
    long sum = 0;

    for(long bananas: piles){
        sum += bananas / speed;
        if(bananas % speed != 0) sum++; // ceil = floor + 1
    }

    return (sum <= hours);
}

public int minEatingSpeed(int[] piles, int hours) {
    long left = 1, right = (long)1e9;

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

        if(isPossible(piles, hours, (int)mid) == true){
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return (int)left;
}
```

O(n)

Time $\Rightarrow O(n \times \log max)$

$10^4 * \log 10^9$

$= 10^4 \times 27 \leq 10^8$

O(logmax)

Accepted!

VVIMP

Hard

Allocate min no of pages

or

Book Allocatn Problem

or

Painter's Partition Problem

n books ⇒ {$P_0, P_1, P_2 \dots P_{N-1}$}

m students → all books needs to be contiguous allocatn

$n \geq m$ → each student atleast one book

"minimizing the max" =

You have **N** books, each with **Ai** number of pages. **M** students need to be allocated contiguous books, with each student getting at least one book. Out of all the permutations, the goal is to find the permutation where the student with the most pages allocated to him gets the minimum number of pages, out of all possible permutations.

→ $n < m$ (books < stud)

Note: Return **-1** if a valid assignment is not possible, and allotment should be in contiguous order (see the explanation for better understanding).

eg) books = 4 { 0: 20, 1: 10, 2: 40, 3: 30 }

Stud = 5
not possible ⇒ -1

Stud = 1

{ 20 + 10 + 40 + 30 = 100 } ← answer

Stud = 2

{ S1: 20 } { S2: 10 + 40 + 30 } ⇒ 80
{ 20 + 10 } { 40 + 30 } ⇒ 70
{ 20 + 10 + 40 } { 30 } ⇒ 70

answer = 70

Stud = 3

{ S1: 20 } { S2: 10 } { S3: 40 + 30 } ⇒ 70
{ 20 } { 10 + 40 } { 30 } ⇒ 50
{ 20 + 10 } { 40 } { 30 } ⇒ 40 ← answer

Stud = 4

{ 20 } { 10 } { 40 } { 30 } ⇒ 40

stud = ②, books = ④, pages = {20, 10, 40, 30}

bs ⇒ load/answer

left = 0, right = 100, mid = 50 ✗

isPossible → to distribute all books among maxm load as 50.

currload = {0+20+10}{40}{30}

noofstud = ∅ ∅ ∅ ③ > 2

2 students with false ⇒ l = m+1

left: 51, right = 100 ⇒ mid = 75 ✓

isPossible → to distribute all books among maxm load as 75.

currload = {0+20+10+40}{30}

noofstud = ∅ ∅ ② ≤ 2

2 students with true ⇒ r = m-1

stud = ②, books = ④, pages = {20, 10, 40, 30} (0, 1, 2, 3)

left = 51, right = 74, mid = 62 ✗

isPossible → to distribute all books among 2 students with maxm load as 62.

load = {0+20+10}{40}{30}

noofstud = 1 2 ③ 3 > 2 ⇒ false

l = m+1

left = 63, right = 74, mid = 68 ✗

isPossible → to distribute all books among 2 students with maxm load as 68.

load = {0+20+10}{40}{30}

noofstud = ③ > 2 ⇒ false

l = m+1

left = 69, right = 74, mid = 72 ✓

left = 69, right = 71, mid = 70 ✓

left = 69, right = 69, mid = 69 ✗ (false)

true {20+10+40}{30}

true {20+10+40}{30}

l = 70, r = 69

```java
public boolean isPossible(int[] books, int allowedStuds, long maxLoad){
    long currLoad = 0;
    int reqdStuds = 1;

    for(int pages: books){
        if(currLoad + pages <= maxLoad){
            currLoad += pages; // same student
        } else {
            currLoad = pages; // new student
            reqdStuds++;
        }
    }

    return (reqdStuds <= allowedStuds);
}
```

→ O(n)
↓
many books

```java
public long max(int[] books){
    long ans = 0;
    for(int pages: books){
        ans = Math.max(ans, pages);
    }
    return ans;
}
```

stud == books

```java
public long sum(int[] books){
    long ans = 0;
    for(int pages: books){
        ans += pages;
    }
    return ans;
}
```

stud == 1

```java
public int splitArray(int[] books, int studs) {
    long left = max(books), right = sum(books);

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

        if(isPossible(books, studs, mid) == true){
            right = mid - 1; // minimize the maxm value
        } else {
            left = mid + 1;
        }
    }

    return (int)left;
}
```

range

O(logrange)

*

Time ⇒ O(n log range)

10^5 10^{11}

$O(10^5 \times \log 10^{11}) \Rightarrow O(10^5 \times 11 \times 3)$

→ Accepted

{1, 4, 4} Sted = 3

left = 0, right = 9, mid = 4 ⇒ {1,3,4}{4} ✓

left = 0, right = 3, mid = 1 ⇒ currload = 0+1 (4)

(*) studs = 2

If any book is having pages > max load

⇒ allocation is never possible

```java
public boolean isPossible(int[] books, int allowedStuds, long maxLoad){
    long currLoad = 0;
    int reqdStuds = 1;

    for(int pages: books){
        if(pages > maxLoad) return false;
        if(currLoad + pages <= maxLoad){
            currLoad += pages; // same student
        } else {
            currLoad = pages; // new student
            reqdStuds++;
        }
    }

    return (reqdStuds <= allowedStuds);
}

public int splitArray(int[] books, int studs) {
    long left = 0, right = (long)1e11;

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

        if(isPossible(books, studs, mid) == true){
            right = mid - 1; // minimize the maxm value
        } else {
            left = mid + 1;
        }
    }

    return (int)left;
}
```

Time ⇒ O(n log max)

Space ⇒ O(1)

Split Array Largest Sum

LC 410

eg books = {20, 10, 40, 30}

allowed Stud = 3

mid(maxLoad) = 70

currload = ~~0+20+10+40~~ 30

reqStuds = ~~1~~ 2

Aggressive cows

You are given an array ***'arr'*** consisting of ***'n'*** integers which denote the position of a stall.

You are also given an integer ***'k'*** which denotes the number of aggressive cows.

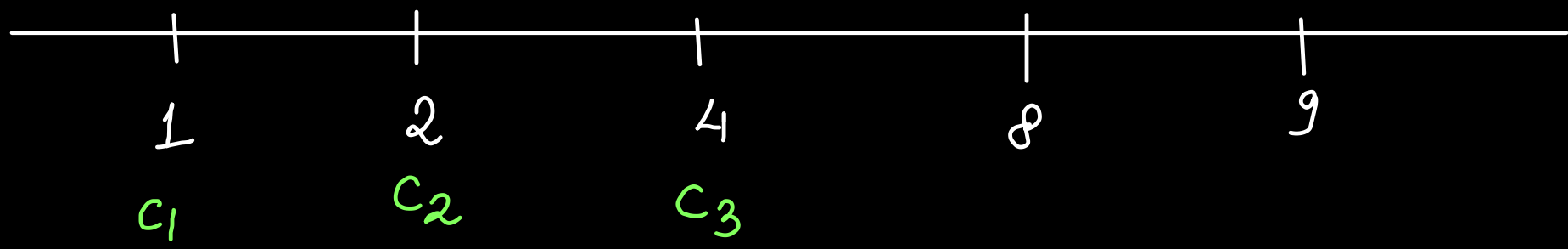
You are given the task of assigning stalls to 'k' cows such that the minimum distance between any two of them is the maximum possible.

Print the maximum possible minimum distance.

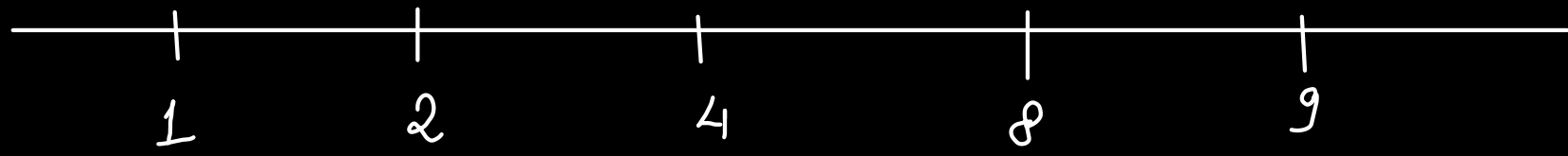
" maximising the min^m distance "

eg stalls = { 1, 2, 9, 8, 4 }

cows = 3



stalls = {1, 2, 4, 8, 9} Stalls should be sorted



we must keep first cow in first stall!

Way1) C_1 adj C_2 adj C_3 (1, 1) $\Rightarrow min^m = 1$

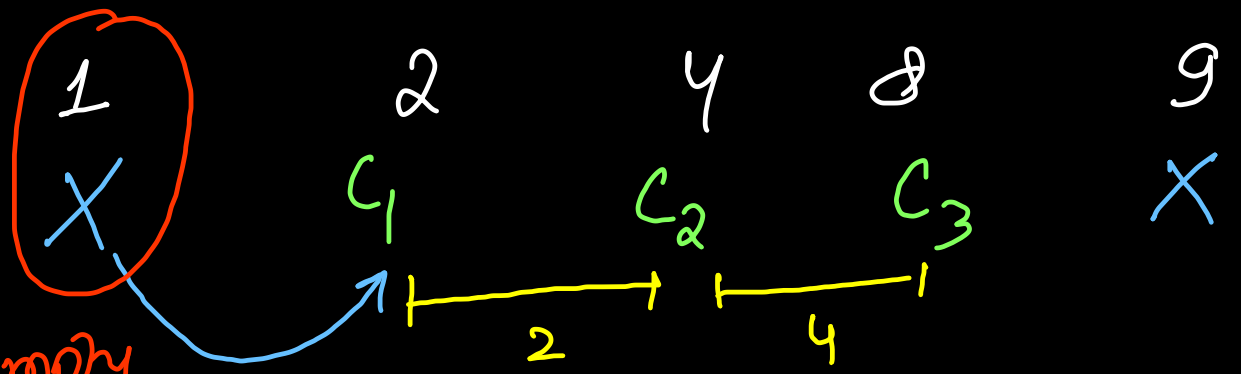
Way2) C_1 C_2 C_3 (1, 6) $\Rightarrow min^m = 1$

Way3) C_1 C_2 C_3 (1, 7) $\Rightarrow min^m = 1$

Way4) C_1 C_2 C_3 (3, 4) $\Rightarrow min^m = 3$ $\Rightarrow$ maximizing the min

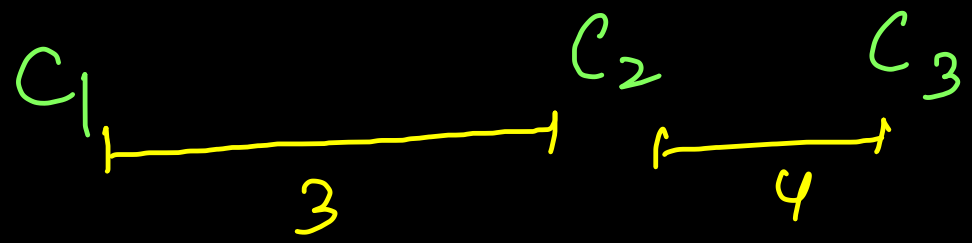
Way5) C_1 C_2 C_3 (3, 5) $\Rightarrow min^m = 3$

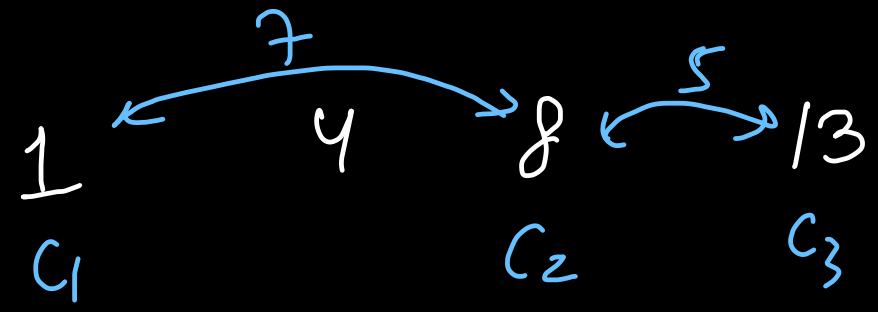
Way6) C_1 C_2 C_3 (7, 1) $\Rightarrow min^m = 1$



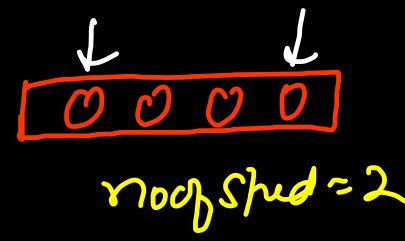
empty

↳ good choice X





test



stalls = {1, 2, 4, 8, 9} cows = ③

left = 0 , right = arr(n-1) - arr(0) = 9 - 1 = 8

mid = ④

is Possible → to assign stalls to all cows such that minm dist ≥ 4

cows = ~~1~~ 2

last = ~~arr(0) = 1~~ 8

2 < 3 ⇒ false

⇒ right = mid - 1

stalls = {1, 2, 4, 8, 9} cows = 3

left = 0, right = 3, mid = 1

isPossible ⇒ to assign stalls to all cows such that min dist >= 1

cows = 1 2 3 = 3

last = 1 2 4

true

left = mid + 1

stalls = {1, 2, 4, 8, 9} cows = 3

left = 2, mid = 3 ⇒ mid = 2 (dist)

isPossible ⇒ do assign stalls to all cows such that all min dist >= 2

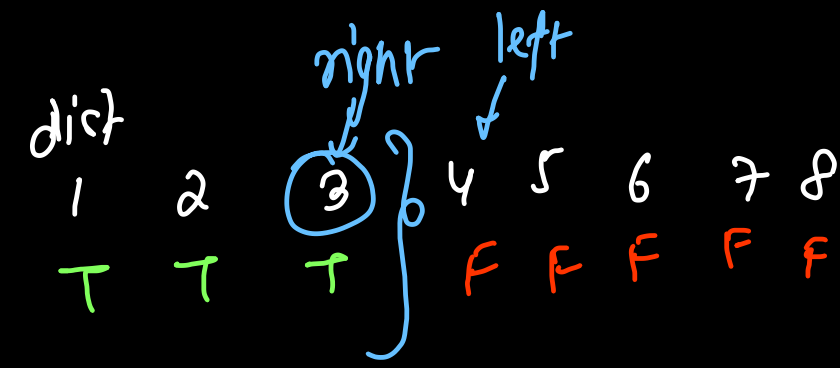
cows = 1 ≠ 3 == 3

true ⇒ left = mid+1

last = 1 4 8

stalls = {1, 2, 4, 8, 9} cows = 3

left = 3, right = 3, mid = 3



isPossible ⇒ to assign stalls to all cows such that all min dist >= 3

cows = 1 2 3 = 3

last = 1 4 8

true

left = mid + 1 = 4

right = 3

```java
public boolean isPossible(int[] stalls, int cows, int dist){
    int placed = 1, last = stalls[0];

    for(int curr: stalls){
        if(curr - last >= dist) {
            placed++;
            last = curr;
        }
    }

    return (placed >= cows);
}

public int maxDistance(int[] stalls, int cows) {
    Arrays.sort(stalls);
    int n = stalls.length;

    int left = 0, right = stalls[n - 1] - stalls[0];
    while(left <= right){
        int mid = left + (right - left) / 2;

        if(isPossible(stalls, cows, mid) == true){
            left = mid + 1; // maximize the minimum distance
        } else {
            right = mid - 1;
        }
    }

    return right;
}
```

→ O(n) (no of stalls)

→ n log n

→ O(log range) (array range)

Time ⇒ $O(\underset{\downarrow}{n}\log\underset{\downarrow}{range})$

n: 10^5, $range$: 10^9

Practice Class

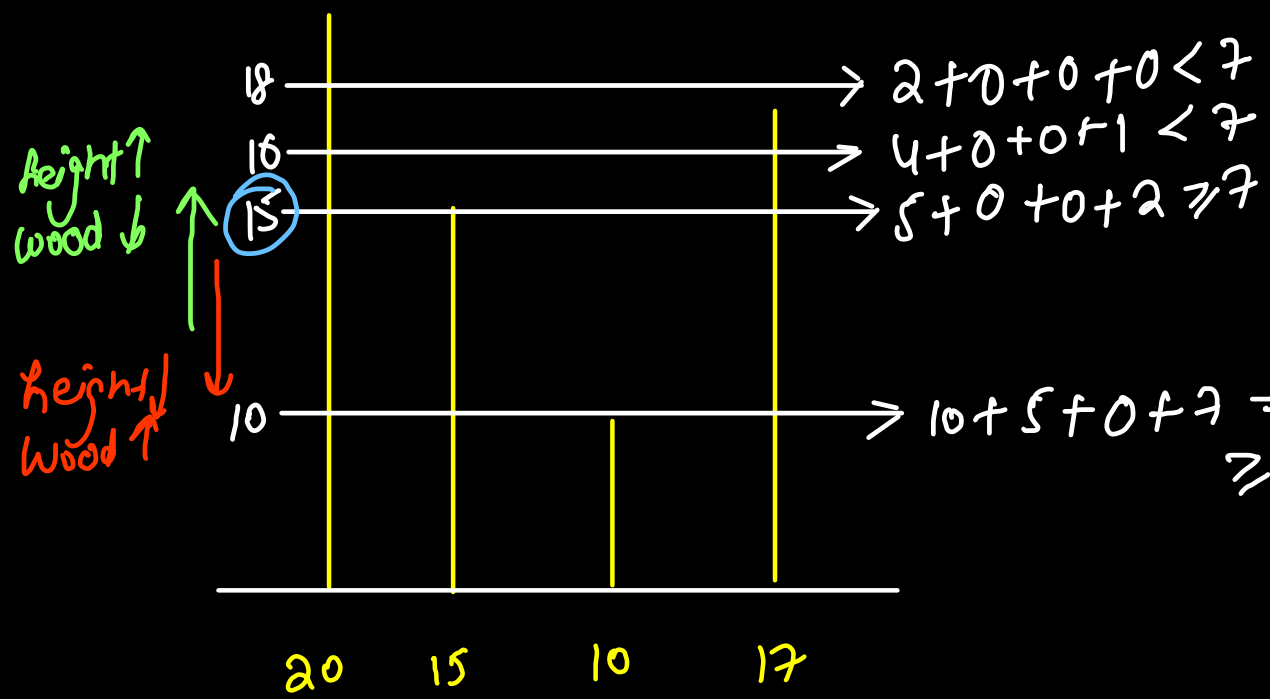
- BS on Answer (Koko, Gas Station, Woodcutting)
- +
- Median of 2 sorted Arrays (Hard)

⋆ Woodcutting

reqd woods = 7m

max height = ?

max height ⟶ min wood requirement?



left = 0 right = 20 (max of array)

mid = 10 isPossible = true ⇒ l = m+1

left = 11, right = 20

mid = 15 isPossible = true ⇒ l = m+1

left = 16, right = 20

mid = 18 isPossible = false ⇒ r = m-1

left = 16, right = 17, isPossible(mid=16) = false ⇒ r = m-1

left = 16, right = 15 ⇐ answer

```java
public boolean isPossible(int[] trees, long reqdWood, long cut){
    long wood = 0;

    for(int tree: trees){
        if(tree > cut) wood += (tree - cut);
    }

    return (wood >= reqdWood);
}

public int solve(int[] trees, int reqdWood) {
    long left = 0, right = (long)1e6;

    while(left <= right){
        long mid = left + (right - left) / 2;
        if(isPossible(trees, reqdWood, mid) == true){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return (int)right;
}
```

O(n) → no. of trees

O(log-range)

integer overflow ↓ long datatype

Time ⇒ O(n log range)

Space ⇒ O(1)

0	1	2	3	4	5
10	15	25	34	72	80

$k = 8$ (extra stations)

d = minimum possible value of max distance of adj gas stations

$k=0 \Rightarrow |72-34| = 38$

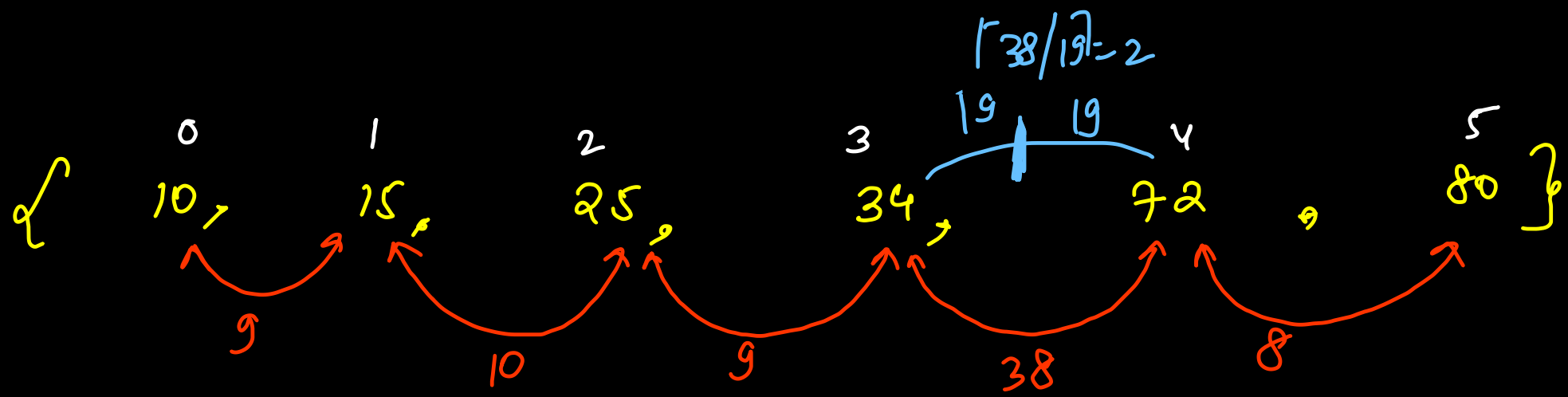
$k=1 \Rightarrow |72-53| = 19$

$k=2 \Rightarrow 38/3 = 12.67$

$k=3 \Rightarrow |25-15| = 10$

$k=4 \Rightarrow 38/4 = 9.5$

$k=5 \Rightarrow |34-25| = 9$



left = 0 , right = 38 , mid = 19

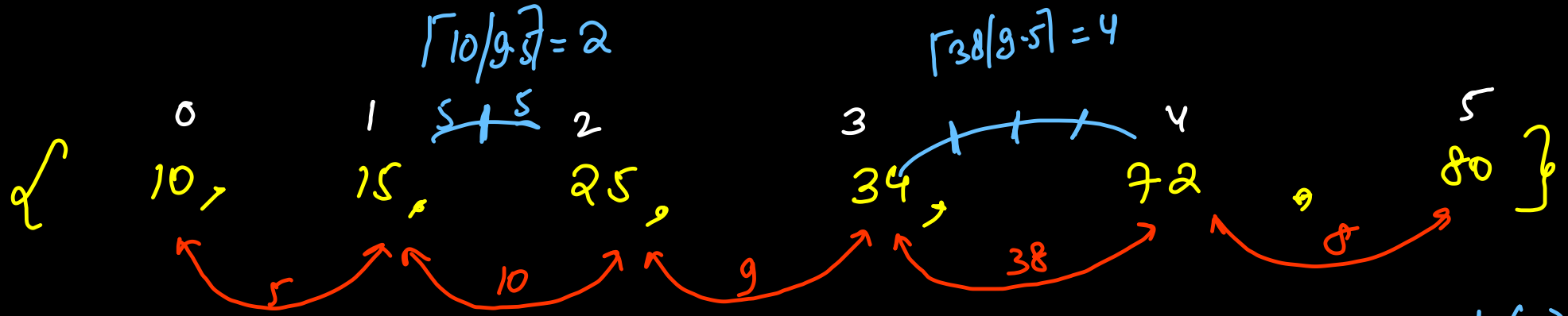
isPossible = to add 4 stations such that max dist b/w adjacent stations is 19

reqd = 0 + 1 ≤ 4

allowedStations(k) - 4

isPossible = true

↳ right = mid ⊛



left = 0 , right = 19 , mid = 9.5

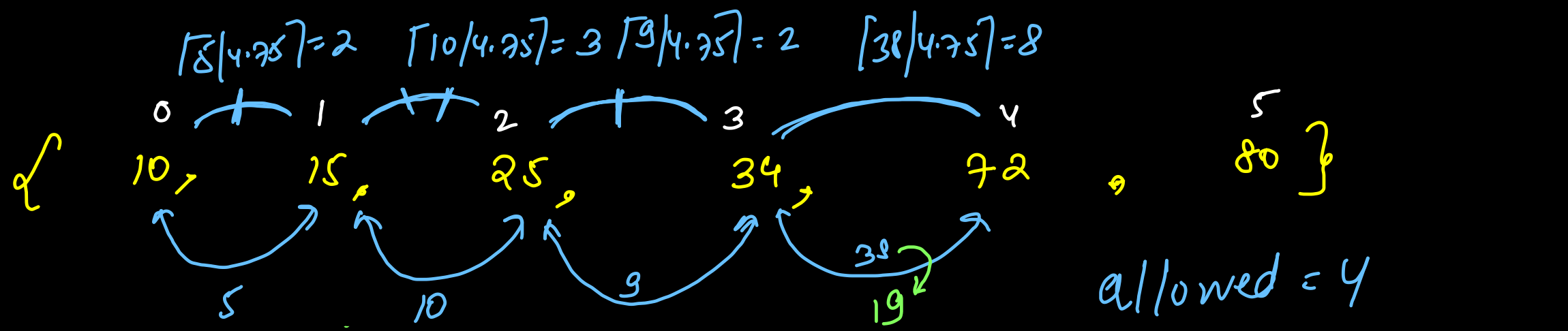
isPossible = to add 4 stations such that max dist b/w adjacent stations is 9.5

allowed (k) = 4

reqd = (0 + 1 + 3) ≤ 4

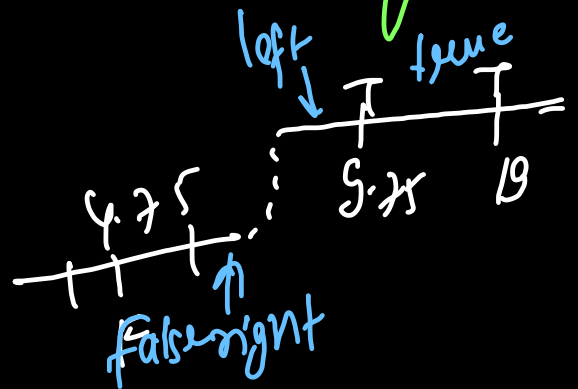
true

⇒ right = mid



left = 0 , right = 9.5 , mid = 4.75

isPossible = to add 4 stations such that max dist b/w adjacent stations is 4.75



allowed = 4

reqd = 0 + 1 + 2 + 1 + 7

> 4

false

⇒ left = mid.

```java
public static int maxAdjacent(int[] stations){
    int ans = 0;
    for(int i = 0; i < stations.length - 1; i++){
        ans = Math.max(ans, stations[i + 1] - stations[i]);
    }

    return ans;
}

public static boolean isPossible(int[] stations, int allowedStations, double dist){
    int reqdStations = 0;

    for(int i = 0; i < stations.length - 1; i++){
        double gap = (stations[i + 1] - stations[i]) / dist;
        reqdStations += (int)Math.ceil(gap) - 1;
    }

    return (reqdStations <= allowedStations);
}

public static double findSmallestMaxDist(int stations[], int allowedStations) {
    Arrays.sort(stations); // adjacent distance

    double left = 0.0, right = maxAdjacent(stations);

    while(right - left > 1e-6){
        double mid = left + (right - left) / 2.0;

        if(isPossible(stations, allowedStations, mid) == true){
            // minimize the max adjacent distance
            right = mid;
        } else {
            left = mid;
        }
    }

    return left;
}
```

→ O(n)

→ O(logrange)

(*) double binary search

$l = m+1 \longrightarrow l = m$

$r = m-1 \longrightarrow r = m$

$l \le r \longrightarrow r - l > 10^{-6}$

Time :: O(nlogrange)

Square Root (Decimal)

$n = 16 \Rightarrow \sqrt{n} = 4.00$

$n = 20 \Rightarrow \sqrt{n} = 4.47$

$n = 25 \Rightarrow \sqrt{n} = 5.00$

$l = 0,\ r = 20,\ m = 10.0$

$10.0 \times 10.0 > 20$ false $\Rightarrow r = m$

$l = 0,\ r = 10,\ m = 5.0$

$5.0 \times 5.0 > 20$ false $\Rightarrow r = m$

$l = 0,\ r = 5,\ m = 2.5$

$2.5 \times 2.5 \leq 20$ true $\Rightarrow l = m$

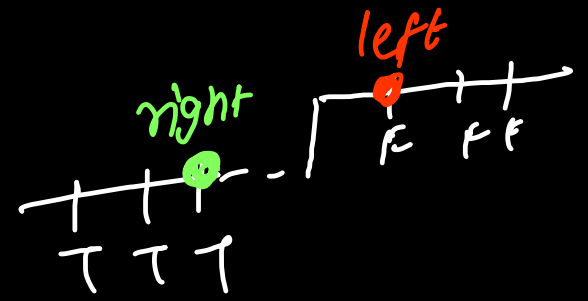
$l = 2.5,\ r = 5.0,\ m = 3.75$

$3.75 \times 3.75 \leq 20 \Rightarrow T \Rightarrow l = m$

$l = 3.75,\ r = 5.0,\ m = 4.375$

$4.375 \times 4.375 \leq 20 \Rightarrow T \Rightarrow l = m$

while $(r - l) > 1e-d$



```java
public static double squareRoot(long n, int d) {
    double left = 0.0, right = 1e15;
    double precision = Math.pow(10, -d);

    while(right - left >= precision){
        double mid = left + (right - left) / 2.0;

        if(mid * mid <= n){
            // potential answer
            left = mid;
        } else {
            right = mid;
        }
    }

    return left;
}
```

Task 7

issues resolve!