Linear Search

Unsorted / Sorted

firstOccurrence / lastOccurence / all occurences

Best case → $\Omega(1)$ first iteration

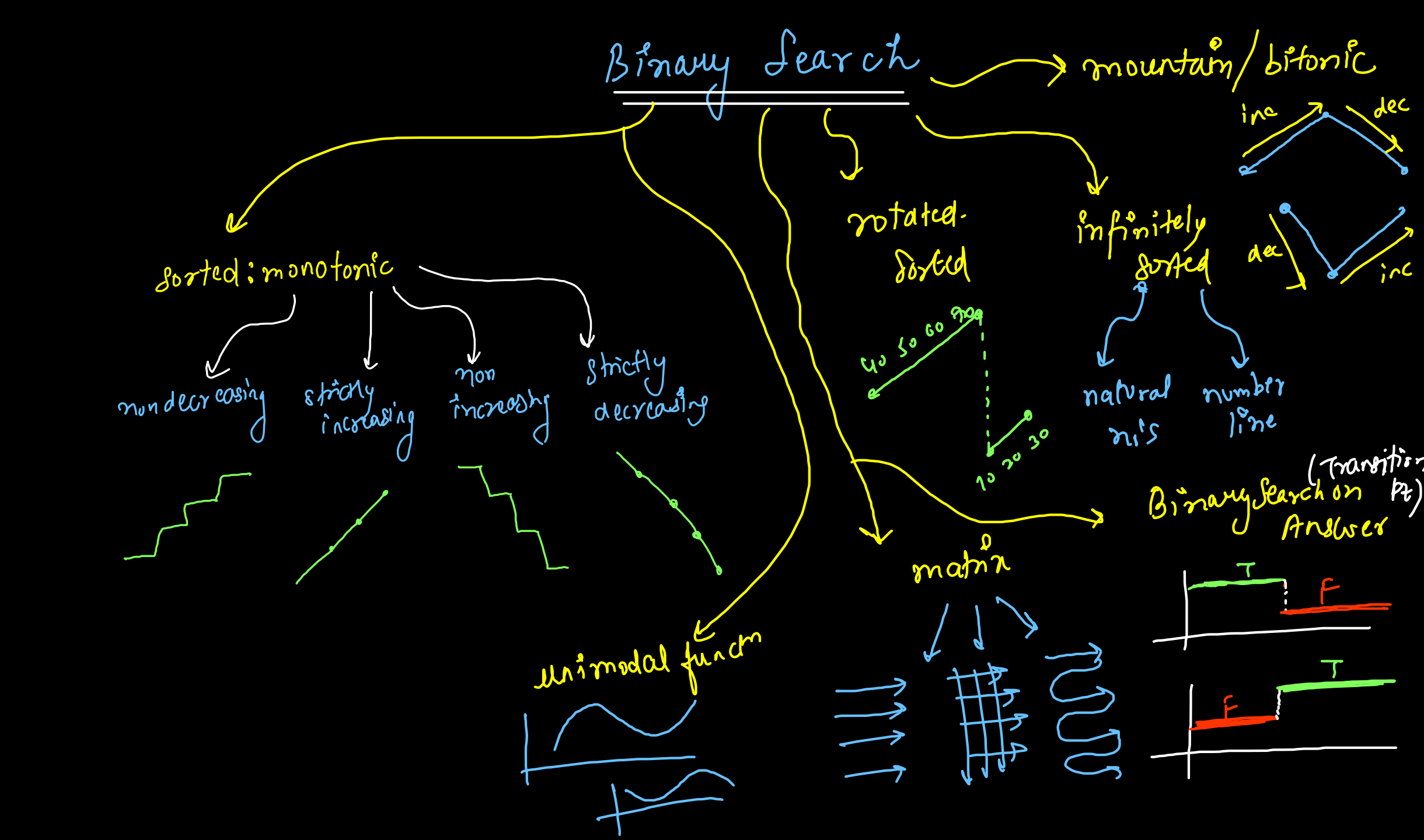
Avg case → $\Theta(n)$

Worst case → $O(n)$ last iteration

Search space after each iteration

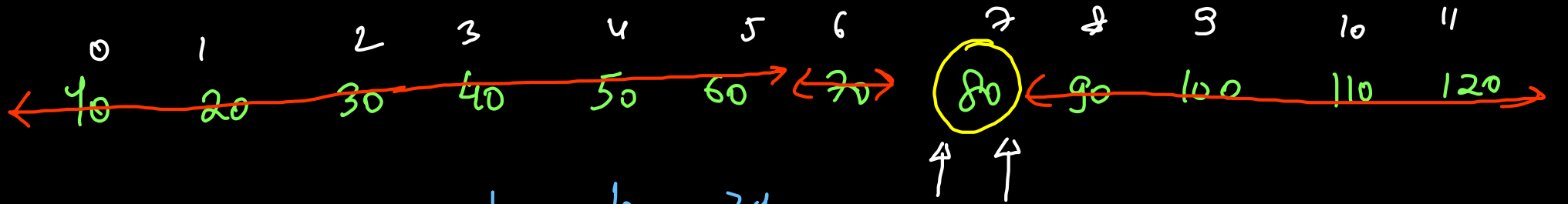
$N \longrightarrow N-1$

Binary Search



Binary Search

↳ Divide & Conquer



$n \to n/2 \to n/4 \to n/8 \to \cdots \to 1$

↳ $O(\log_2 n)$

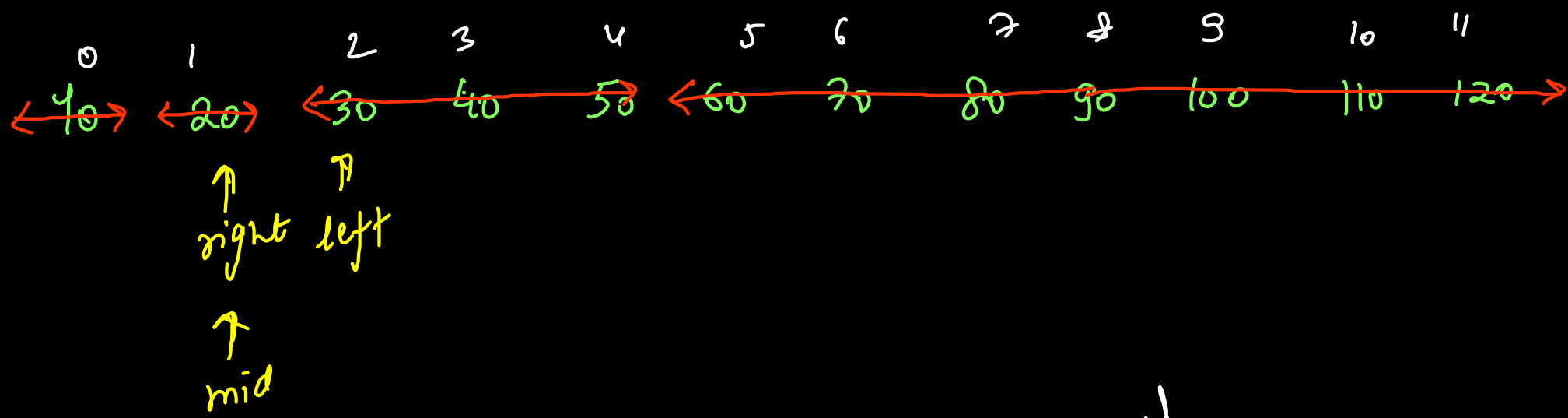
target = 20

① 60 < 80

② 90 > 80

③ 70 < 80

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



target = 25

60 > 25
30 > 25
10 < 25
20 < 25

unsuccessful search

$O(\log_2 N)$

```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; // successful search
        else if(nums[mid] < target) left = mid + 1;
        else right = mid - 1;
    }

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

Leetcode 704

Time

Bestcase : $\Omega(1)$

Avgcase : $\Theta(\log n)$

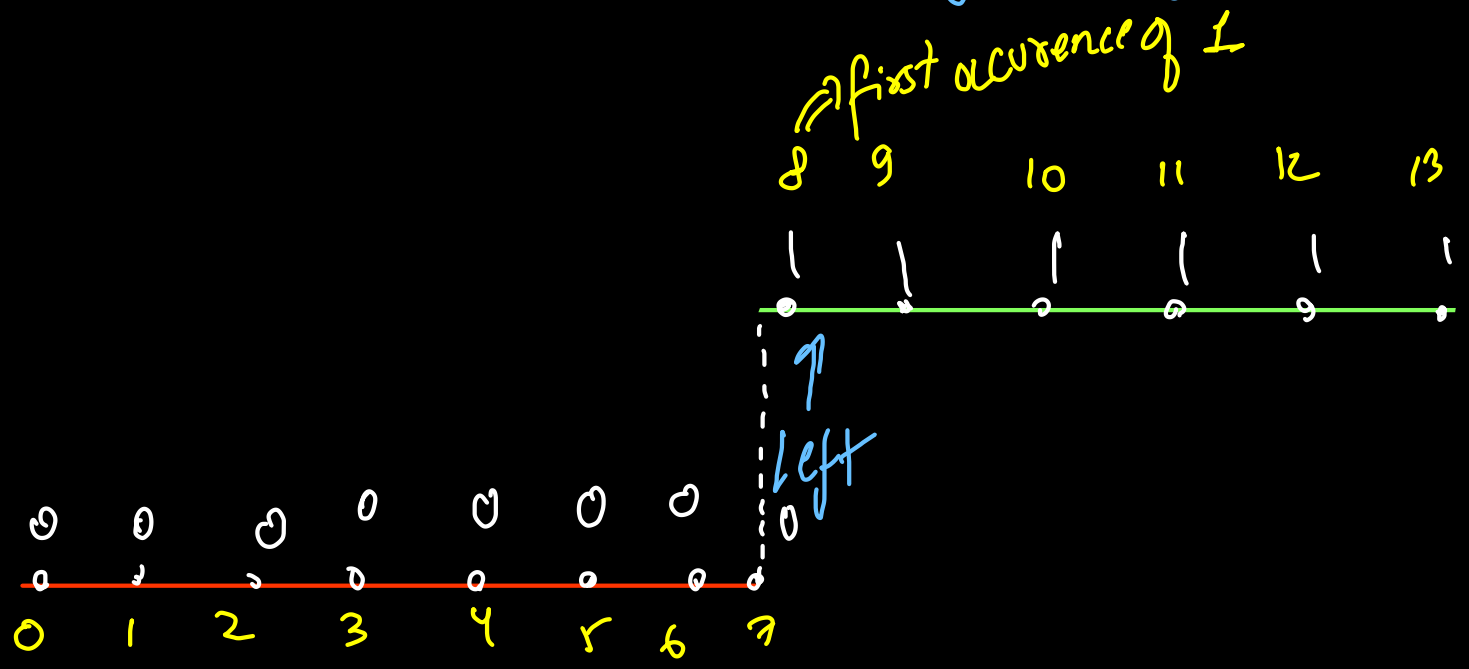
Worstcase : $O(\log n)$

Space : $O(1)$

inplace

Transition Point

Sorted Binary Array

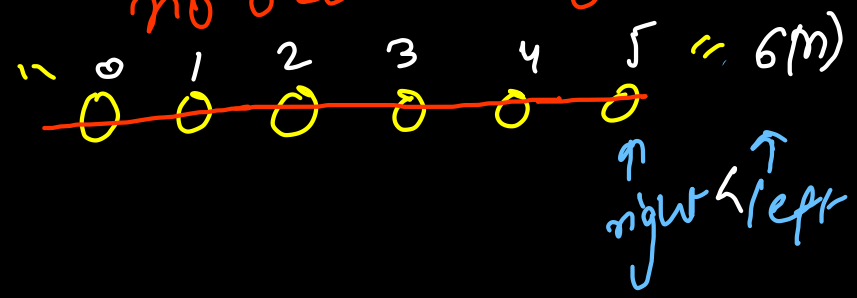



```
while (left <= right){
    if(arr[mid]==0)
        left = mid+1
    else  right = mid+1
}
return left.
```

```
int transitionPoint(int arr[], int n) {
    if(n == 0 || arr[n - 1] == 0) return -1;

    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;
}
```

corner case

no occurrence of 1s



Aditya's Doubt

left = 1 right = 1

(left + right)/2 = (1+1)/2 = 2/2 = 1

left/2 + right/2 = 1/2 + 1/2

= 0 + 0 = 0

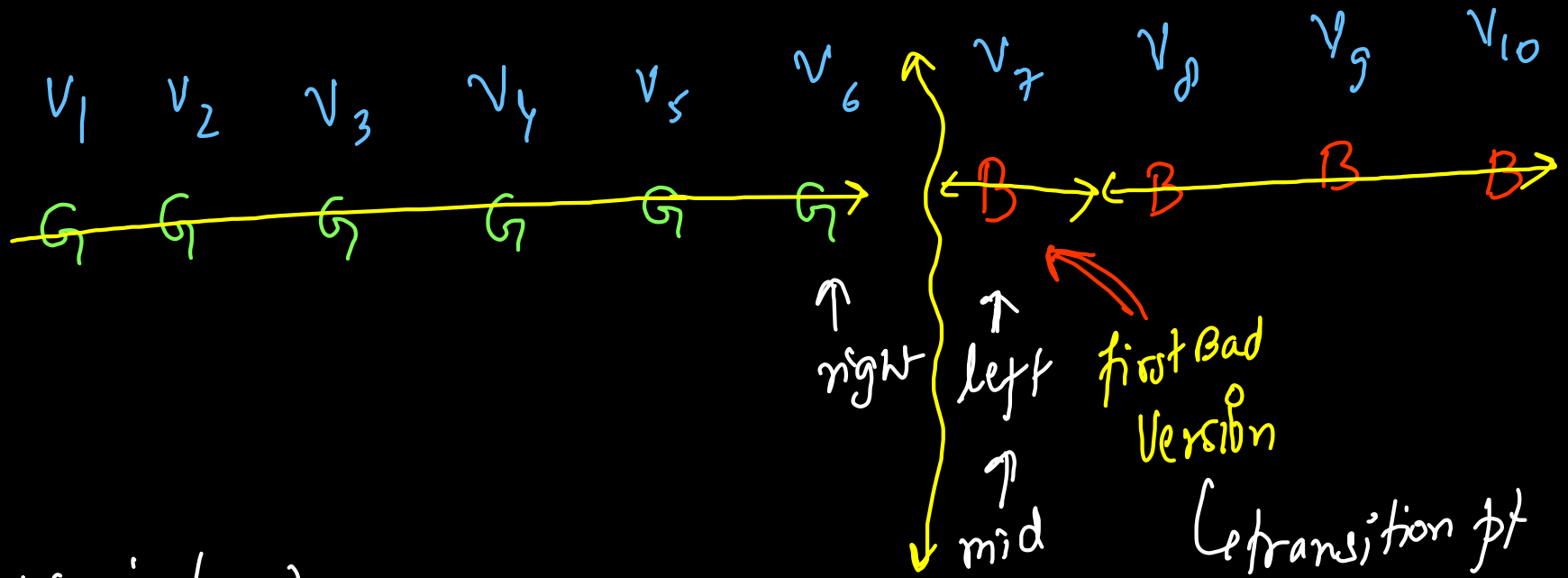
First Bad Version {Google}

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have `n` versions `[1, 2, ..., n]` and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API `bool isBadVersion(version)` which returns whether `version` is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

eg $n = 10$



isBadVersion(mid)

true → Bad

false → Good

transition pt

⇓

Binary Search

```java
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {
        int left = 1, right = n;

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

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

        return left;
    }
}
```

natural no's

TLE (wrong)

Acc to Constraint

$n = 2^{31} - 1$ = Integer.MAX_VALUE

1 G G G ..... G B n

l r

$l = 1 = n$

$r = 2^{31} - 1 = n$

l

$(left + right)/2$

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

$= (-garbage)/2 = -ve$ → overflow

r

$left + (right - left)/2$

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

not overflowing

$left + (right - left)/2$

$left + right - left$ / 2

$left + right$ / 2

```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;
}
```

Accepted

Time → O(log n)

Space → O(1)

Google

recession {us companies}

hiring slowdown
or
hiring freeze
} freshers → job → startups, lower pay, offline

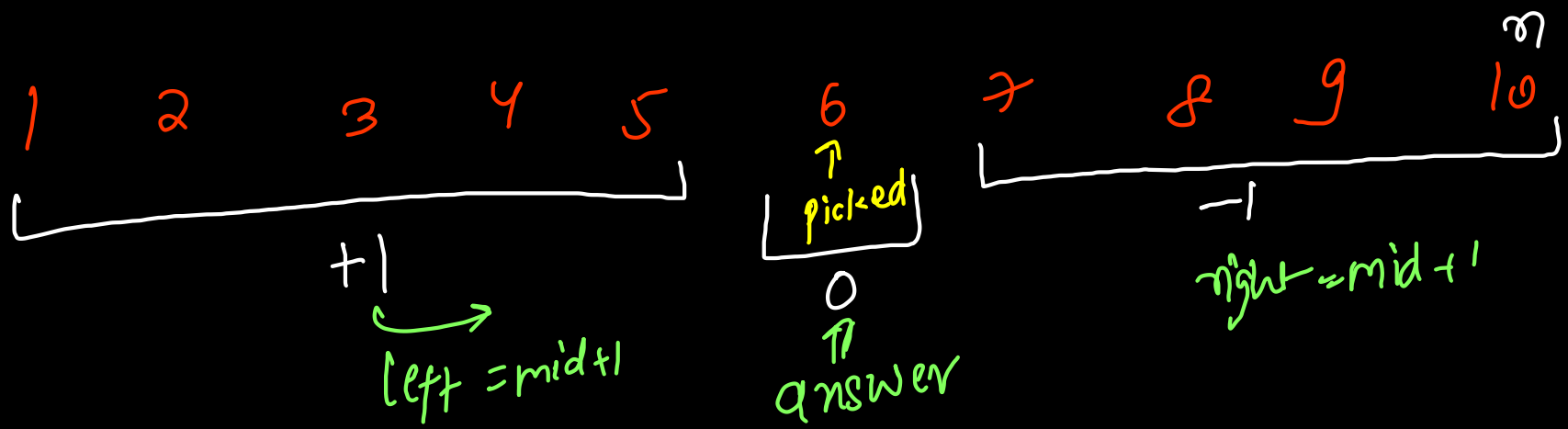
or

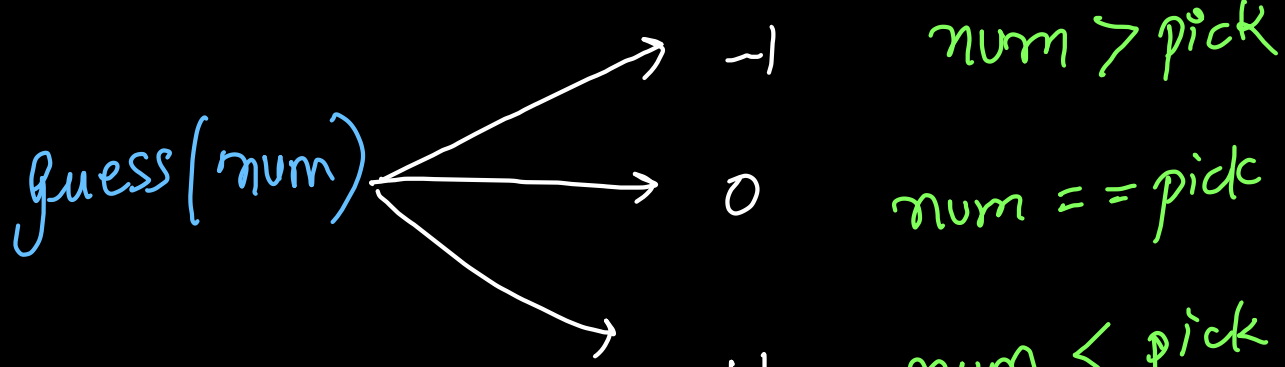
layoffs } job switch

tech stack
{ frontend → backend }

↳ backup
offer letter
↓
interviews

Guess No Higher or Lower





```java
public class Solution extends GuessGame {
    public int guessNumber(int n) {
        int left = 1, right = n;

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

            int choice = guess(mid);
            if(choice == 0) return mid;
            else if(choice == 1) left = mid + 1;
            else right = mid - 1;
        }

        return -1;
    }
}
```

Time → $O(\log_2 n)$

Space → $O(1)$

Lower Bound
vs
Upper Bound

Flour
vs
Ceil

First Occurence
vs
Last Occurence

Binary Search with Duplicates, Sorted

10	10	20	20	20	20	30	50	50	50	60	60	70	80	80	80
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

First occurence (10) = 0

Last occurence (10) = 1

leftmost
FirstOcc(20) = 2

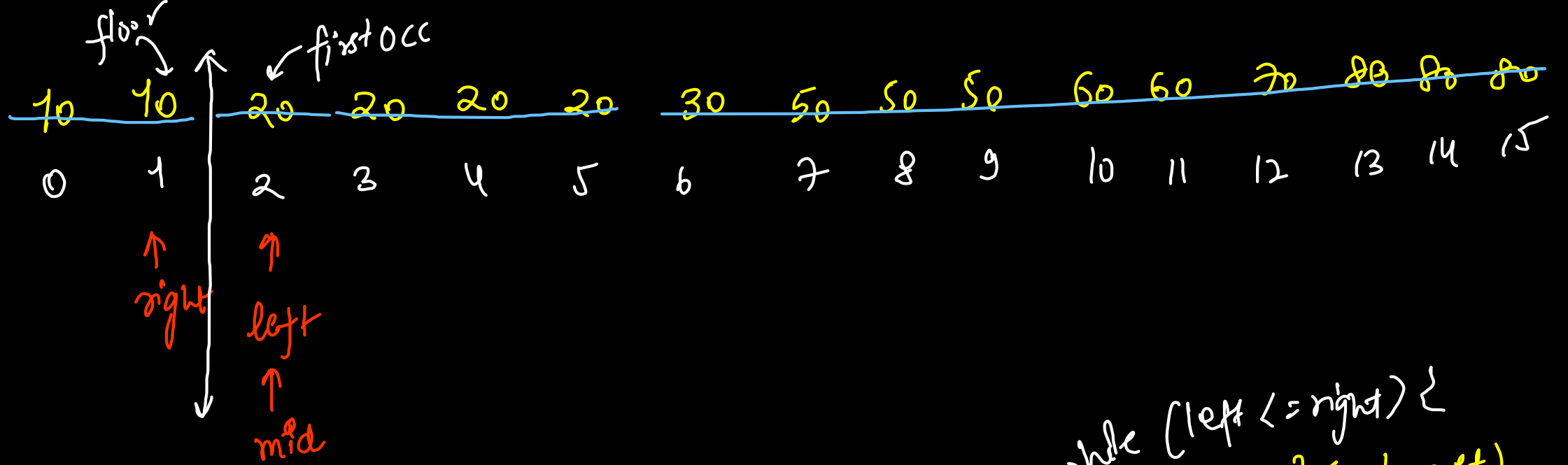
LastOcc(20) = 5
rightmost

FistOcc(30) = 6

LastOcc(30) = 6
Single occ

FistOcc(40) = -1

LastOcc(40) = -1
target not occ

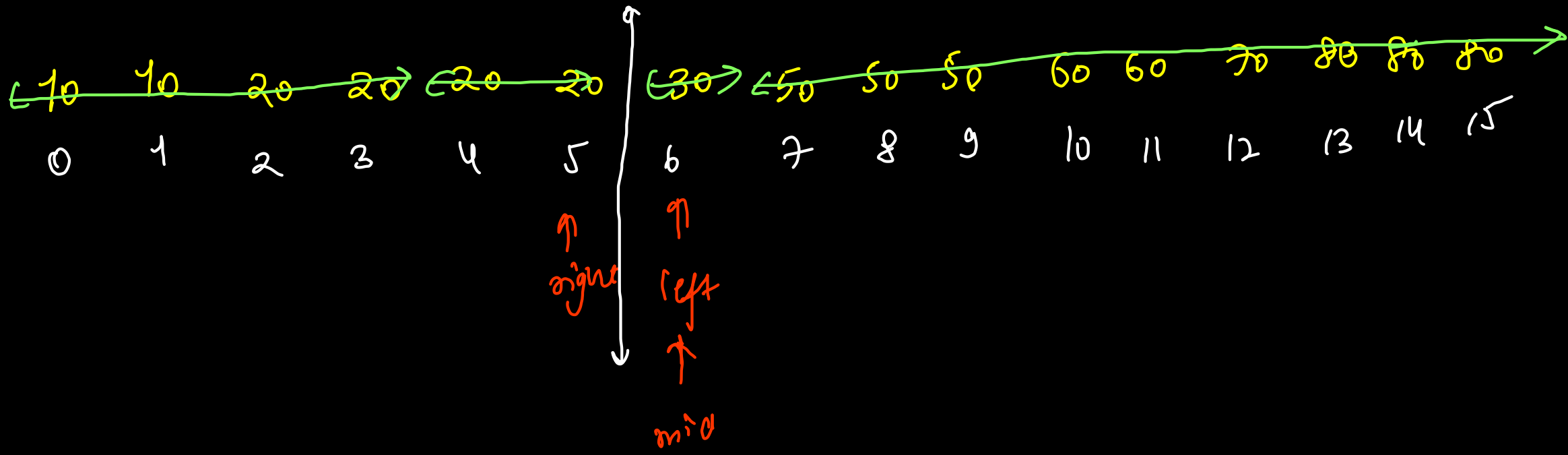


target = 20

firstOcc(20) = ②

```
if(arr[mid] == target)
    right = mid-1;
else if(arr[m] < target)
    left = midth;
else
    right = mid-1;

return left;
```

```
while (left <= right) {
    if(arr[mid] <= target)
        right = mid-1
    else left = midth;
}
return left;
```



lastOcc(20) = 5

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

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

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

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

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

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

    return left;
}
```

$O(\log_2 n)$

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

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

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

    return right;
}
```

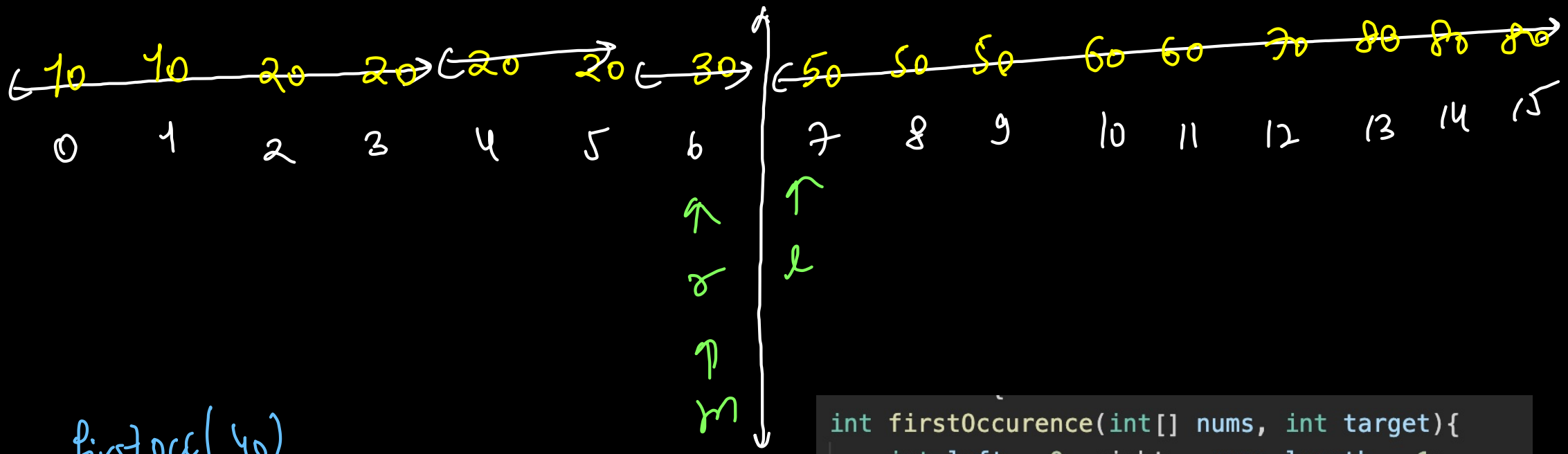
$O(\log_2 n)$

```
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};
}
```

target not found

$O(2\log_2 n)$

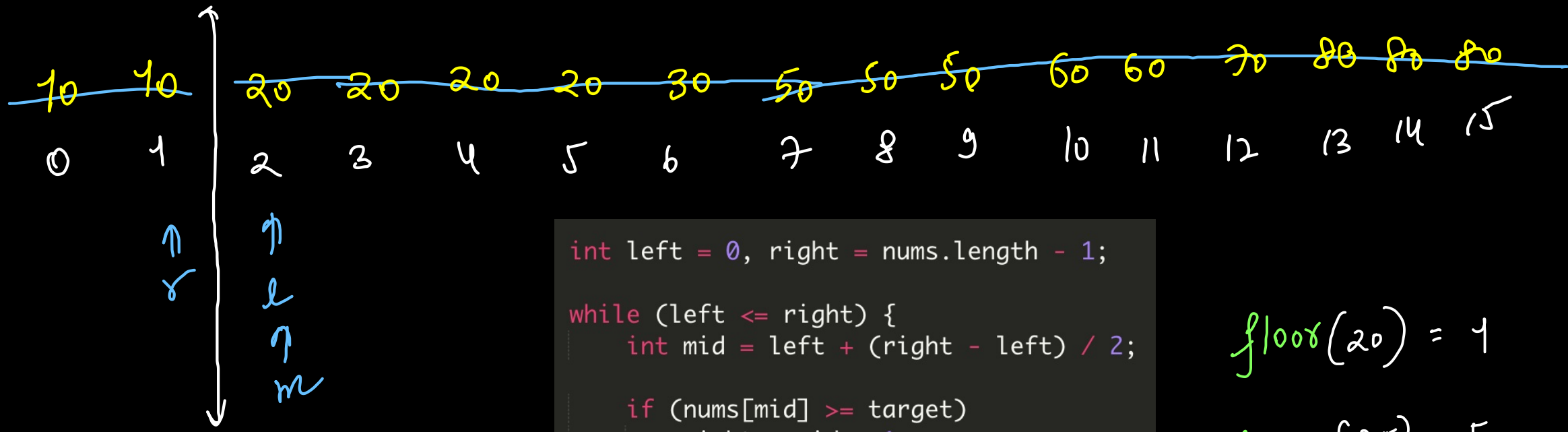
$\approx O(\log_2 n)$



firstOcc(40)

lastOcc(40)

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

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

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

    return left;
}
```




```
int left = 0, right = nums.length - 1;

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

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

return right;
```

if (arr[mid] == target)
right = mid - 1

else if (arr[mid] < target)
left = mid + 1

else
right = mid - 1

floor(20) = 4

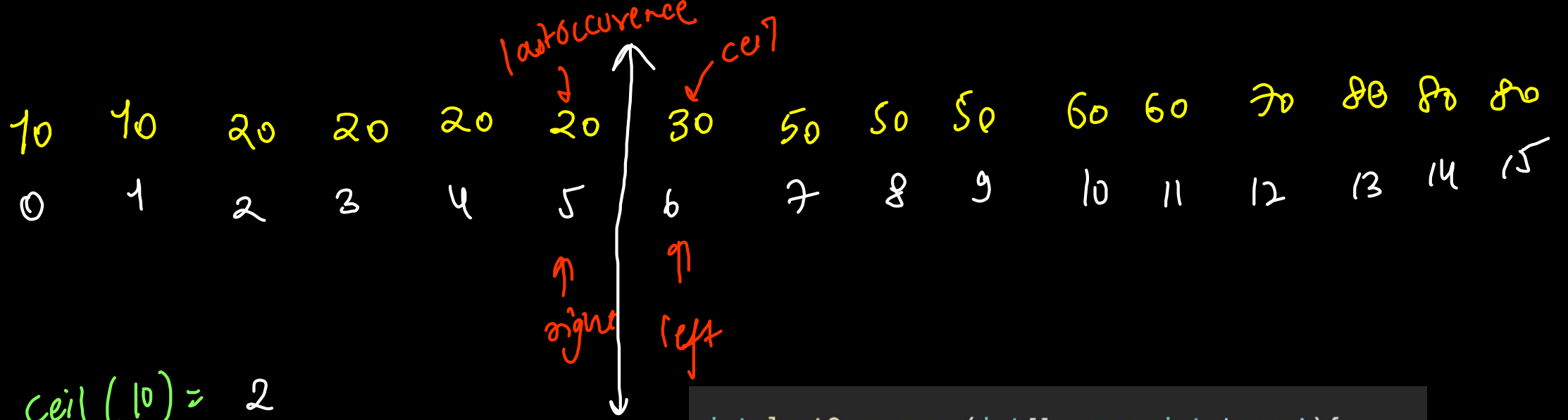
floor(25) = 5

floor(80) = 6

floor(100) = 15

floor(10) = -1

floor(0) = -1



ceil (10) = 2

ceil (20) = 6

ceil(25) = 6

ceil (80) = 16 (arr.length)

ceil (100) = 16 (arr.length)

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

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

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

    return left;
}
```

floor (on GFG)

```
static int findFloor(long nums[], int n, long target)
{
    int left = 0, right = n - 1;

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

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

    return right;
}
```

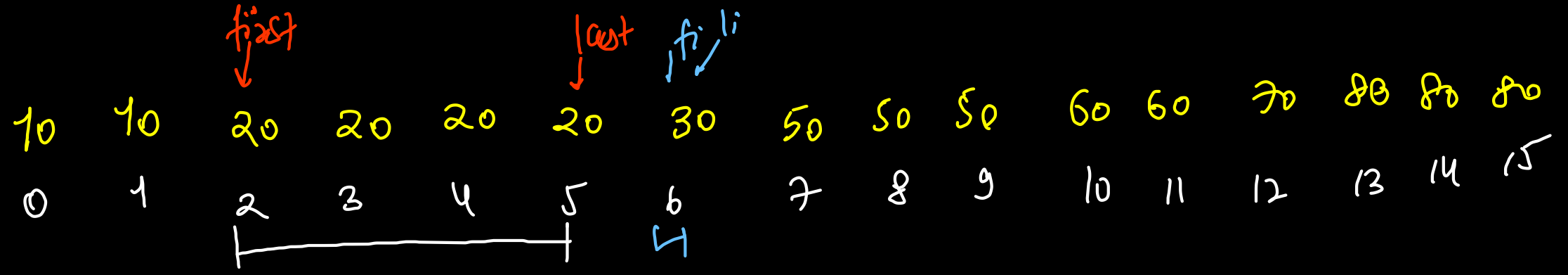
Acc to GFG floor[x] $\leq x$

Ceil (on Codestudio)

```
int left = 0, right = nums.length - 1;

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

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

if(left == nums.length) return -1;
return nums[left];
```

Acc to Codestudio

ceil[x] $\geq x$



Count occurrence (20) = li - fi + 1;

```
class Solution {
    int firstOccurence(int[] nums, int target) {}

    int lastOccurence(int[] nums, int target) {}

    int count(int[] nums, int n, int target) {
        int fi = firstOccurence(nums, target);
        int li = lastOccurence(nums, target);
        return li - fi + 1;
    }
}
```

O(log n)

O(log n)

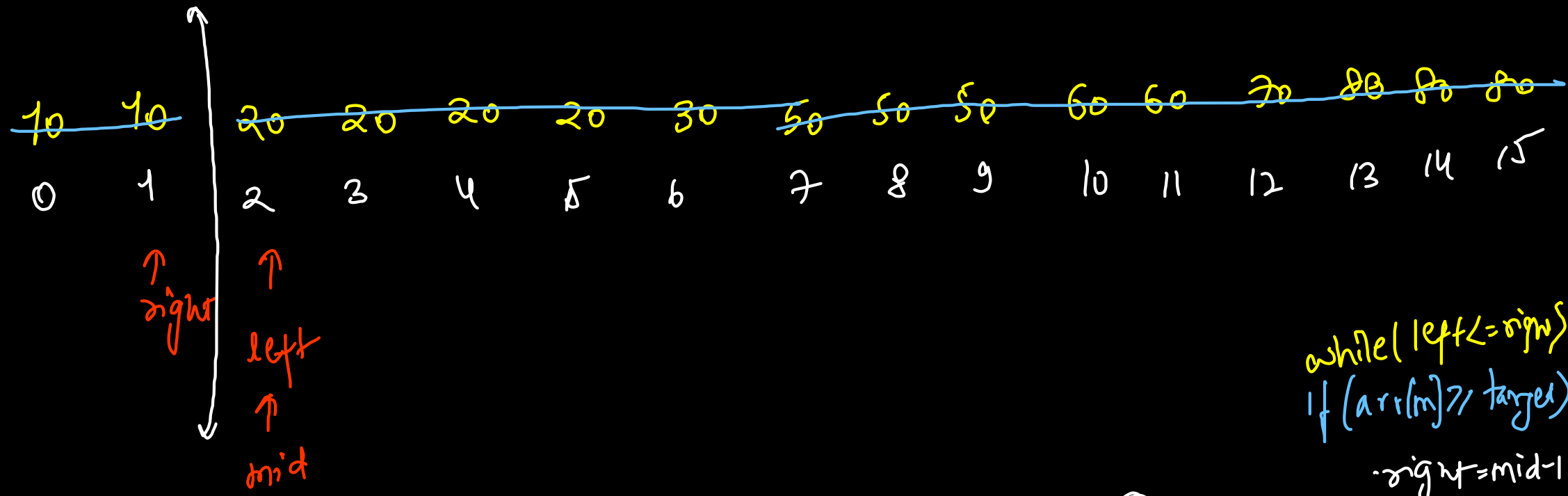
O(log n)

Lower Bound :→ If target is found ⇒ first occurrence

else target is not present ⇒ ceil

Upper Bound :→ Always the ceil value

* If target is not found ⇒ Lower Bound == upper bound



lowerBound (20) = 2

lowerBound (30) = 6

lowerBound (80) = 13

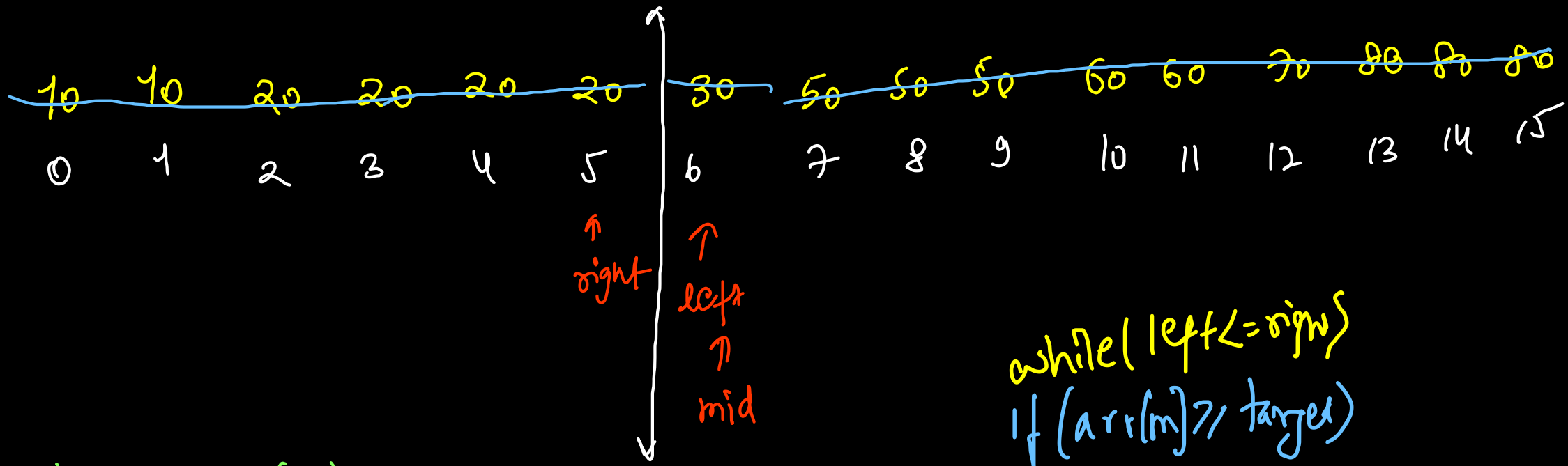
lowerBound (5) = 0

lowerBound (15) = 2

lowerBound (25) = 6

lowerBound (85) = arr.length or -1

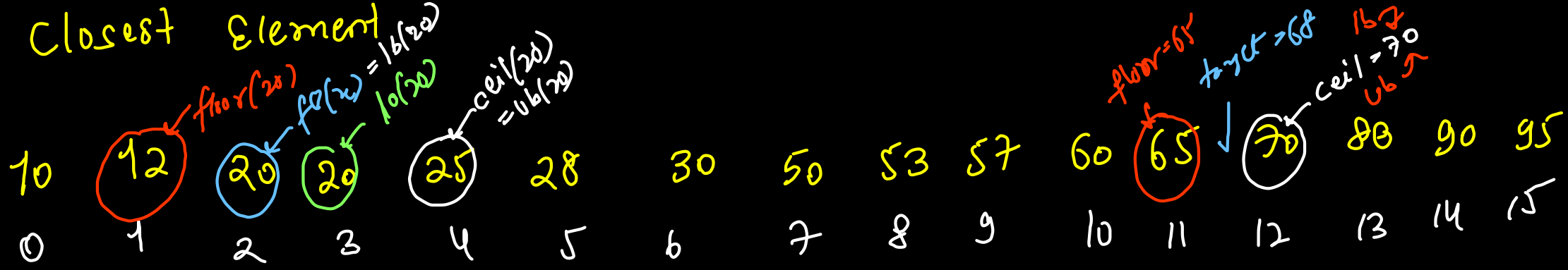
```
while(left<=right)
if (arr[m]>= target)
    right=mid-1
else
    left=mid+1
}
return left;
```



lower Bound (25) =

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

Closest Element



target = 45

- 30 → |45-30| = 15
- 50 ✓ → |50-45| = 5

target = 60

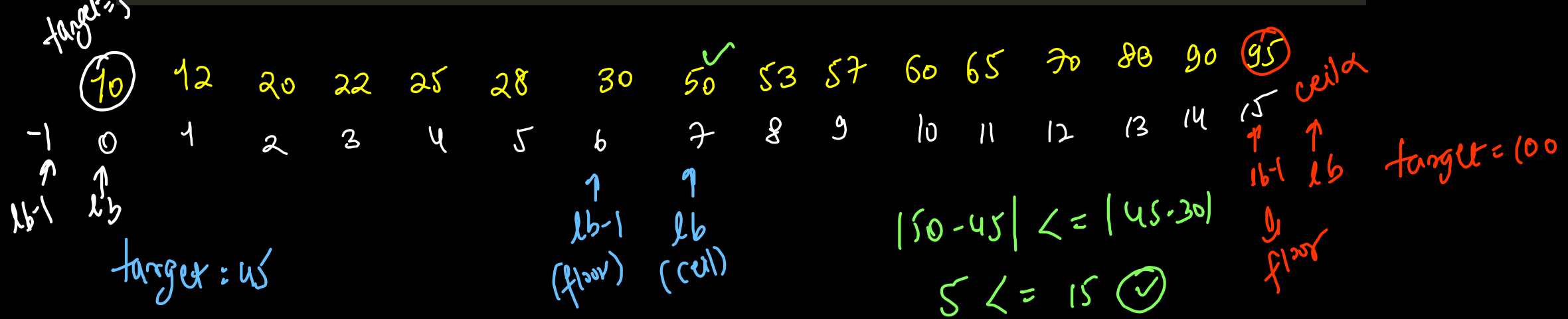
- 60 ✓ → |60-60| = 0
- 65 → |65-60| = 5

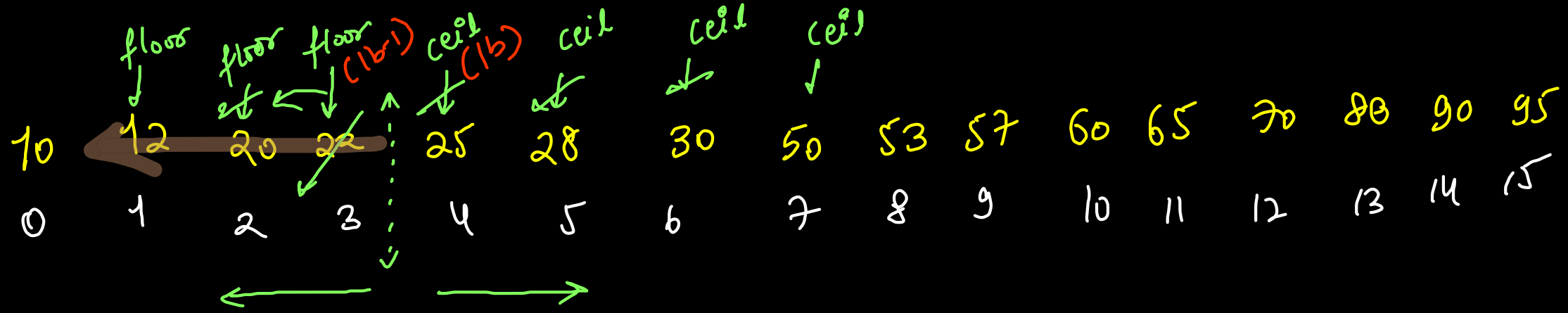
lowerBound → target ✓ → firstAcc
→ ceil

vs

lowerBound - 1 ⇒ floor

```java
public static int lowerBound(int[] nums, int target) {}

public static int findClosest(int arr[], int n, int target)
{
    int lb = lowerBound(arr, target);

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

    if(arr[lb] - target <= target - arr[lb - 1]) return arr[lb];
    else return arr[lb - 1];
}
```





target = 23

22, 25, 20, 28, 30

k = 5

```java
public int lowerBound(int[] nums, int target) {
    // Same as First Occurence
    int left = 0, right = nums.length - 1;

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

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

    return left; // if target -> first occ, else ceil
}
```

O(logn)

Time

O(logn + k + klogk)

↑ BS

↑ 2Pointer

↑ Collections.sort(res)

```java
public List<Integer> findClosestElements(int[] nums, int k, int target) {
    int right = lowerBound(nums, target);
    int left = right - 1;

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

    while(left >= 0 && right < nums.length && res.size() < k){
        if(target - nums[left] <= nums[right] - target){
            res.add(nums[left]);
            left--;
        } else {
            res.add(nums[right]);
            right++;
        }
    }
```

TwoPointer ⇒ O(k)

```java
    while(left >= 0 && res.size() < k){
        res.add(nums[left]);
        left--;
    }

    while(right < nums.length && res.size() < k){
        res.add(nums[right]);
        right++;
    }

    Collections.sort(res); → O(klogk)
    return res;
}
```

Square Root → Integral LC 69

Square Root → Fractional Codestudio

Valid Perfect Square LC 367

Nth Root GfG

Square Root

$N = 16$

$\sqrt{N} = 4$

$N = 25$

$\sqrt{N} = 5$

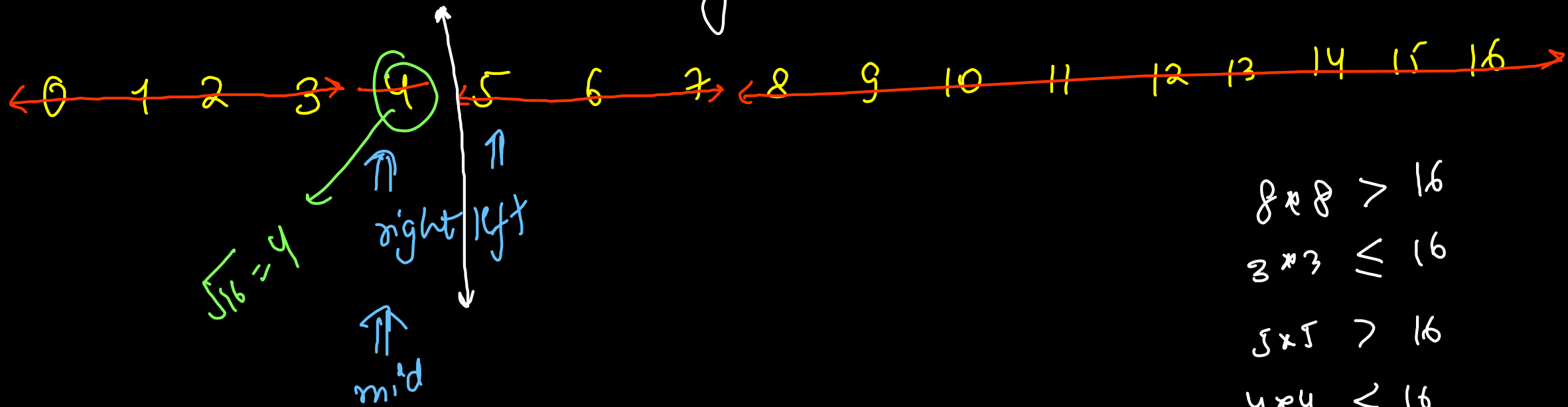
$N = 20$

$\sqrt{N} = \lfloor 4.4 \rfloor = 4$

$N = 30$

$\sqrt{N} = \lfloor 5.3 \rfloor = 5$

Binary Search $0 - N$

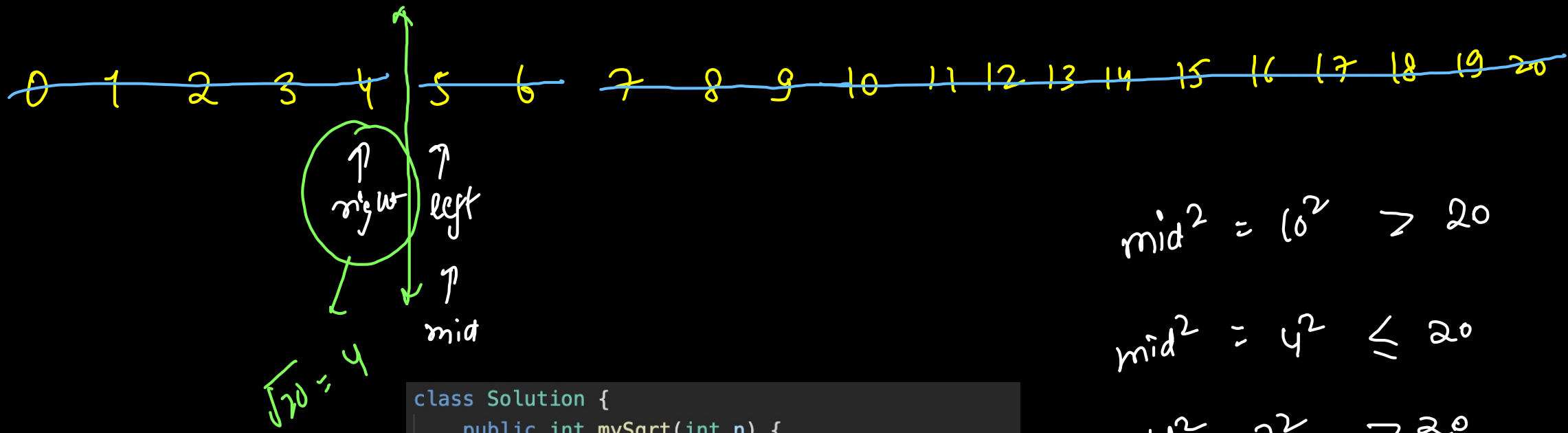


$8 * 8 > 16$

$3 * 3 \leq 16$

$5 \times 5 > 16$

$4 * 4 \leq 16$



$mid^2 = 10^2 > 20$

$mid^2 = 4^2 \leq 20$

$mid^2 = 7^2 > 20$

$5^2 > 20$

O(log n)

```java
class Solution {
    public int mySqrt(int n) {
        int left = 1, right = n;

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

            if(1l * mid * mid <= n){
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }

        return right;
    }
}
```

Square Root Fractional

$N = 16$ $D = 3$ $\Rightarrow$ 4.000 $\quad floor(\sqrt{N}) = 4.000$

$N = 20$ $D = 2$ $\Rightarrow$ 4.47 $\quad floor(\sqrt{N}) = 4.47$

$N = 30$ $D = 5$ $\Rightarrow$ 5.47722 $\quad floor(\sqrt{N}) = 5$

$N_{max} = 10^{15}$ $D_{max} = 6$

Floor → Square Root + Linear Search on Every Decimal

$O(\log_2 n) + O(10 * D)$

$i = 0$

$4 + 0 * 0.1$

$j = 0$ 4.0

$+ 0.1$

$j = 1$ 4.1 $4 + 1 * 0.1$ $10^{-1} = 1/10$

$+ 0.1$

$j = 2$ 4.2 $4 + 2 * 0.1 = 4 + 0.2 = 4.2$

$+ 0.1$

$j = 3$ 4.3 $4 + 3 * 0.1 = 4.3$

$j = 4$ 4.4

4.9

$i = 1$

4.00

$+ 0.01$

4.01 $10^{-2} = 1/100$

$+ 0.01$

4.02

$+ 0.01$

4.09

$i = 2$

4.000

$+ 0.001$

4.001 $10^{-3} = 1/1000$

$+ 0.001$

4.002

4.009

```java
public static long mySqrt(long n) {
    long left = 1, right = n;

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

        if (mid * mid <= n) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return right;
}
```

O(logn)

```java
public static double squareRoot(long n, int d) {
    long floor = mySqrt(n);
    double ans = floor;
    double factor = 1.0;

    for(int i = 0; i < d; i++){
        factor = factor / 10;
        for(int j = 0; j < 10; j++){
            if((ans + factor) * (ans + factor) <= n)
                ans += factor;
            else break;
        }
    }

    return ans;
}
```

O(10*D)

n = 20

ans = 4 +0.1 +0.1 +0.1 = 4.4 +0.01 +0.01 +0.01 ... 7 times

d = 2

factor = 0.1

$(4.0 + 0.1) * (4.0 + 0.1) \le 20$

$(4.1 + 0.1) * (4.0 + 0.1) \le 20$

$4.3 * 4.3 \le 20$

$4.4^2 \le 20$

$4.5^2 \not\le 20$

factor = 0.01

$4.41^2 \le 20$

$4.42^2 \le 20$

$4.43^2 \le 20$

$4.47^2 \le 20$

left = 0.0 right = 20.0

mid = 10.0 $10.0^2 = 100 \not\leq 20$

left = 0.0 right = 10.0

mid = 5.0 $5.0^2 \not\leq 20$

left = 0.0 right = 5.0

mid = 2.5 $2.5^2 \leq 20$

left = 2.5 right = 5.0

mid = 3.75 $3.75^2 \leq 20$

l = 3.75 r = 5.0

m = 4.375 $4.375^2 \leq 20$

l = 4.375 r = 5.0

$m = 4.6875^2 > 20$

l = 4.375 r = 4.6875

m = 4.531

N = 20 D = 2

4.471 l ~ 4.476 r

$10^{-2} = 10^{-D}$

```
while (right - left >= 0.01) {
    double mid = l + (r-l)/2.0;
    if (mid^2 <= n) left = mid - 0.01;
    else right = mid - 0.01;
}
```

```java
public static double squareRoot(long n, int d) {
    double left = 1.0, right = n;
    double epsilon = Math.pow(10, -d);

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

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

    return left;
}
```

Approach 2

Valid Perfect Square

Leetcode 367

N = 16 ✓

$\sqrt{N} = 4$

floor = 4

$(4)^2 = 16$

N = 20 ✗

$\sqrt{N} = 4.47\ldots$

floor = 4

$4^2 \neq 20$

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

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

        if (1l * mid * mid <= n) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return right;
}

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

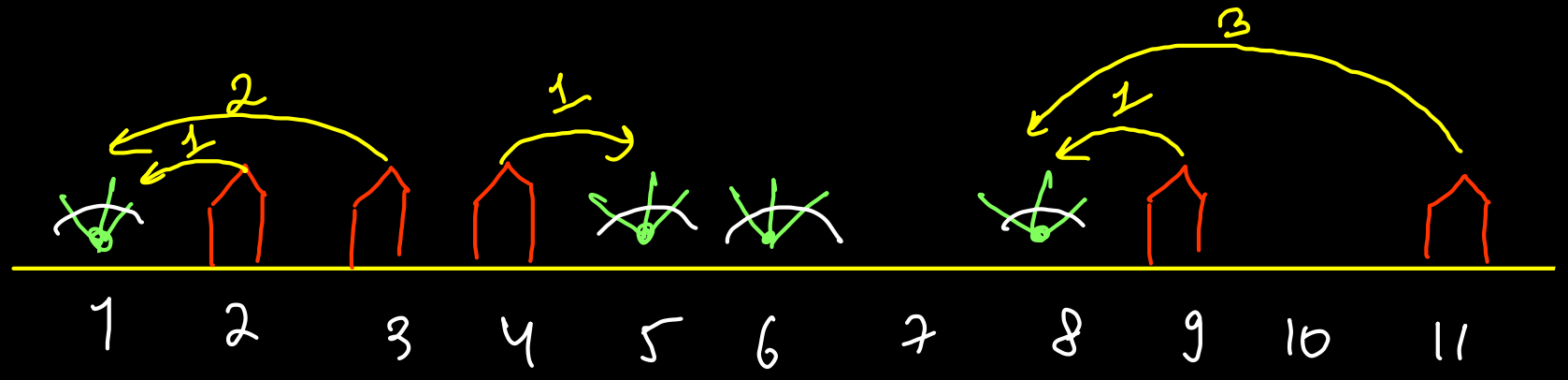
Heaters

Min radius = max of → Farthest house from any heater.
closest heater for each house

houses : 3, 9, 2, 4, 11

heaters : 1, 8, 5, 6 $\xrightarrow{sort}$ 1, 5, 6, 8

minimum radius = (3)



```java
public int findRadius(int[] houses, int[] heaters) {
    Arrays.sort(heaters);
    int minRadius = 0;

    for(int val: houses){
        int closestHeater = findClosest(heaters, val);
        int radius = Math.abs(closestHeater - val);
        minRadius = Math.max(radius, minRadius);
    }

    return minRadius;
}
```

n_1 n_2

$O(n_2 \log n_2)$

$O(n_1 * \log n_2)$

Total time

$\Rightarrow O((n_1 + n_2) * \log n)$

$\Rightarrow O(n \log n)$

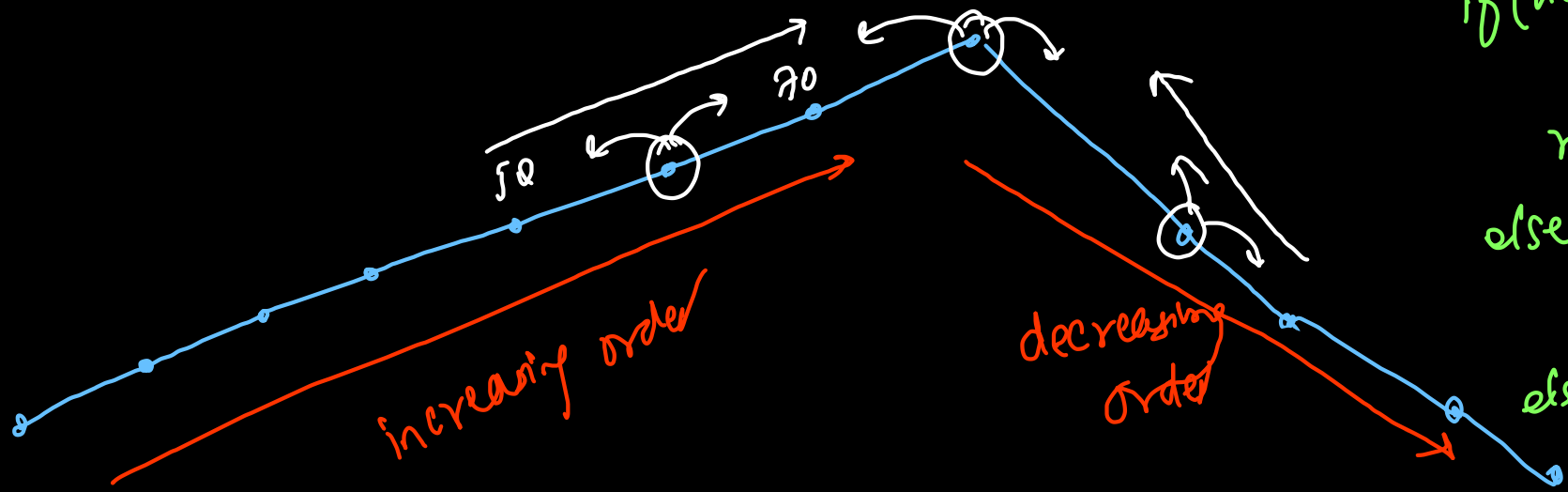
Mountain Array ⇒ Peak Element ⇒ Bitonic Array

leetcode 852 :

0	1	2	3	4	5	6	7	8	9	10	11
10	20	30	40	50	60	70	80	50	30	20	5

right left

mid

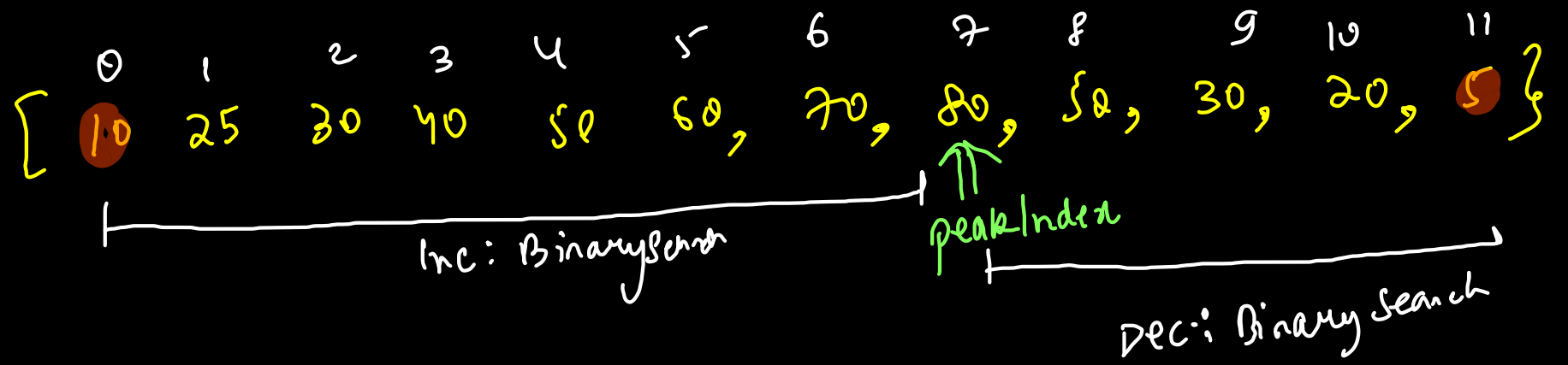



```
if (arr[m] > arr[m-1]
  && arr[m] > arr[m+1])
  return m;
else if (arr[m] < arr[m+1])
  left = m+1;
else right = m-1;
```

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

    return -1;
}
```

Time → O(logn)

Space → O(1)



target = 20 :→ 10^{th}

target = 40 :→ 3^{rd}

target = 50 :→ 7^{th}

target = 100, 15, 0 : −1

```java
public int binarySearchInc(int[] arr, int left, int right, int target){
    while(left <= right){
        int mid = left + (right - left) / 2;

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

public int binarySearchDec(int[] arr, int left, int right, int target){
    while(left <= right){
        int mid = left + (right - left) / 2;

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

public int solve(int[] arr, int target) {
    int peakIdx = peakIndexInMountainArray(arr);

    int lans = binarySearchInc(arr, 0, peakIdx, target);
    if(lans != -1) return lans; // Inc → search successful

    return binarySearchDec(arr, peakIdx + 1, arr.length - 1, target);
}
```

O(logn)

O(logn)

O(logn)

Time
↳ O(logn)

Space
↳ O(1)

// Dec → search successful

// Target not found
(-1)

Peak Element Leetcode 162

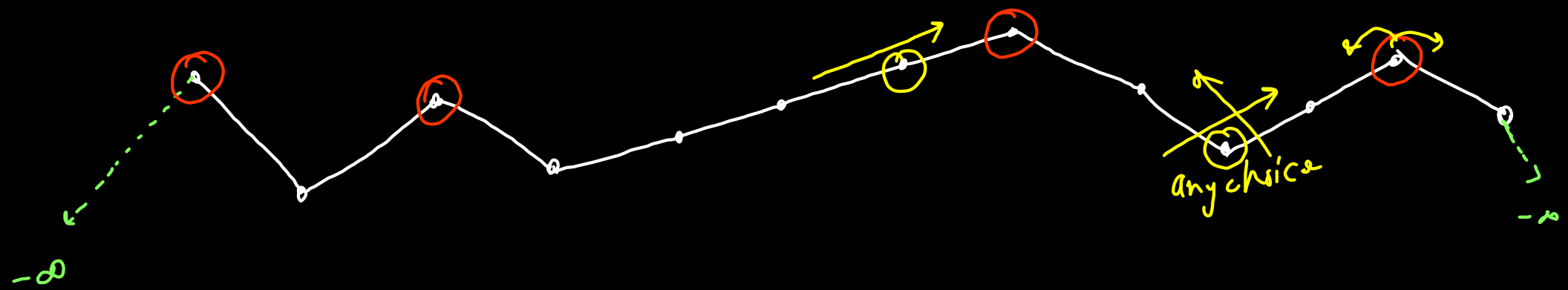
- array : unsorted
- multiple peaks → any peak
- nums[-1] = nums[n] = $-\infty$

~~global peak~~

local peak

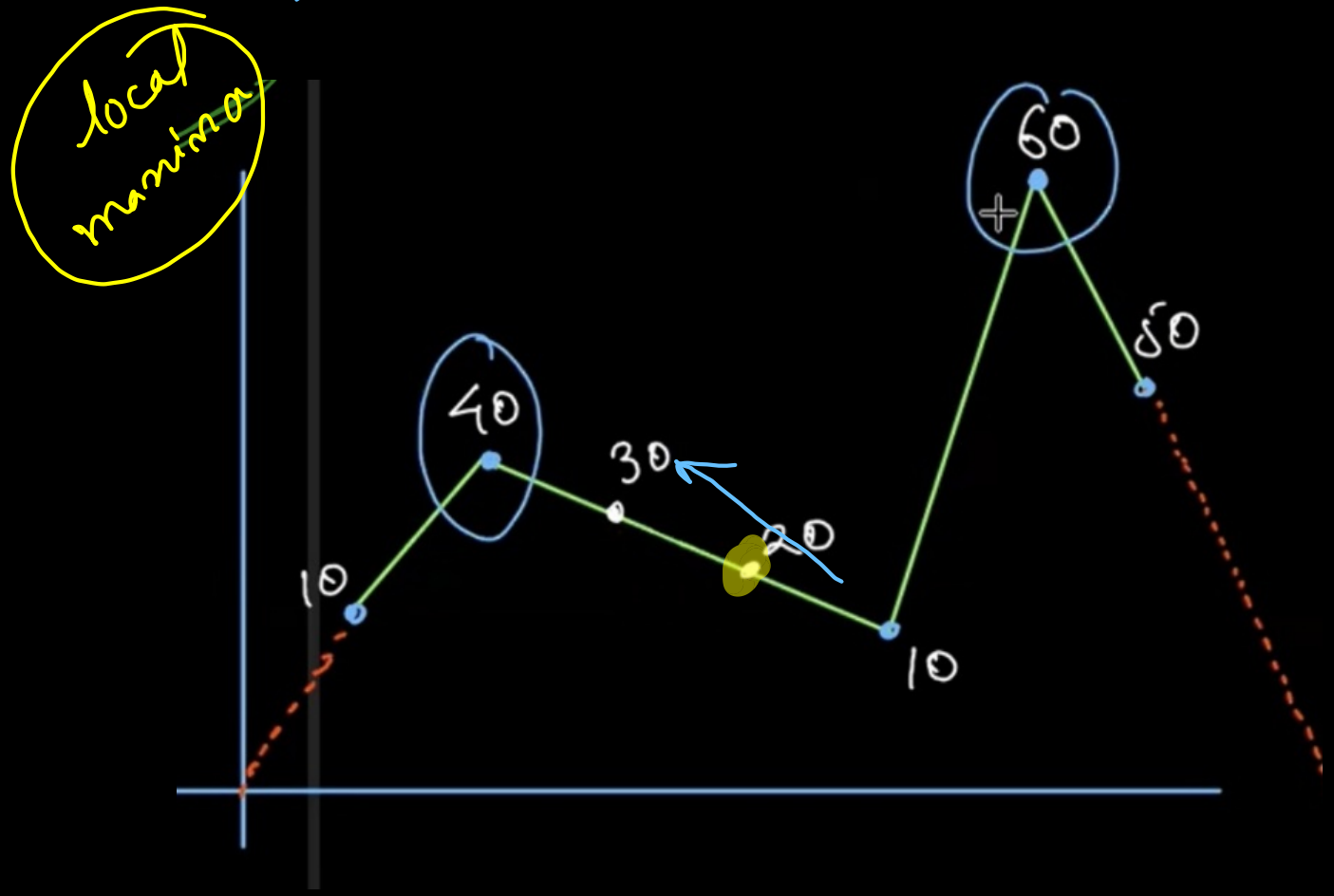
Example 1

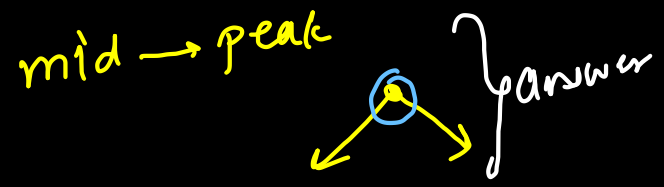
0	1	2	3	4	5	6	7	8	9	10	11	12
20	10	40	30	50	70	90	100	20	10	30	50	20
										↑ left	↑ mid	↑ right

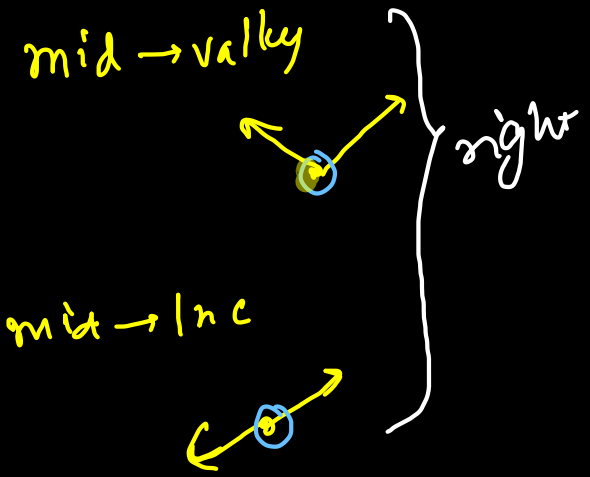


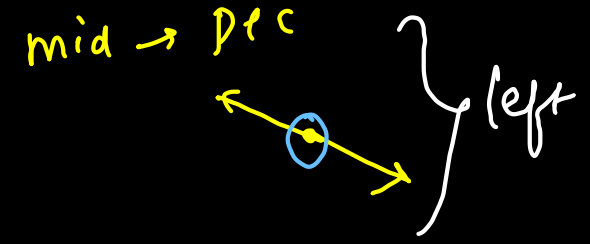
Example:

0	1	2	3	4	5	6
10,	40,	30,	20,	10,	60,	50
↑	↑				↑	
l	m				r	

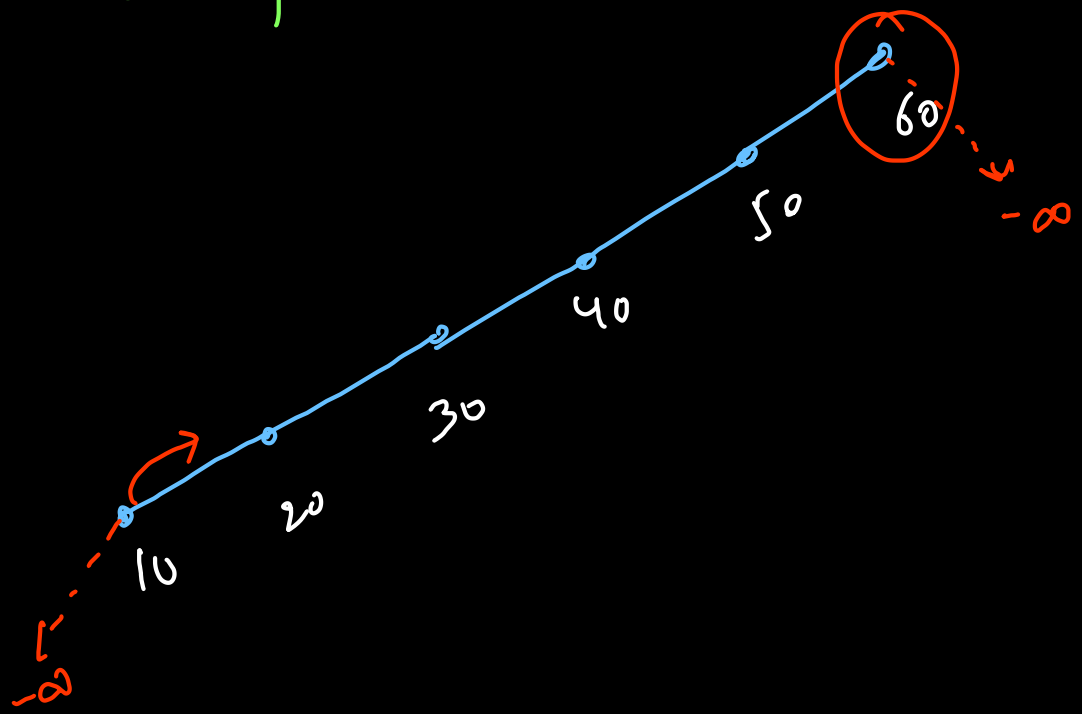




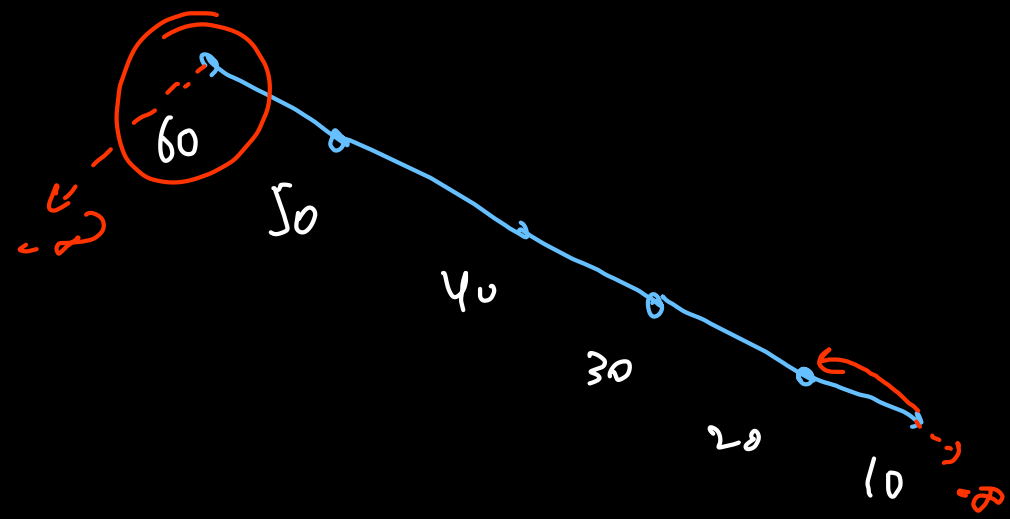




Example 3; Corner case (Inc)



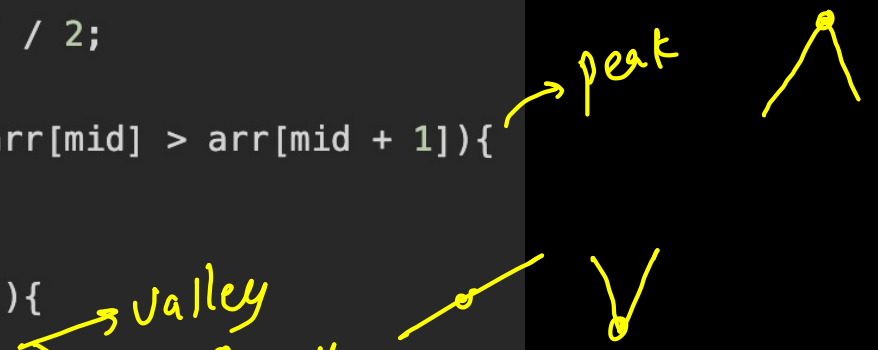
Example 4; Dec

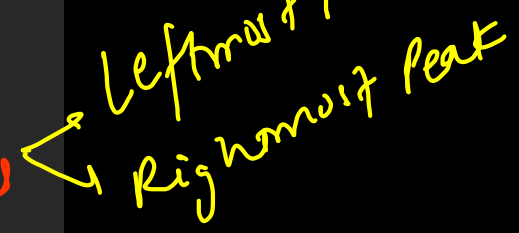


```java
public int peakIndexInMountainArray(int[] arr) {
    int left = 1, right = arr.length - 2;

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

        if(arr[mid] > arr[mid - 1] && arr[mid] > arr[mid + 1]){
            return mid;
        }
        else if(arr[mid] < arr[mid + 1]){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }

    return -1;
}

public int findPeakElement(int[] arr) {
    int n = arr.length;
    if(n == 1 || arr[0] > arr[1]) return 0;
    if(arr[n - 1] > arr[n - 2]) return n - 1;
    return peakIndexInMountainArray(arr);
}
```

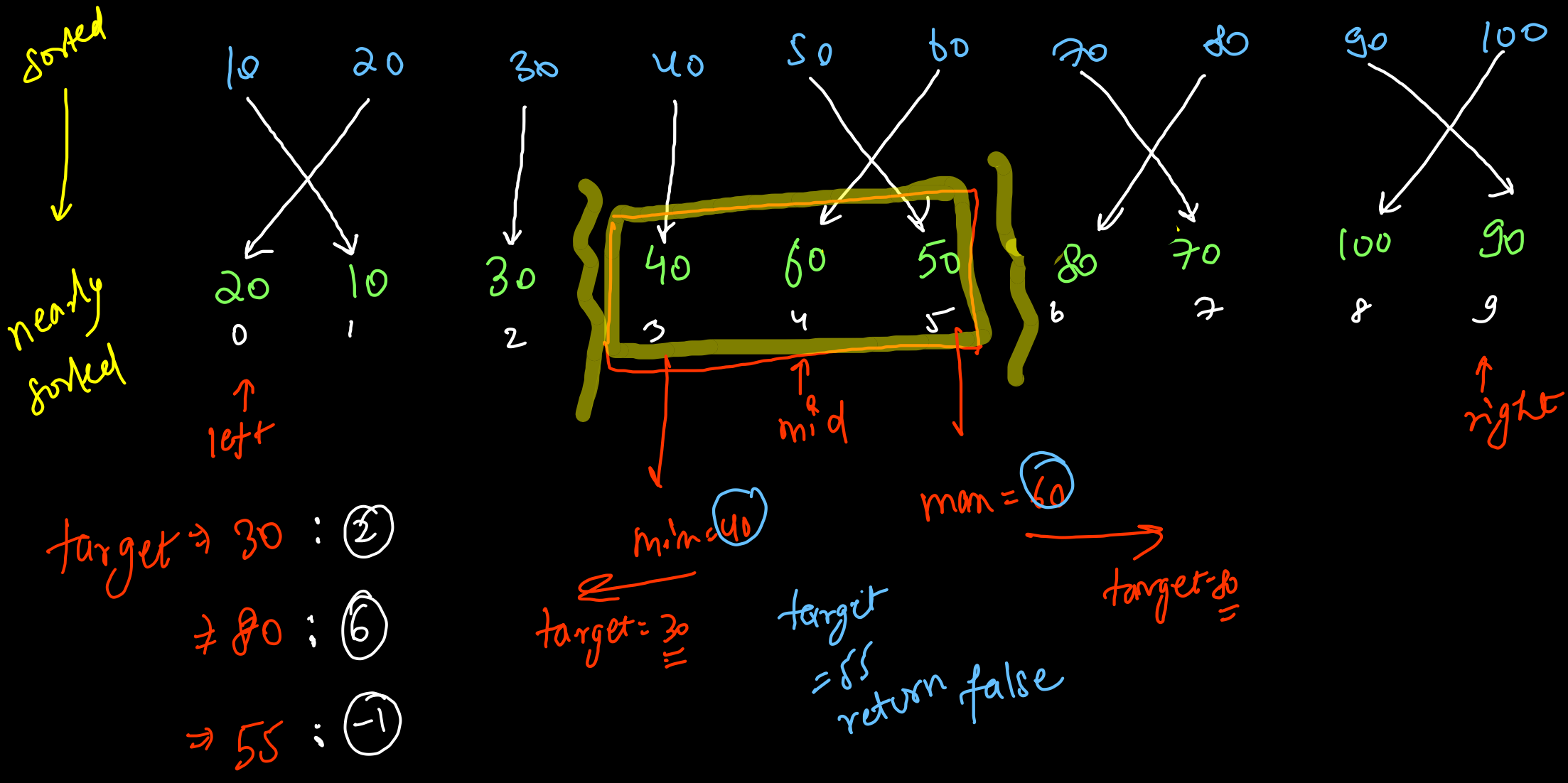
Time
O(logn)

Space
O(1)





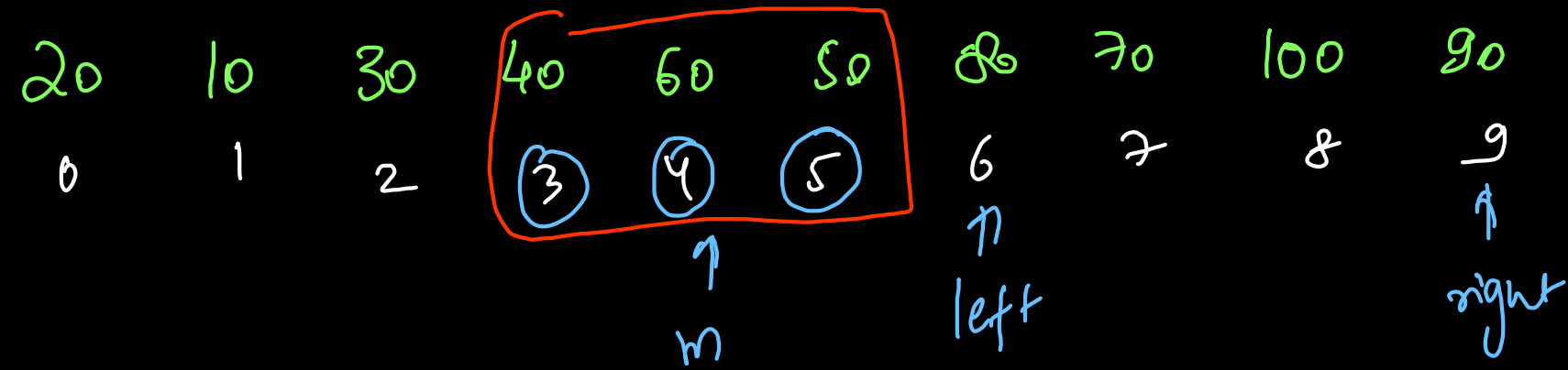
Search in Nearly Sorted Array



target ⇒ 30 : (2)

⇒ 80 : (6)

⇒ 55 : (-1)



target = 30

80

55

① arr(m), arr(m-1), arr(m+1)
return true;

② target < min
right = mid-2.

③ target > max
left = mid-2;

④ return false

```java
public static int solve(int[] arr, int target) {
    int left = 0, right = arr.length - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;

        int lval = (mid == 0) ? Integer.MIN_VALUE : arr[mid - 1];
        int rval = (mid == arr.length - 1) ? Integer.MAX_VALUE : arr[mid + 1];

        if (arr[mid] == target)
            return mid;
        if (lval == target)
            return mid - 1;
        if (rval == target)
            return mid + 1;

        int min = Math.min(arr[mid], Math.min(rval, lval));
        int max = Math.max(arr[mid], Math.max(rval, lval));

        if (target < min)
            right = mid - 2;
        else if (target > max)
            left = mid + 2;
        else
            break;
    }
    return -1;
}
```

Time :→ $O(\log_3 n)$

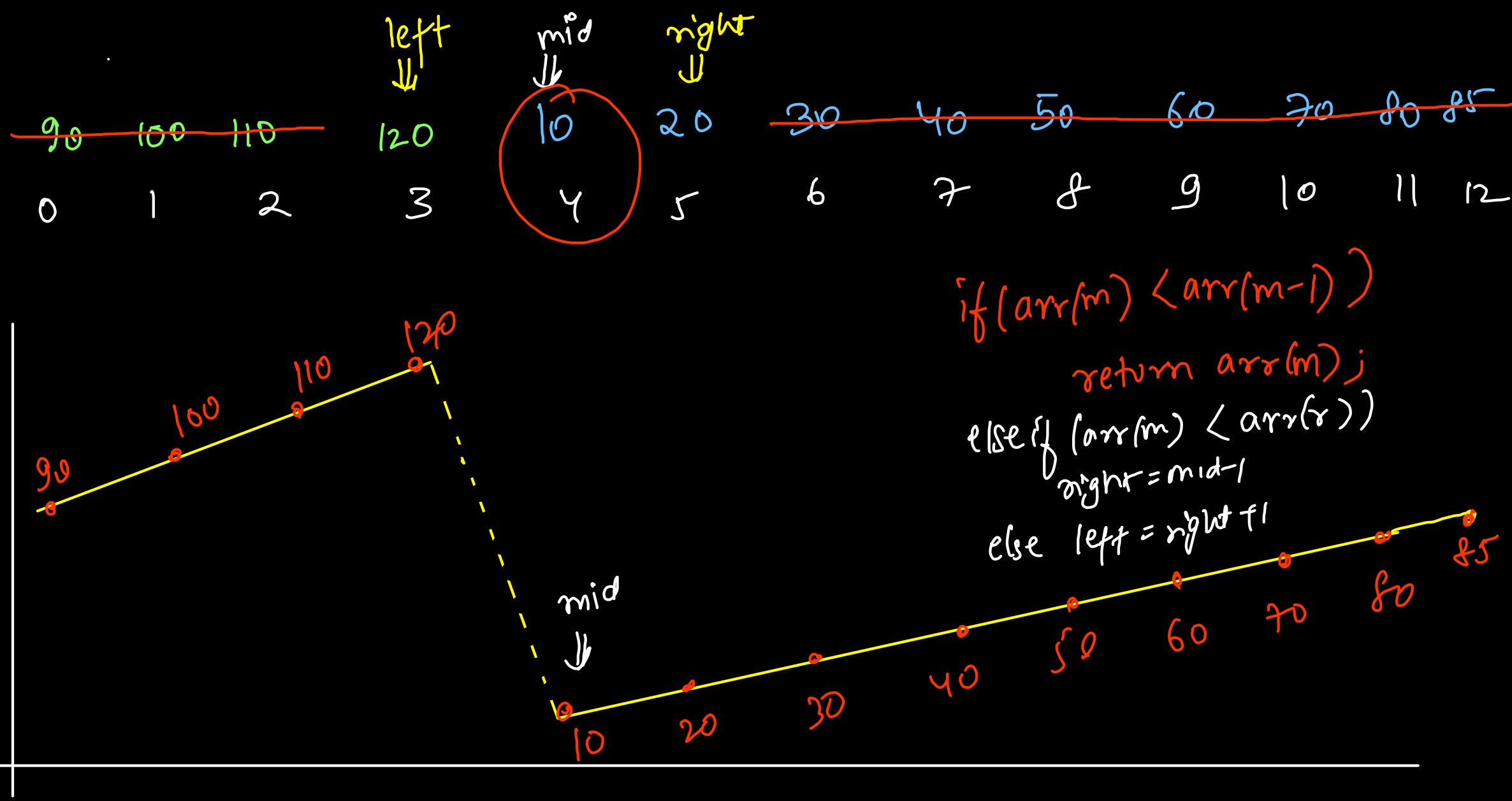
Space :→ $O(1)$

Rotated Sorted Array

1. Pivot Element / Minm Element - I Leetcode 153
2. Rotation Count - GFG
3. Search in rotated Sorted - II Leetcode 33

Rotated Sorted Array

	0	1	2	3	4	5	6	7	8
Sorted / k=0	10	20	30	40	50	60	70	80	90
k=1	90	10	20	30	40	50	60	70	80
k=2	80	90	10	20	30	40	50	60	70
k=3	70	80	90	10	20	30	40	50	60
k=4	60	70	80	90	10	20	30	40	50
⋮									
k=8	20	30	40	50	60	70	80	90	10


left
mid
right
90 100 110 120 10 20 30 40 50 60 70 80 85
0 1 2 3 4 5 6 7 8 9 10 11 12
if(arr(m) < arr(m-1))
return arr(m);
else if (arr(m) < arr(r))
right = mid-1
else left = right+1
90 100 110 120
mid
10 20 30 40 50 60 70 80 85


```java
public int findMin(int[] nums) {
    int n = nums.length;
    if(n == 1 || nums[0] < nums[n - 1]) return nums[0];
    // Array not rotated at all: completely sorted

    int left = 1, right = n - 1;
    while(left <= right){
        int mid = left + (right - left) / 2;
        if(nums[mid - 1] > nums[mid]) return nums[mid];
        if(nums[mid] <= nums[right])
            right = mid - 1; // unrotated part: min in left side
        else
            left = mid + 1; // rotated part: min in right side
    }

    return -1;
}
```

Time
↳ O(log n)

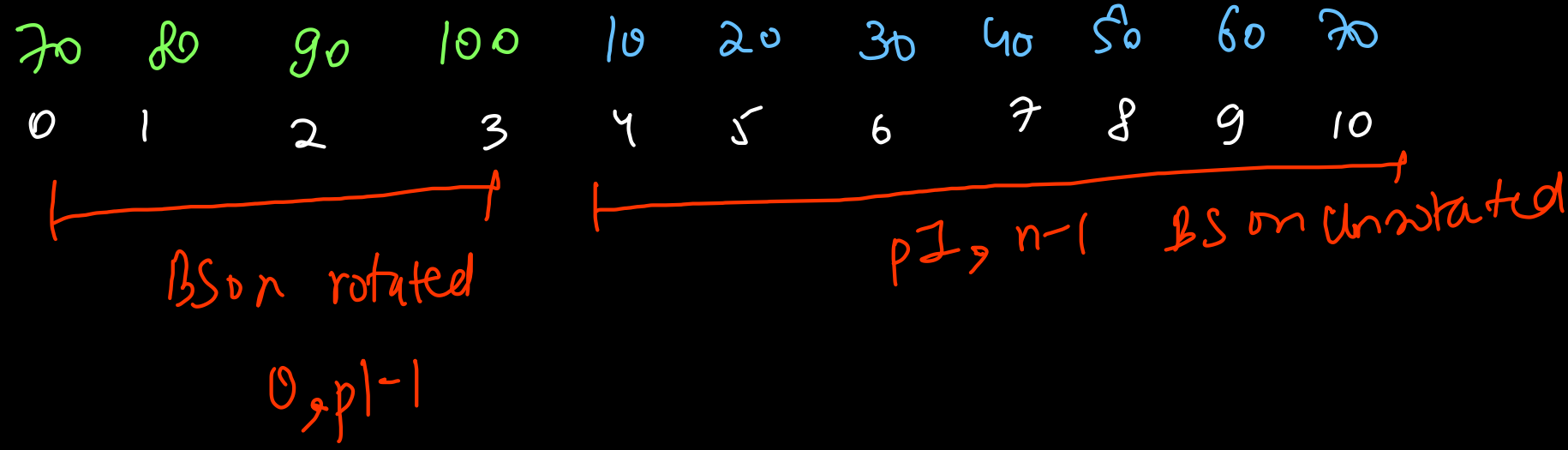
Space
↳ O(1)

(l-r) avg

Rotation Count Pivot / min^m Index = no of rotations

```
int findKRotation(int nums[], int n) {
    if(n == 1 || nums[0] < nums[n - 1]) return 0;

    int left = 1, right = n - 1;

    while(left <= right){
        int mid = left + (right - left) / 2;
        if(nums[mid - 1] > nums[mid]) return mid;
        if(nums[mid] <= nums[right])
            right = mid - 1; // unrotated part: min in left side
        else
            left = mid + 1; // rotated part: min in right side
    }

    return -1;
}
```

Search in Rotated Sorted Array



target: 60

Approach! →

1. Find Pivot Index
2. BS on rotated
3. BS on Unrotated!

```java
public int binarySearch(int[] nums, int left, int right, int target){
    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;
}

public int search(int[] nums, int target) {
    int pi = findMin(nums);
    int lans = binarySearch(nums, 0, pi - 1, target);
    if(lans != -1) return lans;

    return binarySearch(nums, pi, nums.length - 1, target);
}
```

Time ↘

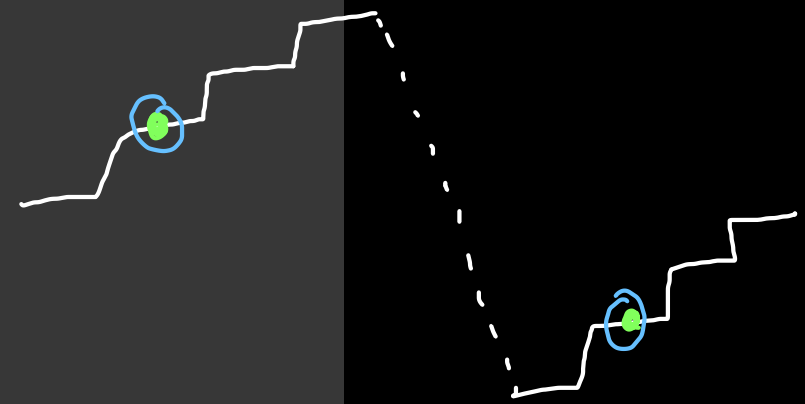
$O(2\log n)$

= logarithmic

Search Minm in Rotated Sorted Array with Duplicates

```java
class Solution {
// Time Complexity will be O(log n) in average case, but O(n) in worst case
// Worst case: All elements equal: We have to linearly reduce the search space
    public int findMin(int[] nums) {
        int low = 0, high = nums.length - 1;

        while(low < high){
            int mid = low + (high - low) / 2;

            if(nums[mid] < nums[high]){
                high = mid;
            } else if(nums[mid] > nums[high]){
                low = mid + 1;
            } else {
                high--;
            }
        }

        return nums[low];
    }
}
```

} plain region ⇒ linear search



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

unsorted

Remove 2 min ⇒ Max

0	1	2	3	4	5	6	7	8
4	5	9	10	11	12	15	100	200

Sorted :→

① min=4 ⇒ max=8 ⇒ delete 9-200 : 6 deletions

② delete 4 ⇒ min=5 ⇒ max=10 ⇒ delete 11-200 ⇒ 6 deletions

③ delete 4,5 ⇒ min=9 ⇒ max=18 ⇒ delete 100-200 ⇒ 4 deletions

leftDeletion ⇒ idx

rightDelem ⇒ n-upperbound(2x arr[idx])

Given an unsorted array **Arr** of size **N**. Find the minimum number of removals required such that twice of minimum element in the array is greater than or equal to the maximum in the array.

Dynamic Sliding window + Binary Search!

```java
class Solution {
    public static int upperBound(int[] nums, int target) {...}

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

        int minRemoval = n;

        for(int idx = 0; idx < n; idx++){
            int leftDel = idx;
            int rightDel = n - upperBound(arr, 2 * arr[idx]);

            minRemoval = Math.min(minRemoval, leftDel + rightDel);
        }

        return minRemoval;
    }
}
```

Time ⇒ nlogn

Space ⇒ O(1)

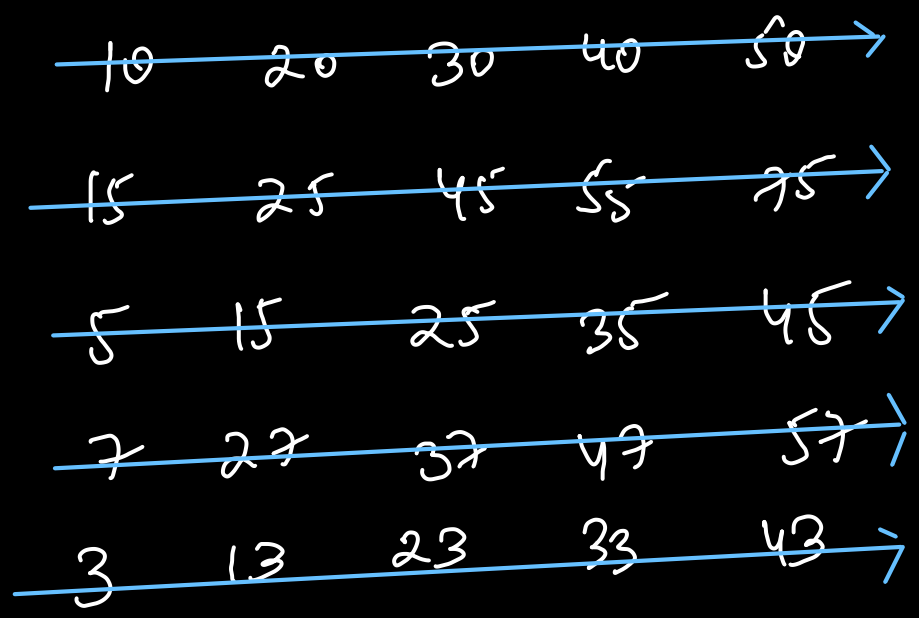
Searching in 2D Matrix

(1) unsorted : Binary Search not applicable ∠

Linear Search :→ $O(r * c)$

(2) only rows sorted but cols are not sorted

r rows, c cols



target = 23

Worstcase :→ Binary Search on each row

↳ $O(r \log c)$

$= O(n \log n)$

```java
class Solution {
    public boolean binarySearchRow(int[][] mat, int row, int target){
        int left = 0, right = mat[0].length - 1;
        while(left <= right){
            int mid = left + (right - left) / 2;

            if(mat[row][mid] == target) return true;
            else if(mat[row][mid] < target) left = mid + 1;
            else right = mid - 1;
        }

        return false;
    }

    public boolean searchMatrix(int[][] mat, int target) {
        // Brute Force: Binary Search On Each Row
        for(int row = 0; row < mat.length; row++){
            if(binarySearchRow(mat, row, target) == true){
                return true;
            }
        }
        return false;
    }
}
```

Time → $n \log m$

Space → $O(1)$

(3) rows (3.1) sorted, cols (3.2) sorted, (3.3) Each row's last ele ≤ Next row's first ele

Search Matrix - I
leetcode 74

Assume BS on 1D array

- Integers in each row are sorted from left to right.
- The first integer of each row is greater than the last integer of the previous row.

0	1	2	3	4
10 (0,0)	20 (0,1)	30 (0,2)	40 (0,3)	50 (0,4)
5	6	7	8	9
60 (1,0)	70 (1,1)	80 (1,2)	90 (1,3)	100 (1,4)
10	11	12	13	14
105 (2,0)	110 (2,1)	120 (2,2)	140 (2,3)	150 (2,4)
15	16	17	18	19
160 (3,0)	170 (3,1)	190 (3,2) mid	195 (3,3)	200 (3,4)

l, r, mid

target = 190

idx → row: idx / cols

idx → col: idx % cols

```java
class Solution {
    // Optimized Approach: Binary Search on 1D Array
    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;
    }
}
```

$O(\log(\text{rows} * \text{cols}))$

$= O(\log(n^2))$

$= O(2 \log n)$

$\approx O(\log n)$

④ Rows Sorted, Cols Sorted

Each row may not be smaller than next row

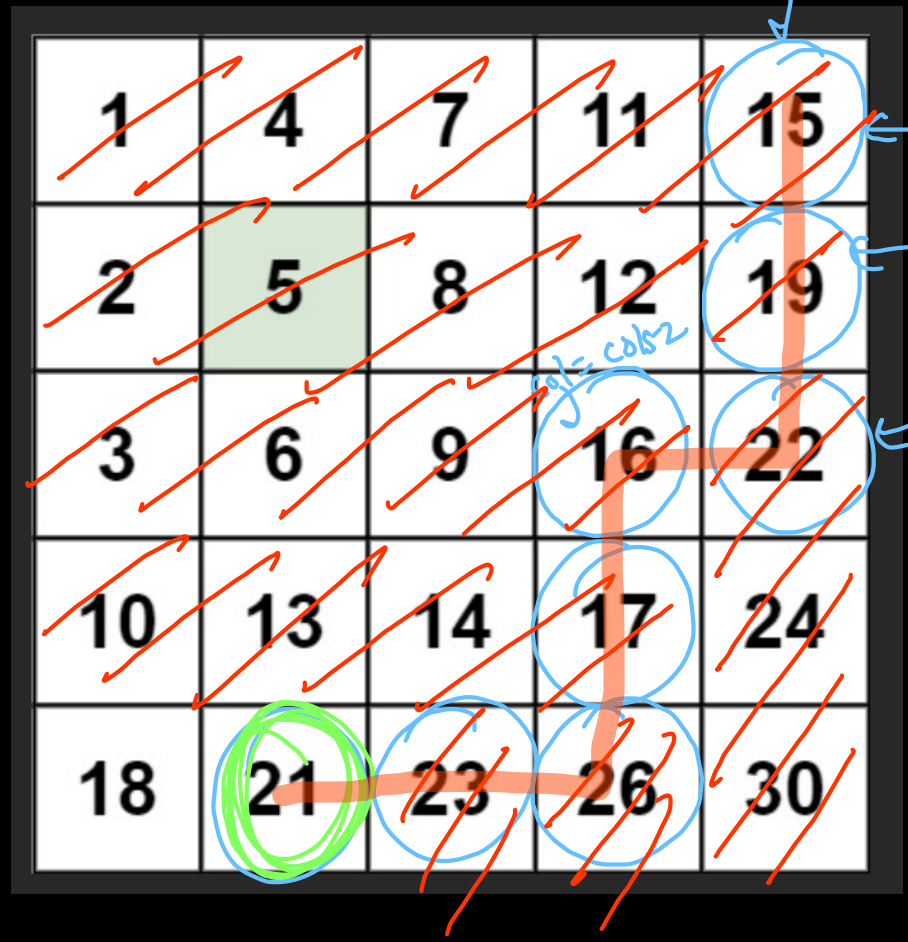
Brute force : O(r log c): Binary search on each rows

↓

Binary search on 1D array not possible

↓

O(log rc)



target = 27

15 < 21
discard row : rowtt

17 < 21
discard row

19 < 21
discard row : rowtt

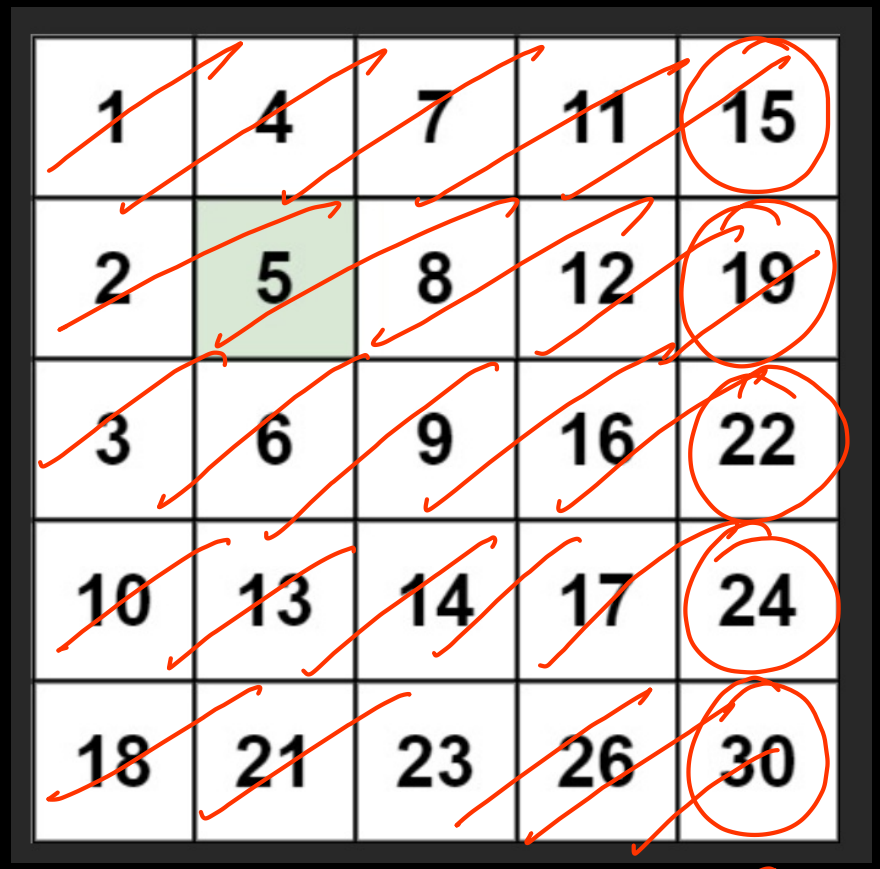
26 > 21
discard col

22 > 21
discard col : col--

23 > 21
discard col

16 < 21
discard row

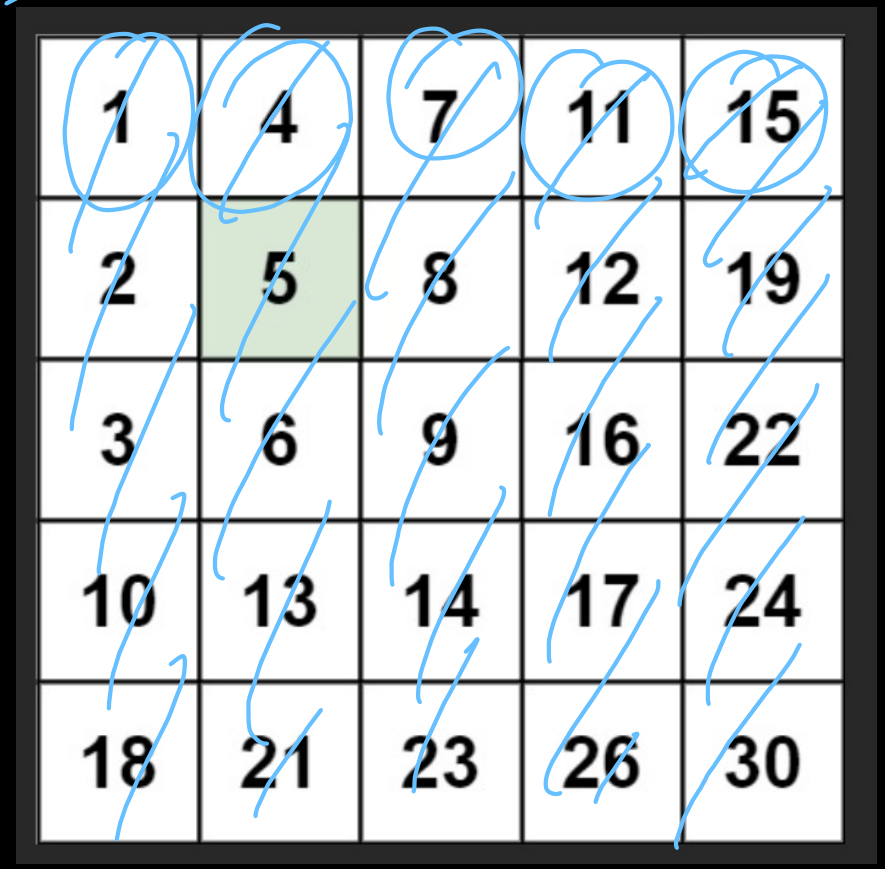
You can only start the stepcase/Staircase search from either top-right corner or bottom-left corner



target = 35

row = rows

cd == -1



target = -5

Search Un-successful

Staircase Search

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

    while(row < rows && col >= 0){
        if(mat[row][col] == target) return true;
        if(mat[row][col] < target)
            row++; // discard the topmost row
        else col--; // discard the rightmost column
    }

    return false;
}
```

Time

↳ O(n)

O(rows + cols)

↑ down + ↑ left

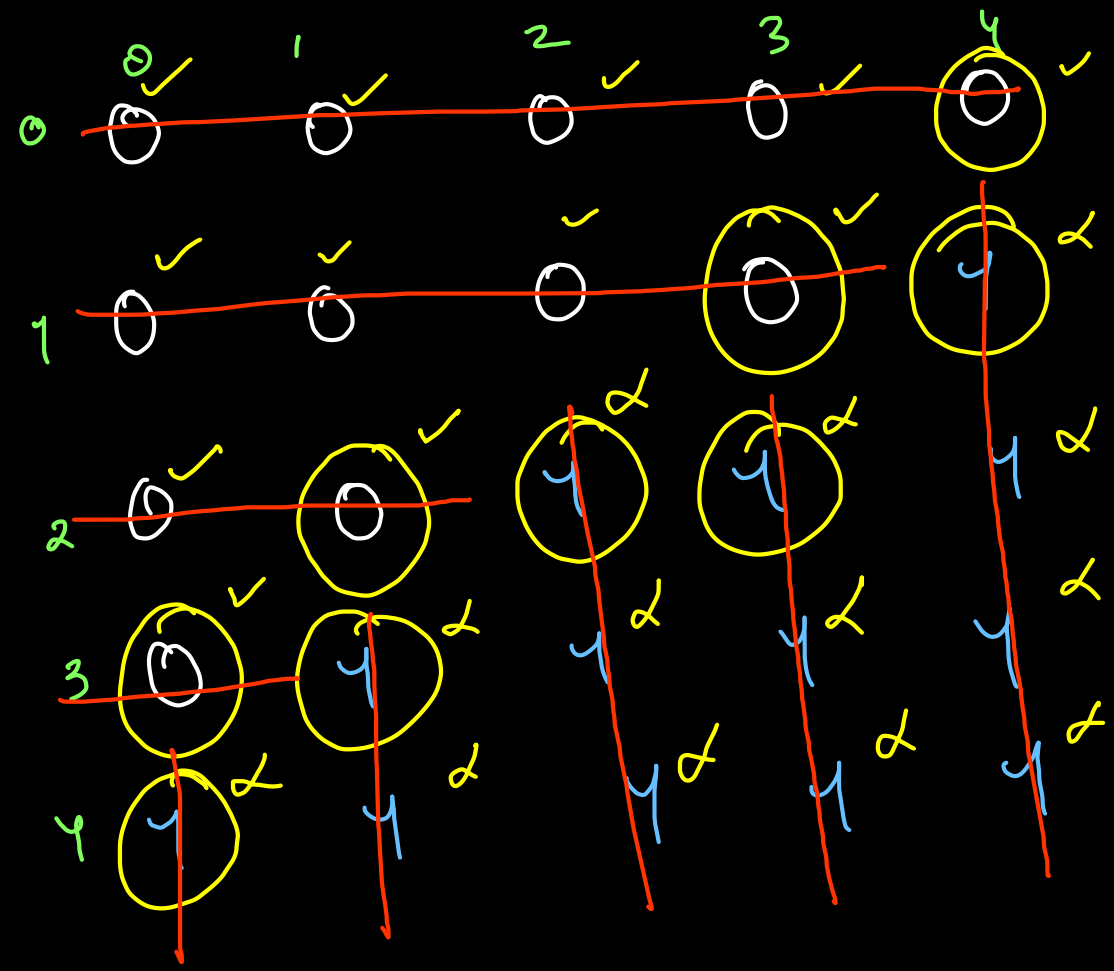
= O(n + n) = O(2n)

Binary matrix → 0, 1

⇒ each row is sorted

⇒ each col is sorted

Staircase search



count zeros

= 0 + 5 + 4 + 2 + 1

= (12)

```
if ( mat[r][c] == 0)
    zeros += (c+1);
    row++;

else col--;
```

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

    while(row < N && col >= 0){
        if(mat[row][col] == 0){
            zeros = zeros + (col + 1); // all zeros in current row
            row++; // no more zeros: discard row
        } else {
            col--; // no zero in current column: discard column
        }
    }

    return zeros;
}
```

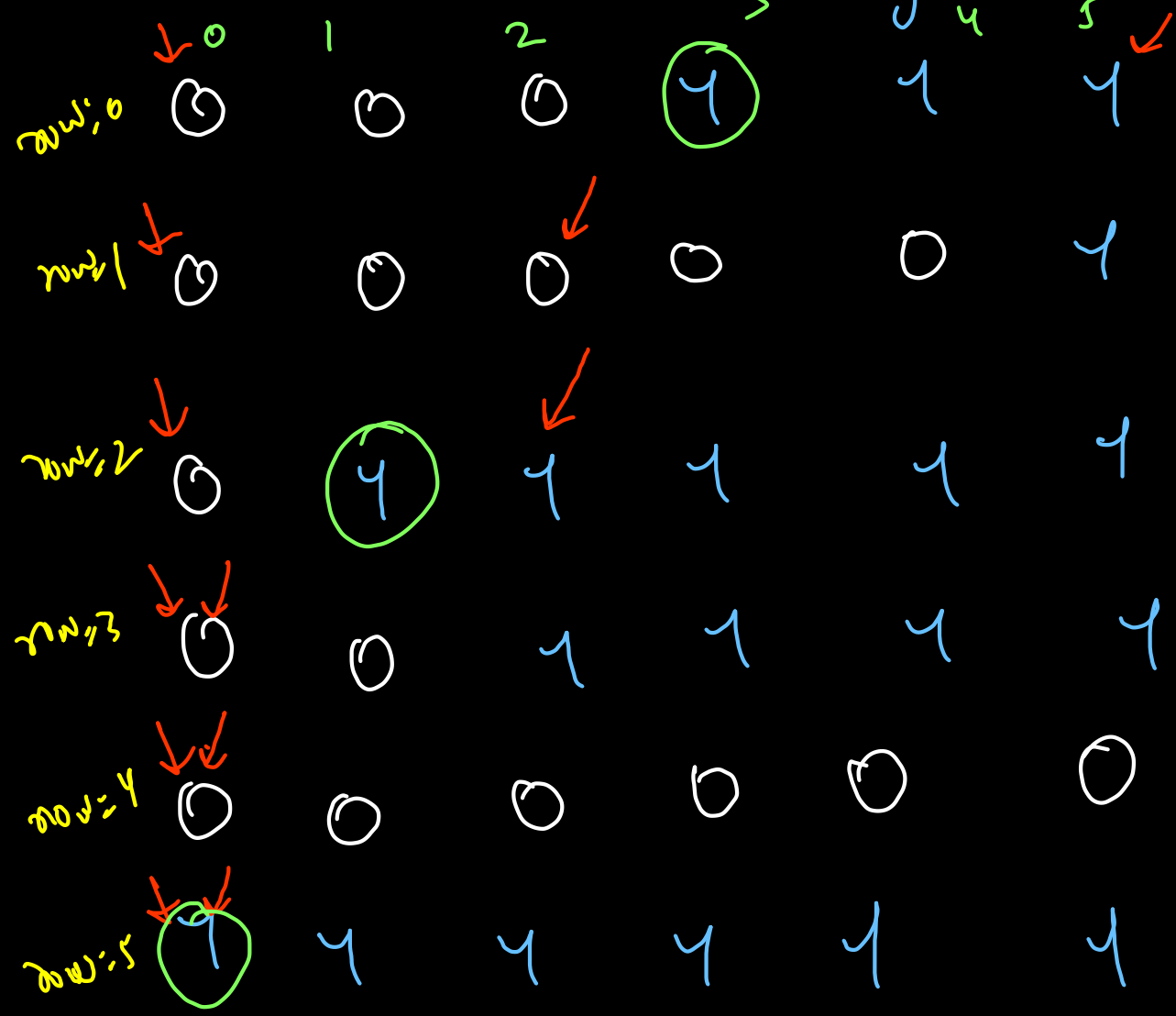
Time ⇒ O(n) → Staircase Search

Space ⇒ O(1)

Binary matrix

Row with max 1s

Only rows are sorted → cols are not sorted



Binary search on each row with reduced column

ones = 4 3 2 1 0

row = 0th 2nd 5th

```
int binarySearch(int arr[][], int row, int left, int right){
    while(left <= right){
        int mid = left + (right - left) / 2;

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

int rowWithMax1s(int arr[][], int rows, int cols) {
    int right = cols - 1, ans = -1;

    for(int row = 0; row < rows; row++){
        int mid = binarySearch(arr, row, 0, right);
        if(mid <= right)  {
            ans = row;
            right = mid - 1; // reduced binary search on next row
        }
    }

    return ans;
}
```

first occ of 1

Corner case

If entire matrix is 0, return -1;

Worst case

O(nlogn)

Binary search on each row!

Binary Search on Answer

Hard concept

Very very Important!

Minimizatn / maximization
- Greedy
- DP
- Divide & conquer

- Koko Eating Bananas
- Book Allocation Problem
- Aggressive Cows
- Wood cutting / EKO

Koko Eating Bananas

Leetcode 875

Koko loves to eat bananas. There are `n` piles of bananas, the `i^th` pile has `piles[i]` bananas. The guards have gone and will come back in `h` hours.

Koko can decide her bananas-per-hour eating speed of `k`. Each hour, she chooses some pile of bananas and eats `k` bananas from that pile. If the pile has less than `k` bananas, she eats all of them instead and will not eat any more bananas during this hour.

Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

Return *the minimum integer* `k` *such that she can eat all the bananas within* `h` *hours.*

7, 3, 44, 6

hours h = 8

banana speed = k/hour

min speed = ?

linear search

max hours h = 8

7, 3, 11, 6

- piles.length <= h <= 10^9

Ans always valid

Speed = 1 time: $\frac{7}{1} + \frac{3}{1} + \frac{11}{1} + \frac{6}{1}$ = 27 hours False

Speed = 2 time: $\lceil \frac{7}{2} \rceil + \lceil \frac{3}{2} \rceil + \lceil \frac{11}{2} \rceil + \lceil \frac{6}{2} \rceil$ = 4+2+6+3 = 15 hours False

Speed = 3 time: $\lceil \frac{7}{3} \rceil + \lceil \frac{3}{3} \rceil + \lceil \frac{11}{3} \rceil + \lceil \frac{6}{3} \rceil$ = 3+1+4+2 = 10 hours False

Speed = 4 time: $\lceil \frac{7}{4} \rceil + \lceil \frac{3}{4} \rceil + \lceil \frac{11}{3} \rceil + \lceil \frac{6}{4} \rceil$ = 2+1+3+2 = 8 hours True

Speed = 5 time: $\lceil 7/5 \rceil + \lceil 3/5 \rceil + \lceil 11/5 \rceil + \lceil 6/5 \rceil$ = 2+1+3+2 = 8 hours True

Speed = 6 t: $\lceil 7/6 \rceil + \lceil 3/6 \rceil + \lceil 11/6 \rceil + \lceil 6/6 \rceil$ = 2+1+2+1 = 6 hours True

maxSpeed = max pile {11}

O(max * n)

linear search

```java
class Solution {
    public boolean isPossible(int[] piles, long speed, long totalHours){
        long hoursRequired = 0;

        for(int bananas: piles){
            hoursRequired += bananas / speed; // floor division
            if(bananas % speed != 0) hoursRequired++; // ceil division
        }

        return (hoursRequired <= totalHours);
    }

    public long maxInArray(int[] piles){
        int max = 0;
        for(int bananas: piles)
            max = Math.max(max, bananas);
        return max;
    }

    public int minEatingSpeed(int[] piles, int totalHours) {
        long max = maxInArray(piles);
        for(long speed = 1l; speed <= max; speed++){
            if(isPossible(piles, speed, totalHours) == true)
                return (int)speed;
        }
        return (int)max;
    }
}
```

Linear Search

Search space ⇒ max

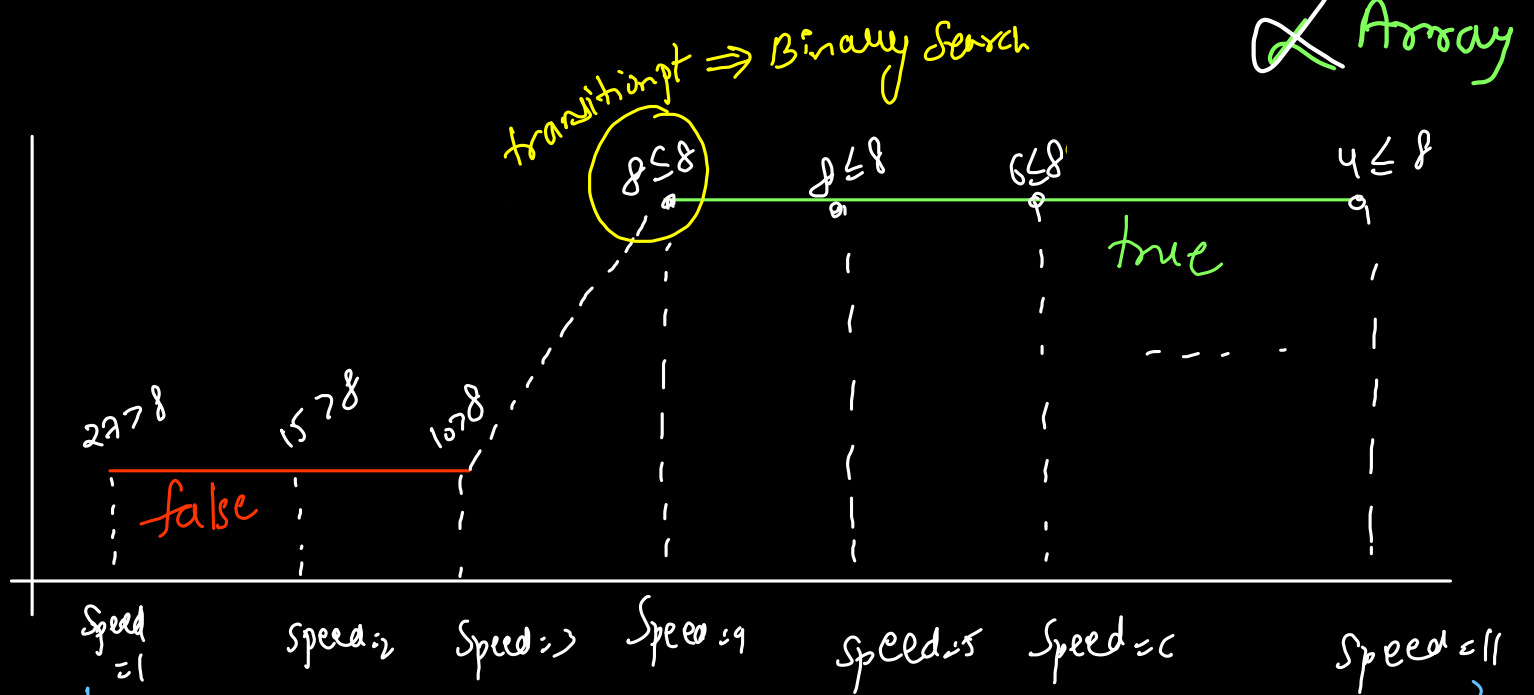
Ispossible ⇒ n

Total time ⇒ O(max * n)

$10^9 * 10^4$

$= 10^{13}$ TLE

Array { need not be Sorted }



left = 1 right = max = 11 mid = 6 isPossible : true

⇒ right = mid - 1

left = 1 right = 5 mid = 3 isPossible = false ⇒ left = mid + 1

left = 4 right = 5 mid = 4 isPossible : true ⇒ right = mid - 1

left = 4 ; right = 3

```java
class Solution {
    public boolean isPossible(int[] piles, long speed, long totalHours){
        long hoursRequired = 0;

        for(int bananas: piles){
            hoursRequired += bananas / speed; // floor division
            if(bananas % speed != 0) hoursRequired++; // ceil division
        }

        return (hoursRequired <= totalHours);
    }

    public long maxInArray(int[] piles){
        int max = 0;
        for(int bananas: piles)
            max = Math.max(max, bananas);
        return max;
    }

    public int minEatingSpeed(int[] piles, int totalHours) {
        long left = 1l;
        long right = maxInArray(piles);

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

        return (int)left;
    }
}
```

1. Array need not be sorted
2. BS on ~~Array~~ Answer
3. BS $\Rightarrow$ ~~O(log N)~~ $\Rightarrow$ O(N log N)
4. BS $\Rightarrow$ Transition pt

O(log(searchspace))
= O(log max)

O(n)

Total
= O(n log max)

10^4 $\log 10^9$
$= 9 * 3$
$= 27$

1283. Find the Smallest Divisor Given a Threshold

Hint

Medium 1.8K 163

Companies

Given an array of integers `nums` and an integer `threshold`, we will choose a positive integer `divisor`, divide all the array by it, and sum the division's result. Find the **smallest** `divisor` such that the result mentioned above is less than or equal to `threshold`.

Each result of the division is rounded to the nearest integer greater than or equal to that element. (For example: `7/3 = 3` and `10/2 = 5`).

The test cases are generated so that there will be an answer.

More Formal Definition!

Same code as Koko Eating Bananas

Book Allocation Problem

You are given **N** number of books. Every **ith** book has **Ai** number of pages.

You have to allocate contiguous books to **M** number of students. There can be many ways or permutations to do so. In each permutation, one of the M students will be allocated the maximum number of pages. Out of all these permutations, the task is to find that particular permutation in which the maximum number of pages allocated to a student is the minimum of those in all the other permutations and print this minimum value.

Each book will be allocated to exactly one student. Each student has to be allocated at least one book.

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

Corner case

{ 20, 10, 30, 40 }

books = 4 Stud = 2

⇒ Contiguous allocation

⇒ minimize the maxm no of book pages

⇒ Each book ⇒ one Student

→ Cannot skip any book

→ Cannot give same book to 2 studen

{ 20, 10, 30, 40 }

Stud = 2

Invalid

→ conf1: A: { 20 } B: { 10 } 30, 40 : not distributed

→ conf2: A: { 20, 10 } B: { 10, 30, 40 } (10) → same book to 2 students.

→ conf3: A: { 20, 40 } B: { 10, 30 } A: discontiguous allocation

→ conf4: A: { 20, 10, 30, 40 } B: { } although valid; but not useful

{ 20, 10, 30, 40 } Stud = 2

{ 20 } 20 pages { 10, 30, 40 } 80 pages : man load = 80 pages
S_1 S_2

{ 20, 10 } 30 pages { 30, 40 } 70 pages : man load = 70 pages
S_1 S_2

{ 20, 10, 30 } 60 pages { 40 } 40 pages : man load = 60 pages
S_1 S_2

↓

minimum max load

{ 20, 10, 30, 40 } Stud = 3

{ 20 } 20 load { 10 } 10 pages { 30, 40 } 70 pages : Max load 70

S_1 S_2 S_3

{ 20, 10 } 30 load { 30 } 30 load { 40 } 40 pages : 40

S_2 S_2 S_3

{ 20 } 20 pages { 10, 30 } 40 pages { 40 } 40 pages : 40

S_1 S_2 S_3

Minimize max load

Linear Search : maxm Load : minimize

minm possible answer

= maxm of Array

= 40

maxm possible answer

= Sum of Array

= 100

isPossible({20, 10, 30, 40}, 40, 2)

reqStud = 1 2 3

currentload = 0 + 20 + 10

0 + 30

0 + 40

false

maxload

maxStud

isPossible({20, 10, 30, 40}, 50, 2)

reqStud = 1 2 3

currentload = 0 + 20 + 10

0 + 30

0 + 40

false

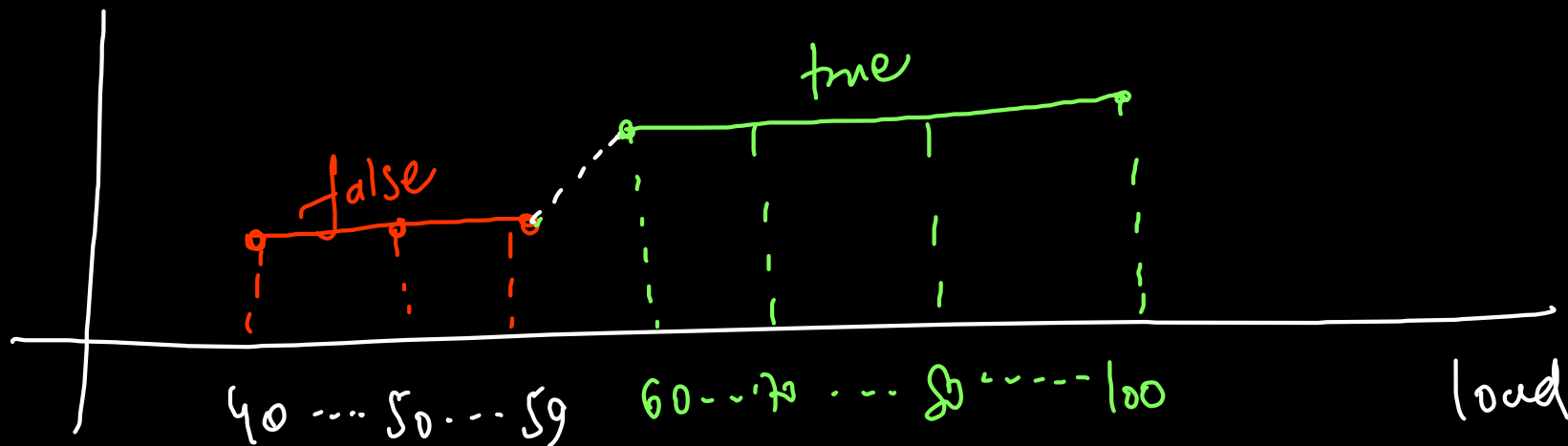
isPossible ({ 20, 10, 30, 40 }, 60, 2)

reqStud = ~~1~~ 2

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

0 + 40

} true



```java
// Can I Distribute all books to the totalStudents within maximum load
public static boolean isPossible(int[] pages, int maxLoad, int totalStud){
    int requiredStud = 1, currentLoad = 0;

    for(int book: pages){
        if(currentLoad + book <= maxLoad) {
            currentLoad += book;
        } else {
            requiredStud++;
            currentLoad = book;
        }
    }

    return (requiredStud <= totalStud);
}

public static int maxOfArray(int[] pages){
    int max = 0;
    for(int book: pages) max = Math.max(max, book);
    return max;
}

public static int sumOfArray(int[] pages){
    int sum = 0;
    for(int book: pages) sum += book;
    return sum;
}

public static int findPages(int[] pages, int books, int stud) {
    if(books < stud) return -1; // some students will not get any book

    int left = maxOfArray(pages); // Someone will have to read the max pages book
    int right = sumOfArray(pages); // Someone might have to read all the books

    for(int load = left; load <= right; load++){
        if(isPossible(pages, load, stud) == true)
            return load;
    }

    return -1;
}
```

no of books

sum of pages

TLE

O(sum of pages * no of books)

$10^6 * 10^5$

{20, 10, 50, 30, 40} stud = 3

left = max of Array = 50 ← O(sum of values)

right = sum of Array = 150

isPossible(50, 3)

reqstud = 1 2 3 4

load = 0+20+10 50 30 40 false

req given

4 7 3

isPossible(60, 3)

1 2 3 4

20+10 50 30 40

4 7 3

false

isPossible(70, 3)

reqstud = 1 2 3

load = 0+20+10 50 30+40

3 ≤ 3

true

```java
public static boolean isPossible(int[] pages, int maxLoad, int totalStud){
    int requiredStud = 1, currentLoad = 0;

    for(int book: pages){
        if(currentLoad + book <= maxLoad) {
            currentLoad += book;
        } else {
            requiredStud++;
            currentLoad = book;
        }
    }

    return (requiredStud <= totalStud);
}

public static int maxOfArray(int[] pages){ }
public static int sumOfArray(int[] pages){ }

public static int findPages(int[] pages, int books, int stud) {
    if(books < stud) return -1; // some students will not get any book

    int left = maxOfArray(pages); // Someone will have to read the max pages book
    int right = sumOfArray(pages); // Someone might have to read all the books

    while(left <= right){
        int mid = left + (right - left) / 2;
        if(isPossible(pages, mid, stud) == true)
            right = mid - 1;
        else left = mid + 1;
    }

    return left;
}
```

O(books)

important

O(log sum of array)

Total time

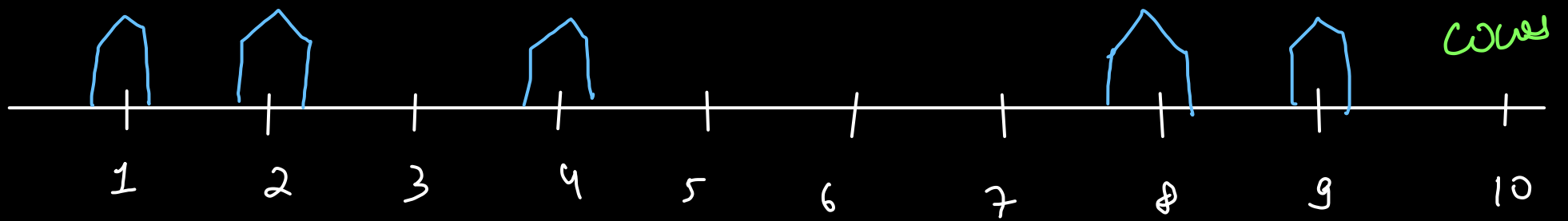
$O(\text{books} * \log \text{sum})$

10^5 $\log 10^6 = 20$

$\approx O(n \log n)$

Aggressive Cows

stall : {4, 1, 9, 2, 8} maximize the min^{m} dist.



c1 ↔ (1) c2 ↔ (2) c3 — minDist = 1

c1 ↔ (1) c2 ↔ (6) c3 — minDist = 1

c1 ↔ (1) c2 ↔ (7) c3 — minDist = 1

c1 ↔ (3) c2 ↔ (4) c3 — minDist = 3

c1 ↔ (3) c2 ↔ (5) c3 — minDist = 3 } answer

c1 ↔ (7) c2 ↔ (1) c3 — minDist = 1

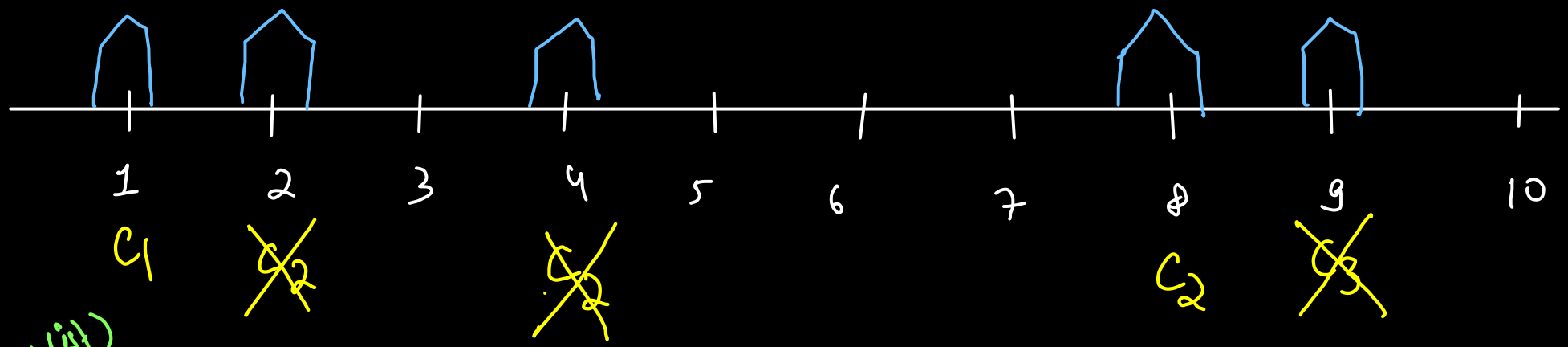
These will never give you optimal answer

c1 ↔ (2) c2 ↔ (4) c3 — minDist = 2

c1 ↔ (2) c2 ↔ (5) c3 — minDist = 2

c1 ↔ (4) c2 ↔ (1) c3 — minDist = 1

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



BSon Answer(dist)

left = 1

mid = (1+8)/2 = 4

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

isPossible:→ Can I place all the cows(3) such that min^m dist b/w any 2 cows ≥ 4

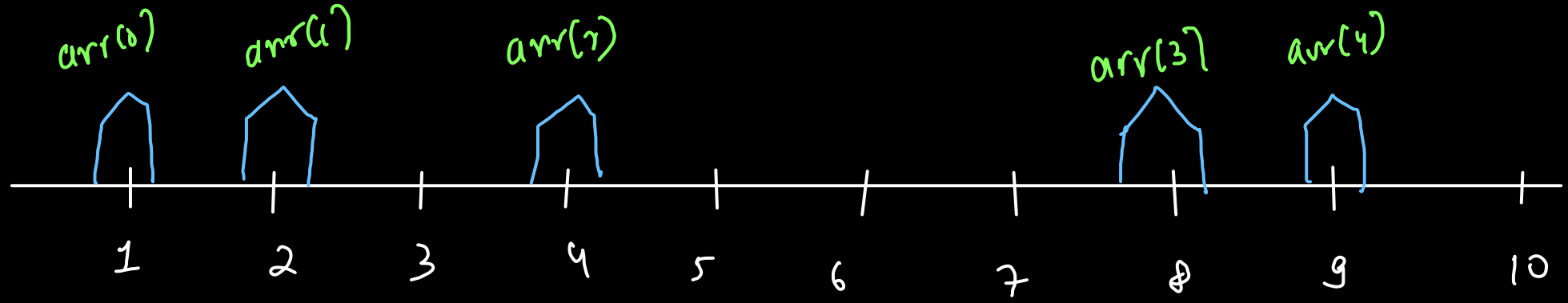
placedCows = ~~1~~ (2)

lastcow = ~~arr(0)~~ = 1

arr[3] = 8

2 < 3 all cows not placed

: false : right = mid-1



cows = (3)

C_1 C_2 C_3

left = 1

mid = (1+3)/2 = (2)

Can I place all cows such that min dist >= 2

right = mid-1 = 4-1 = 3

placedCows = 1, 2, (3)

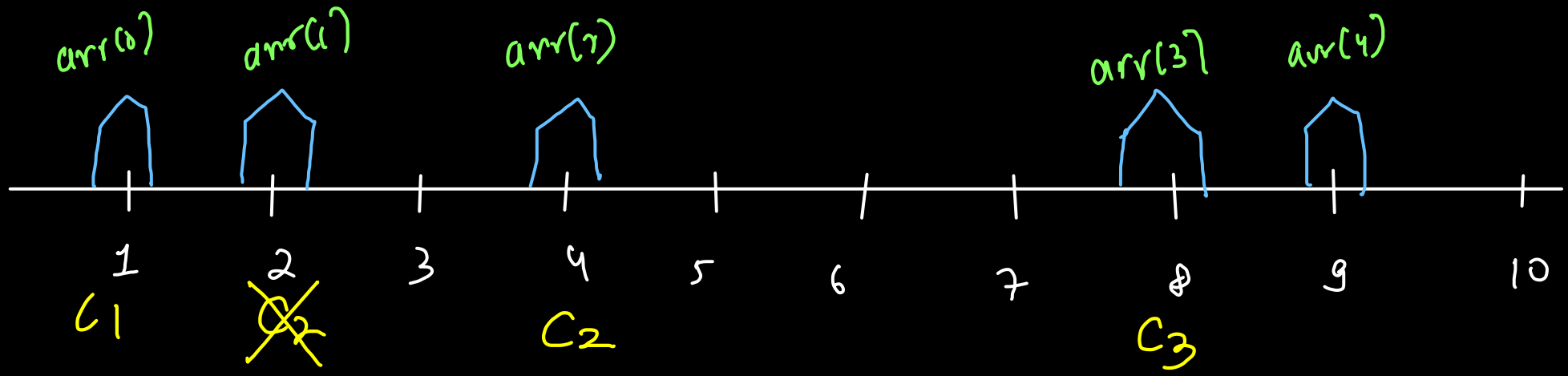
placed all cows : true

= left = mid+1

lastCow = arr(0) = 1

arr(2) = 4

arr(3) = 8



cows = 3

left = mid+1 = 2+1 = 3

mid = 3

placedCows = ~~1~~ ~~2~~ 3

placed all cows ⟹ true

left = mid+1 = 3+1 = 4

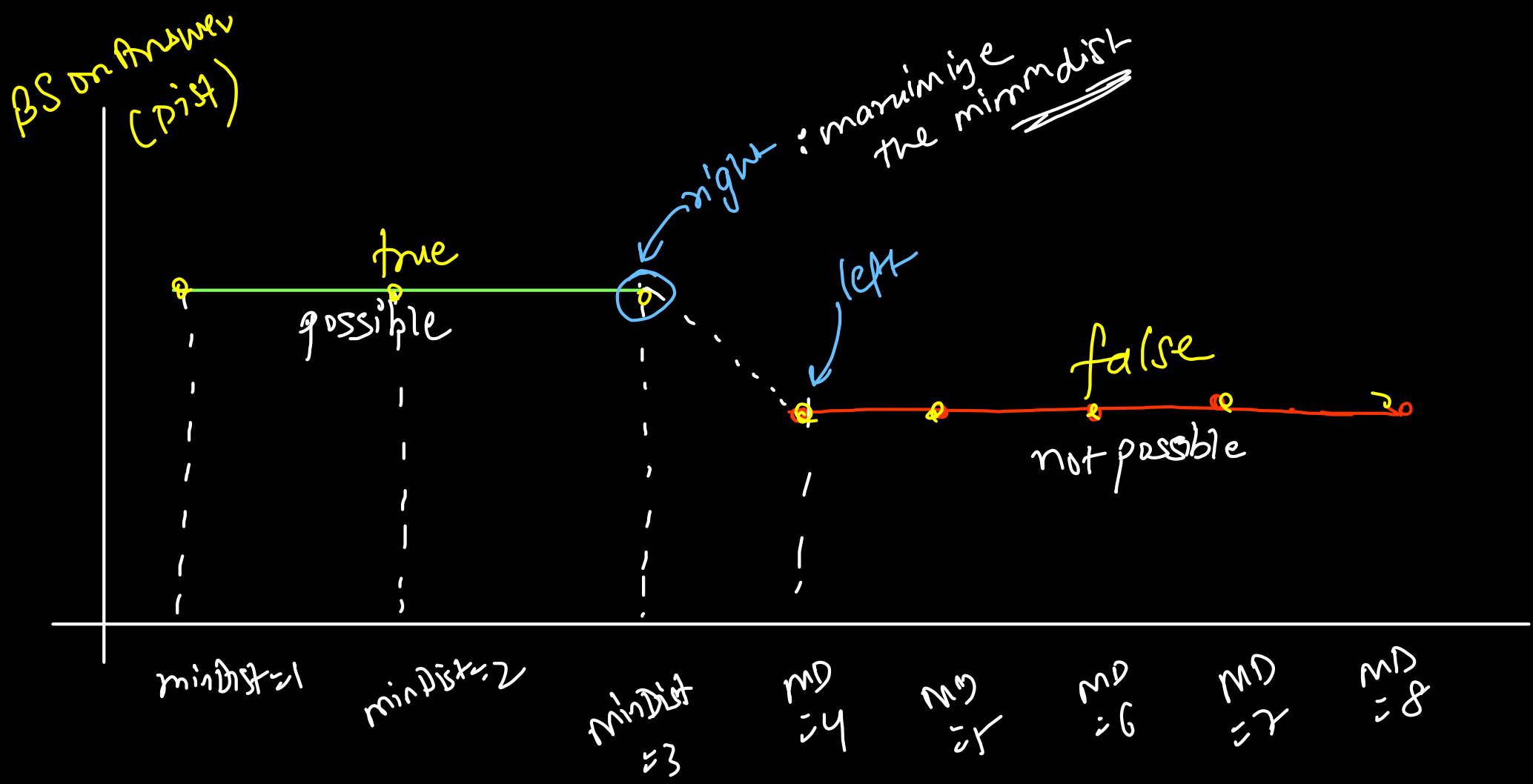
lastFlow = ~~arr(0) = 1~~

~~arr(2) = 4~~

arr(3) = 8

right = 3

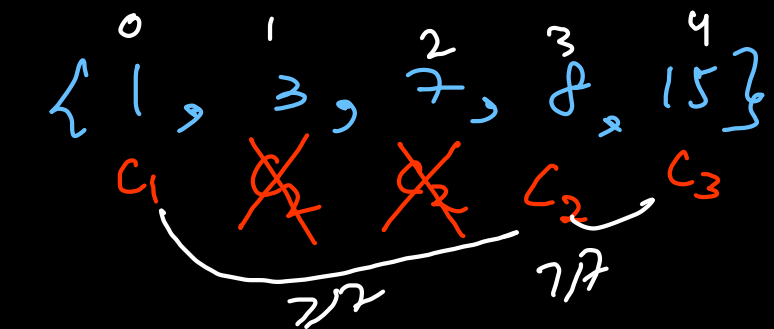
answer after binary search


BS on Answer (Dist)
right : maximize the minmdist
true
possible
left
false
not possible
minDist=1
minDist=2
minDist =3
MD =4
MD =5
MD =6
MD =7
MD =8

```java
public static boolean isPossible(ArrayList<Integer> stalls, int minDist, int totalCows){
    int placedCows = 1, lastIdx = 0;

    for(int currentIdx = 1; currentIdx < stalls.size(); currentIdx++){
        int dist = stalls.get(currentIdx) - stalls.get(lastIdx);

        if(dist >= minDist) {
            placedCows++;
            lastIdx = currentIdx;
        }
    }

    return (placedCows >= totalCows);
}
public static int aggressiveCows(ArrayList<Integer> stalls, int cows)  {
    Collections.sort(stalls);
    // To Find Adjacent Distance, Not to apply Binary Search
    int left = 1; // Minimum Possible Adjacent Distance
    int right = stalls.get(stalls.size() - 1) - stalls.get(0);

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

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

    return right;
}
```

O(n)

O(log range)



left = 1 mid = 7 right = 14

cows = 3 placedcows = 1 2 3

lastcowidx = 0 4

Total time O(n log range)

$10^5 * \log 10^9$

$10^5 * 9 * 3$

$10^5 * 27$

Woodcutting - E120

There is given an integer array **A** of size **N** denoting the heights of **N** trees.

Lumberjack Ojas needs to chop down **B** metres of wood. It is an easy job for him since he has a nifty new woodcutting machine that can take down forests like wildfire. However, Ojas is only allowed to cut a single row of trees.

Ojas's machine works as follows: Ojas sets a height parameter **H** (in metres), and the machine raises a giant sawblade to that height and cuts off all tree parts higher than **H** (of course, trees not higher than **H** meters remain intact). Ojas then takes the parts that were cut off. For example, if the tree row contains trees with heights of 20, 15, 10, and 17 metres, and Ojas raises his sawblade to 15 metres, the remaining tree heights after cutting will be 15, 15, 10, and 15 metres, respectively, while Ojas will take 5 metres off the first tree and 2 metres off the fourth tree (7 metres of wood in total).

Ojas is **ecologically** minded, so he doesn't want to cut off more wood than necessary. That's why he wants to set his sawblade as high as possible. Help Ojas find the **maximum integer height** of the sawblade that still allows him to cut off **at least B** metres of wood.

NOTE:

- The sum of all heights will exceed **B**, thus Ojas will always be able to obtain the required amount of wood.

Example Input

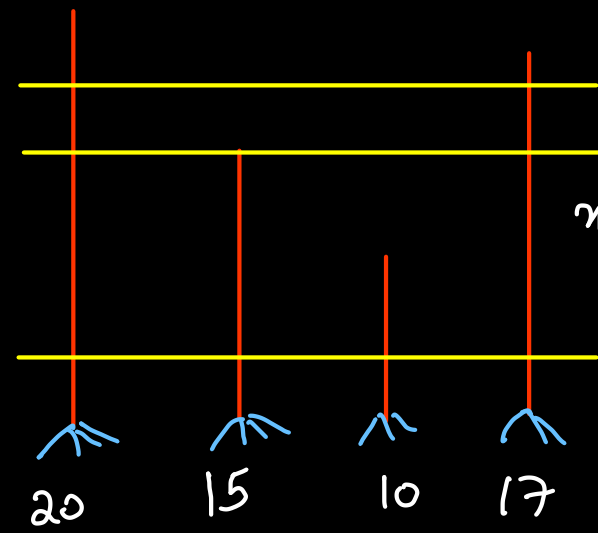
Input 1:

```
A = [20, 15, 10, 17]
B = 7
```

Input 2:

```
A = [4, 42, 40, 26, 46]
B = 20
```

answer = 15



x = 16 : (20-16) + (17-16) = 4+1 = 5

x = 15 : (20-15) + (17-15) = 7 ≥ 7

x = 3 : 17 + 12 + 7 + 14 ≥ 7

maxm Height cut ⇒ Requirement is fulfilled

low = 0 (root cut) ⇒ maxm wood

high = 20 (treetop cut ⇒ least wood)

mid = 10 (cut)

isPossible = (20-10) + (15-10) + (10-10) + (17-10) = 10 + 5 + 0 + 7 = 22

Binary search on height

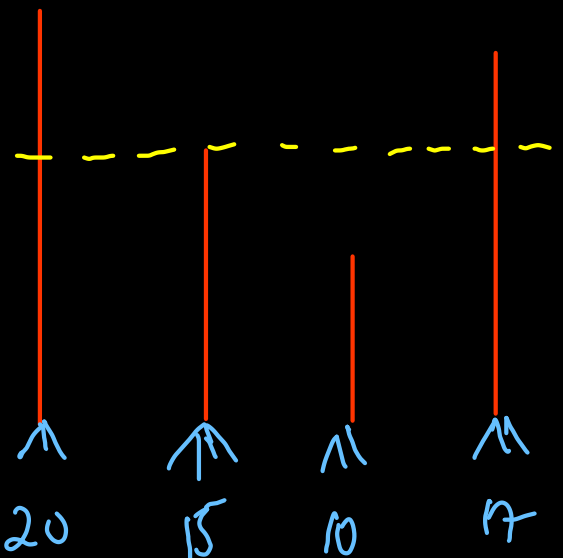
low = 0 (root cut) ⇒ no wood

right = 20 (tree top cut ⇒ least wood)

mid = 10 (cut)

isPossible = (20-10) + (15-10) + (10-10) + (17-10)

= 10 + 5 + 0 + 7 = 22 true



low = 10+1 = 11 , right = 20 , mid = 15

isPossible = (20-15) + (15-15) + (10-5)

+ (17-15) = 5+2 = 7 true

low = 16 , right = 20 , mid = 18

isPossible = (20-18) + (15-18) + (10-18) + (17-18)

= 2 false

low = 16, right = 17

requirement = 7

```java
class Solution {
    public boolean isPossible(int[] trees, int cut, int requiredWood) {
        int wood = 0;
        for (int treeHeight : trees) {
            if (cut < treeHeight)
                wood += (treeHeight - cut);
        }

        return (wood >= requiredWood);
    }

    public int solve(int[] trees, int requiredWood) {
        int left = 0, right = 1000000;

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

            if (isPossible(trees, mid, requiredWood) == true) {
                left = mid + 1; // maximize cut height => decrease wood
            } else {
                right = mid - 1; // wood less than required
                                 // => increase wood
            }
        }
        return right;
    }
}
```

O(n)

O(log n)

Total time

$\Rightarrow O(n * \log n)$

Problem Statement

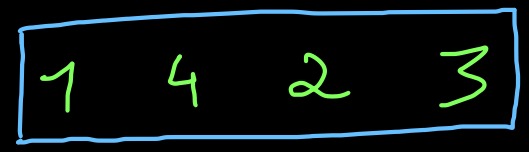
Suggest Edit

In Ninja Land, there is a famous restaurant named ‘CookingNinjas’. There are ‘N’ cooks in ‘CookingNinjas’ numbered from 0 to N-1. Each cook has rank ‘R’ (1 <= R <= 10).

A cook with a rank ‘R’ can prepare 1 dish in the first ‘R’ minutes, 1 more dish in the next ‘2R’ minutes, 1 more dish in next ‘3R’ minutes, and so on (A cook can only make complete dishes) For example if a cook is ranked 2. He will prepare one dish in 2 minutes, one more dish in the next 4 mins and one more in the next 6 minutes hence in a total of 12 minutes he can make 3 dishes, Note, In 13 minutes also he can make only 3 dishes as he does not have enough time for the 4th dish).

One day ‘CookingNinjas’ receive an order of ‘M’ dishes that they need to complete as early as possible. You are given an integer array ‘rank’ of size ‘N’ in which ‘rank[i]’ gives ‘rank’ of ith cook and an integer ‘M’, You need to find out the minimum times required to complete this order of ’M’ dishes.

Example



10 dishes

rank=1

4, 2, 3, 4, 5

1st 3rd 6th 10th 15th minute

rank=2

2, 4, 6, 8, 10

2nd 6th 12th 20th 30th

1	2	3	4

cooks

10 dishes

all dishes prepared by slowest cook

left = 1

$1+2+3+\dots+10 = \frac{10\times11}{2} = 55$

mid = 110

isPossible: true

right = $\frac{40\times11}{2}$

= 220

left = 1

mid = 55

isPossible: true

right = 109

left = 1

mid = 27 isPossible: true

right = 54

Rank : ~~1~~ ~~2~~ ~~3~~ 0+1+2+3+4+5+6+~~7~~

Count: ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ (10) 0+2+4+8+~~16~~

0+3

6 dishes by R_1

3 dishes by R_2

1 dish by R_3

Rank ⇒ K dishes

$R+2R+3R+4R+\dots+kR$

$\Rightarrow R(1+2+3+\dots+k)$

$\Rightarrow \frac{R(k)(k+1)}{2}$ time

```java
public static boolean isPossible(ArrayList<Integer> cooks, int timeAllowed, int requiredDishes){
    for(int rank: cooks){ → O(n)
        int mult = 1;
        for(int timeTaken = rank; timeTaken <= timeAllowed; timeTaken += rank * mult){ → O(√range)
            requiredDishes--; mult++;
            if(requiredDishes == 0) return true;
        }
    }
    return false;
}

public static int minCookTime(ArrayList<Integer> cooks, int dishes) {
    Collections.sort(cooks); // Fastest Cook is assigned first
    int left = 1, right = (cooks.get(0) * dishes * (dishes + 1)) / 2;

    while(left <= right){ → O(log range)
        int mid = left + (right - left) / 2;
        if(isPossible(cooks, mid, dishes) == true){
            right = mid - 1; // minimize the time to prepare all dishes
        } else {
            left = mid + 1; // not possible in so much less time => increase time
        }
    }

    return left;
}
```

$1R + 2R + 3R + \dots + kR = T$

$T = \frac{R(k)(k+1)}{2}$

$R = \sqrt{T}$

Total time

$O(n\sqrt{R}\log R)$

Step Array Search

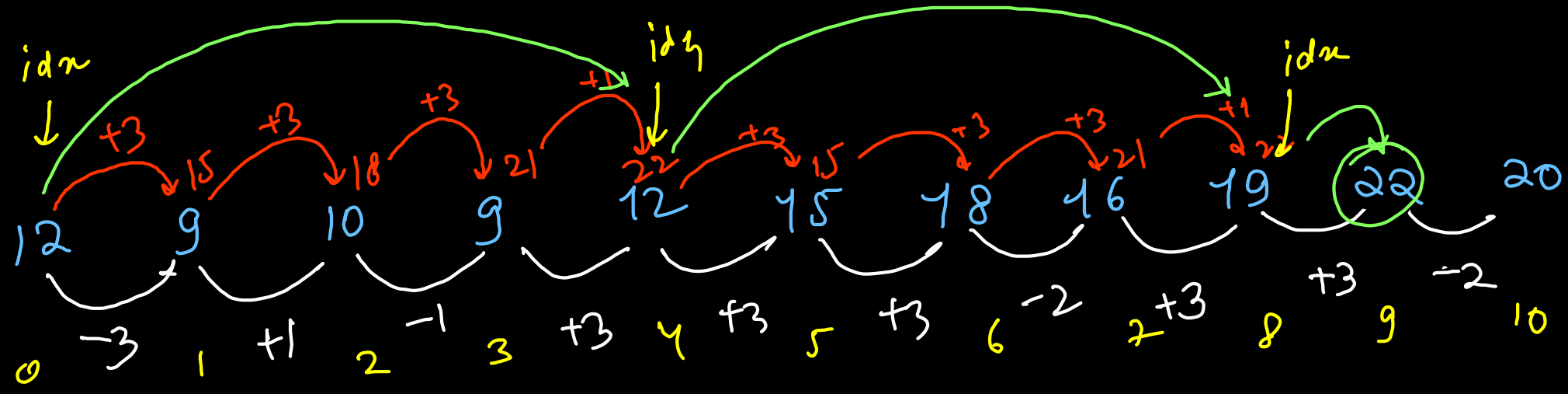
K = 3

Jump Search → O(root N)

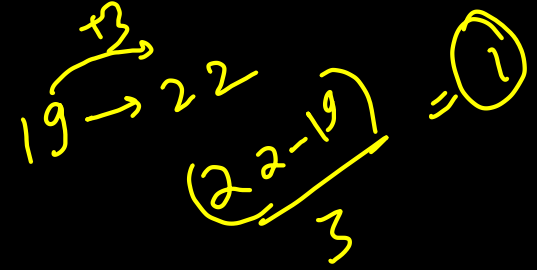
Better than linear search

array sorted

Bur than binary search



target = 22

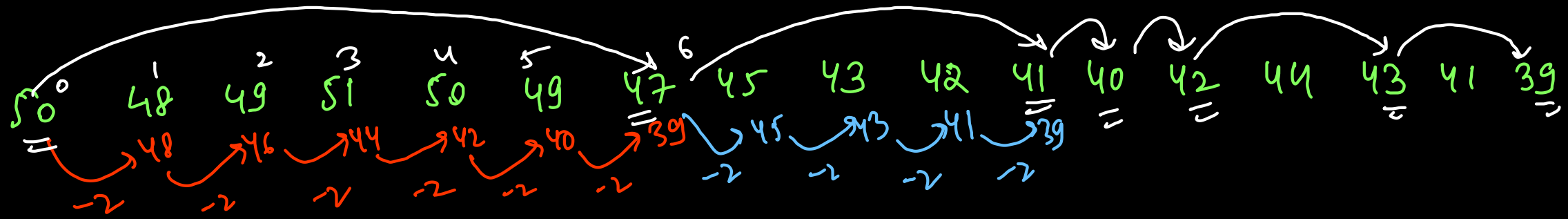


reg jump = (22-12) = 10

each jump = +3 (k)

min^m _ steps = ⌈10/3⌉ = 4

$k = 2$



target = (39)

req jump ⇒ 50 — 39 = 11

each jump = k = 2

$\text{minm jumps} = \lceil 11/2 \rceil = (6)$

req jumps ⇒ 47 − 39 = 8

each jump ⇒ k = 2

$\text{minm jump} = \lceil 8/2 \rceil = (4)$

req jump = (41 − 39) = 2

each jump = 2

$\text{minm jump} = 2/2 = 1$

$(40 - 39) = 1$

2

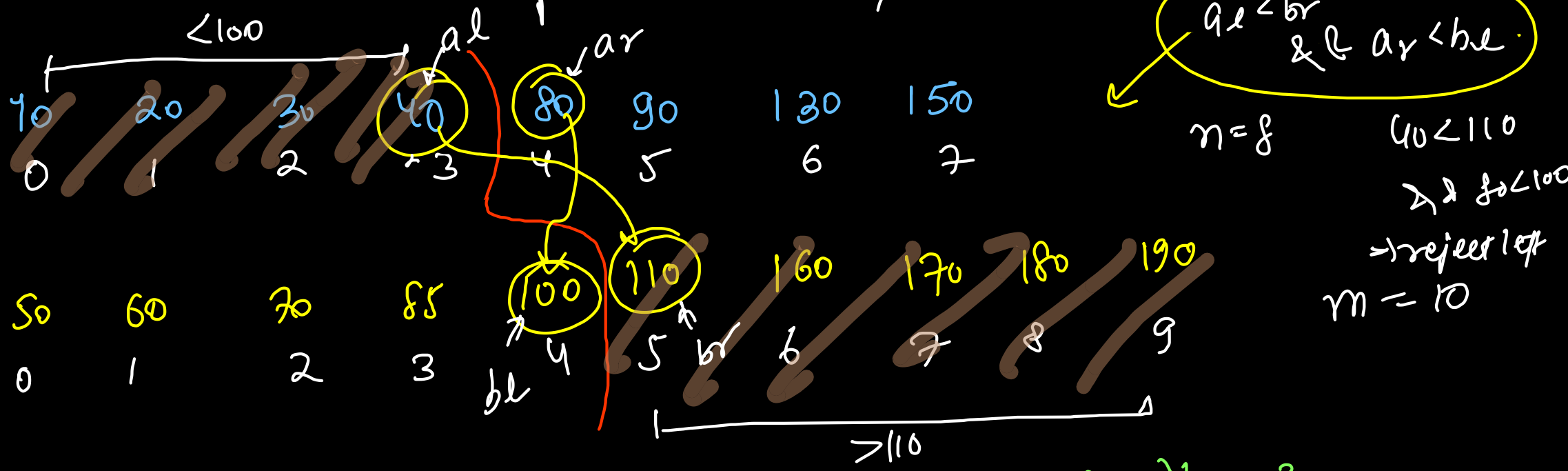
$\lceil 1/2 \rceil = 1$

```java
public static int search (int arr[], int n, int target, int jump) {
    int idx = 0;

    while(idx < n){
        if(arr[idx] == target) return idx;

        int requiredJump = Math.abs(arr[idx] - target);
        int minJumps = requiredJump / jump; // floor division
        if(requiredJump % jump != 0) minJumps++; // ceil division

        idx = idx + minJumps;
    }

    return -1;
}
```

Worst case
↳ O(n)
linear search

Avg case
↳ O(n/k)

Median of 2 Sorted Arrays.



al < br && ar < bl.

n = 8

40 < 110 ✓ 80 < 100 ✓

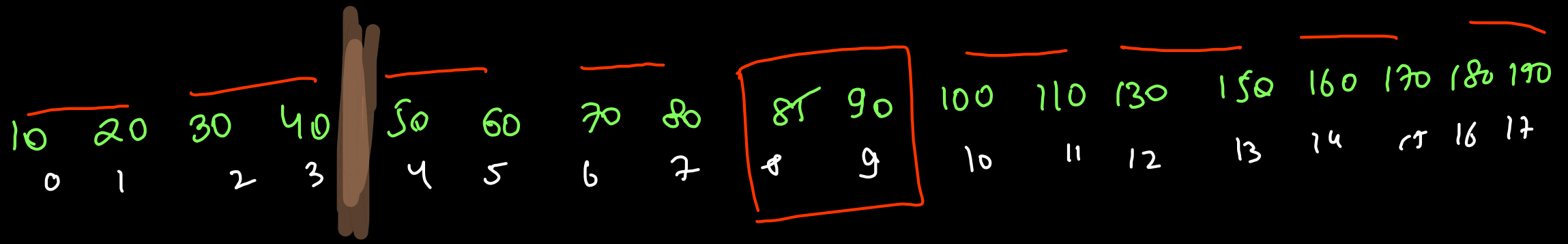
=> reject left

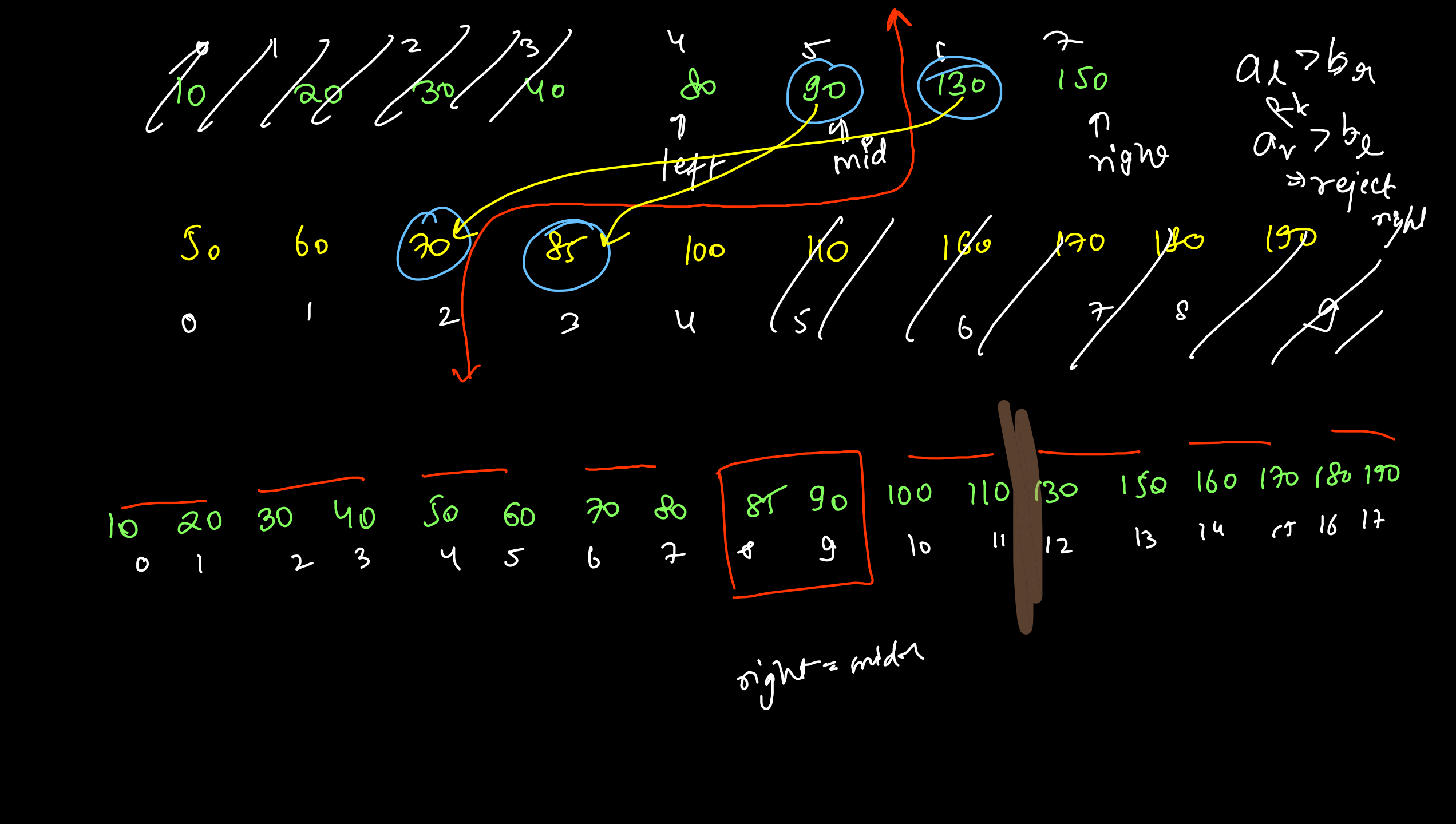
m = 10

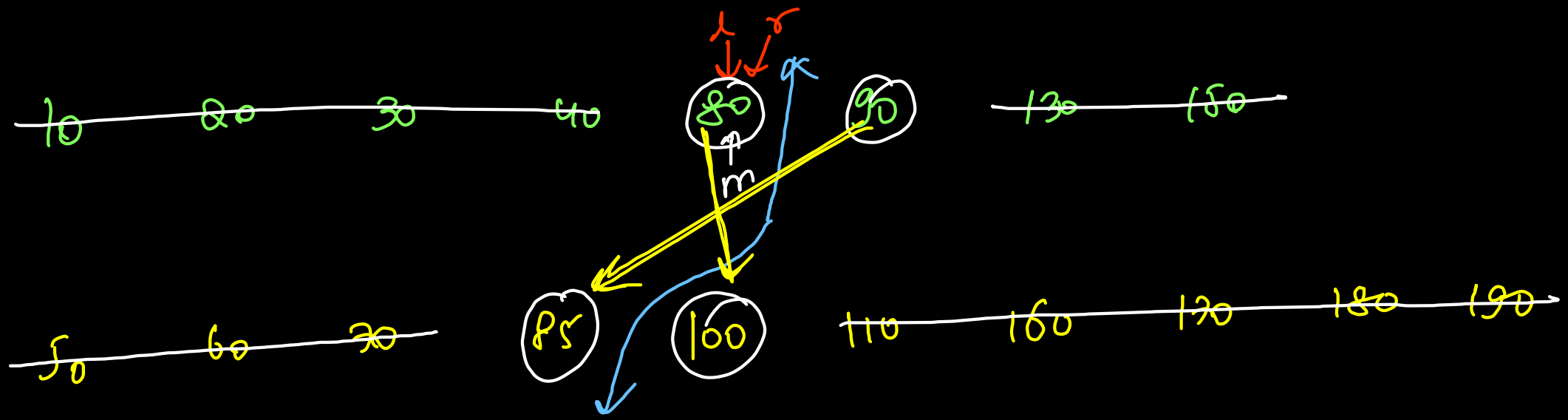
left = 0

right = n - 1 = 7

mid = (0+7)/2 = 3




0 1 2 3 4 5 6 7
10 20 30 40 90 130 150
left
mid
right
a1 > b2
&
a2 > b1
⇒ reject
50 60 70 85 100 110 160 170 180 190
0 1 2 3 4 5 6 7 8 9
10 20 30 40 50 60 70 80 85 90 100 110 130 150 160 170 180 190
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
right = mid-1



$a_{l} = 80$

$a_{r} = 90$

$b_{l} = 85$

$b_{r} = 100$

$a_{l} < b_{r}$ && $a_{r} > b_{l}$

80 90

85 100

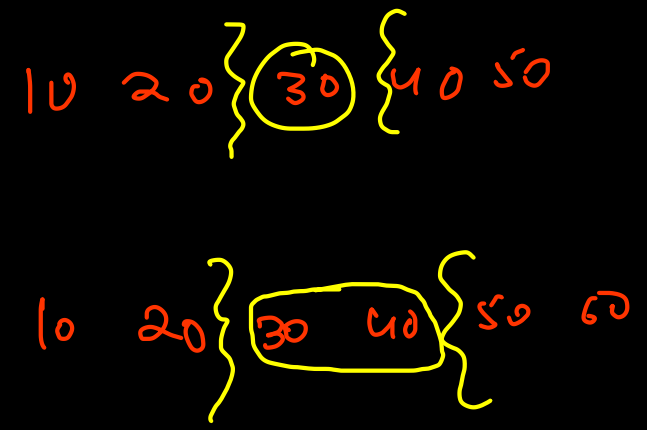
80	85	90	100
0	1	2	3

↓

Approach 1

① merge 2 Sorted Arrays → $O(m+n)$

② Find the median ← odd, even → $O(1)$



Approach 2

① Binary Search on 2 arrays.

```java
public double findMedianSortedArrays(int[] nums1, int[] nums2) {
    if(nums1.length > nums2.length){
        return findMedianSortedArrays(nums2, nums1);
    }

    int low = 0, high = nums1.length;
    int N = nums1.length + nums2.length;
    int Nby2 = (N + 1) / 2;
```

$n_1 > n_2$: swap arrays

corner case : 1

corner case (2)

ceil division

```java
while(low <= high){
    int mid = low + (high - low) / 2;

    int aleft = (mid - 1 >= 0)
                ? nums1[mid - 1] : Integer.MIN_VALUE;

    int aright = (mid < nums1.length)
                ? nums1[mid] : Integer.MAX_VALUE;

    int bleft = (Nby2 - mid - 1 >= 0)
                ? nums2[Nby2 - mid - 1] : Integer.MIN_VALUE;

    int bright = (Nby2 - mid < nums2.length)
                ? nums2[Nby2 - mid] : Integer.MAX_VALUE;

    if(aleft <= bright && bleft <= aright){
        // return median
        if(N % 2 == 1){
            // odd elements
            return Math.max(aleft, bleft);
        } else{
            int[] arr = {aleft, bleft, aright, bright};
            Arrays.sort(arr);
            return (arr[1] + arr[2]) / 2.0;
        }

    }
```

$a_l > b_r$ && $a_r > b_l$

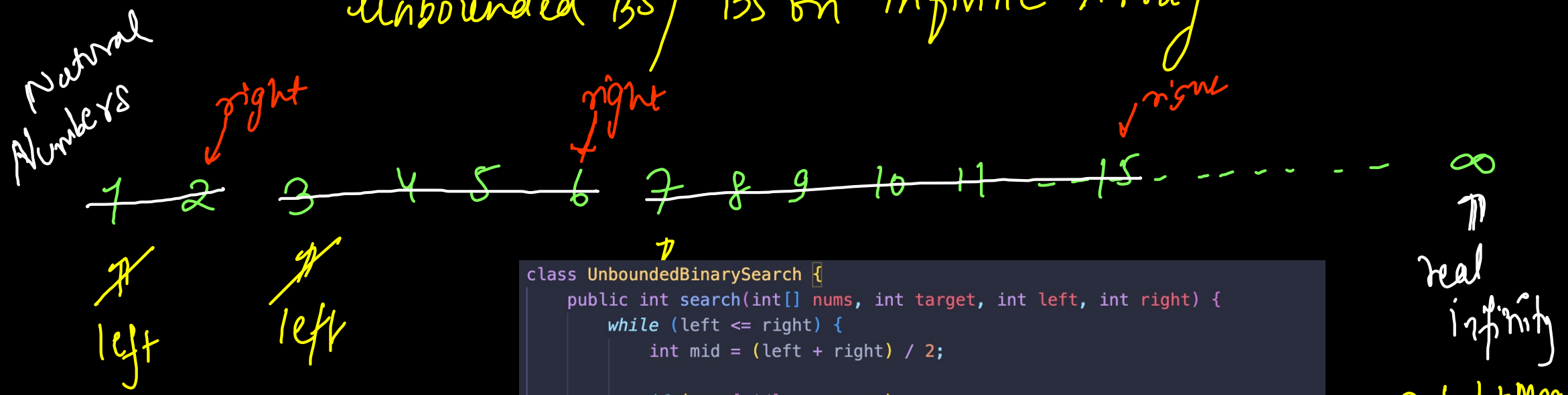
```java
    else if(aleft > bright){
        high = mid - 1;
    } else {
        // bleft > aright
        low = mid + 1;
    }
}

return 0.0;
```

0 1 2 3

$a_l < b_r$ && $a_r < b_l$

$a_l < b_r$ && $a_r > b_l$

Unbounded BS / BS on Infinite Array



target = very large

```java
class UnboundedBinarySearch {
    public int search(int[] nums, int target, int left, int right) {
        while (left <= right) {
            int mid = (left + right) / 2;

            if (nums[mid] == target)
                return mid; // successful search
            else if (nums[mid] < target)
                left = mid + 1;
            else
                right = mid - 1;
        }

        return -1; // unsuccessful search
    }

    public int infiniteSearch(int[] nums, int left, int right, int target) {
        if (nums[left] >= target && target <= nums[right]) {
            return search(nums, target, left, right);
        } else {
            return infiniteSearch(nums, right + 1, 2 * right, target);
        }
    }
```