Binary Search

Time complexity : $O(\log_2 N)$

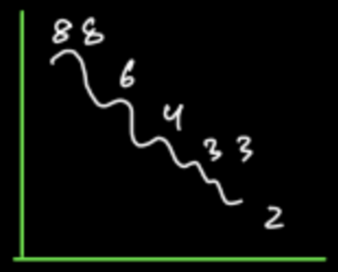
Constraint : Monotonic

- Binary
  - 0000 111
  - 111 000
- Increasing
  - Strictly
  - Non-strictly
- Decreasing
  - Strictly
  - Non strictly

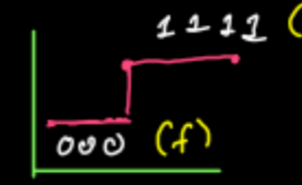
Modified Binary Search

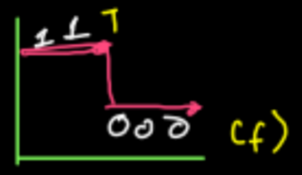
- Matrix
- Infinite (Unbounded)
- Nearly Sorted
- Rotated Sorted
- Bitonic

Non strictly inc



Non Strictly Dec





BS on a binary array is the Binary Search on Answer Concept & it is very very imp.

** The hardest problems on BS are mostly based on this concept.

Apply Binary Search

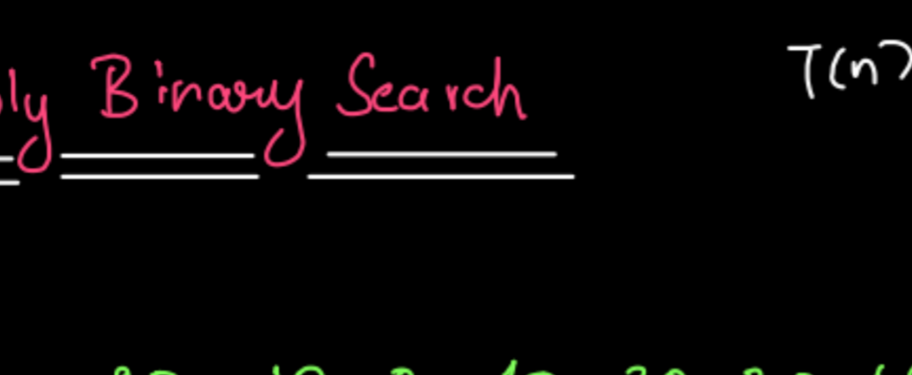
$T(n) = T(n/2) + K$ $O(\log_2 n)$

target => 60

-40	-20	-10	0	10	30	30	60	80	90	100
0	1	2	3	4	5	6	7	8	9	10

low, mid, high

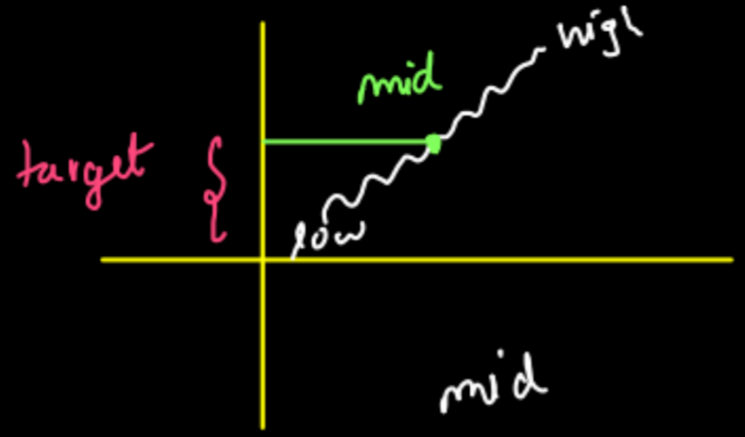
-40	-20	-10	0	10	(30)	30	60	80	90	100
0	1	2	3	4	5	6	7	8	9	10



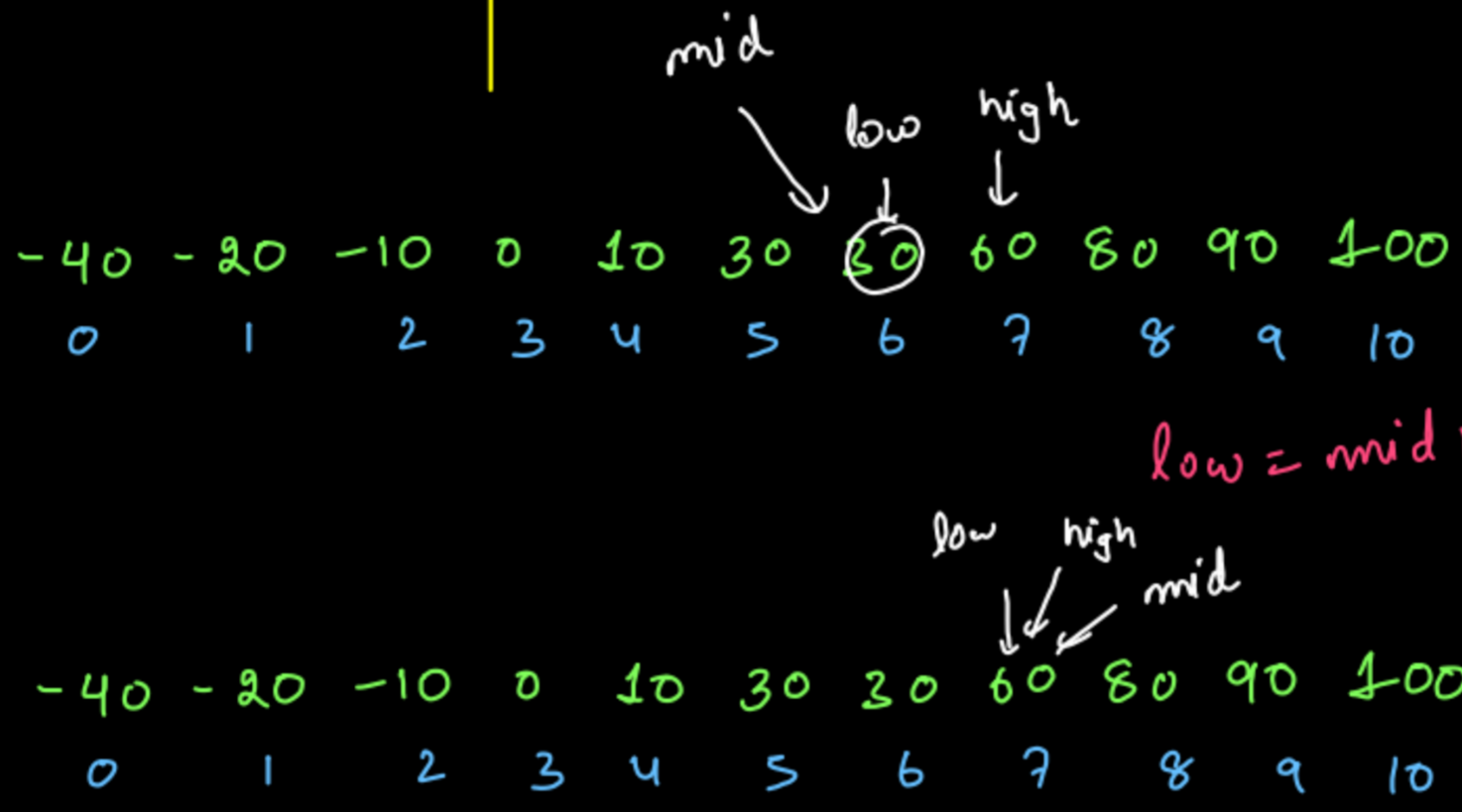
So, low = mid + 1

low, mid, high

-40	-20	-10	0	10	30	30	60	(80)	90	100
0	1	2	3	4	5	6	7	8	9	10



So, high = mid - 1



low = mid + 1

in case of unsuccessful search low > high & element is not found.

↗ This is leetcode code

```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;
       } else {
           right = mid - 1;
       }
   }

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

mid = (left + right) / 2 → can cause overflow

⇒ instead use

mid = left + (right - left) / 2;

Binary Search Recurrence Relation

```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;
       } else {
           right = mid - 1;
       }
   }

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

Recursive code ⟶

```
if(l>r) return -1;
int mid = lo+(hi-lo)/2;
if(arr[mid] == target)
    return mid;
else if (arr[mid] < target)
  return bs(arr, mid+1, hi);
else return bs(arr, lo, mid-1);
```

$T(n) = k + T(n/2)$

$T(n) = T(n/2) + k$

$T(n/2) = T(n/4) + k$

$\vdots$

$T(n/2^{x-1}) = T(n/2^x) + k$

$T(n) = T(n/2^x) + x * k$

for $n = 1$ $T(n) = 1$

$T(\frac{n}{2^x}) = T(1) \Rightarrow n = 2^x$

$\Rightarrow x = \log_2 n$

$T(n) = T(n/2^{\log_2 n}) + \log_2 n \, k$

$\boxed{T(n) = O(\log_2 N)}$

Avoiding overflow

$left = 10^9$

$right = 10^9$

$mid = \frac{left + right}{2} = (2 * 10^9) / 2$

↓

might cause overflow (exceed Integer range)

$\frac{l+r}{2} \Longleftrightarrow l + \frac{(r-l)}{2}$ ⤳ using this will not cause overflow

Transition Point (GfG)

Given a sorted array containing only 0s and 1s, find the transition point.

Example 1:

```
Input:
N = 5
arr[] = {0,0,0,1,1}
Output: 3
Explanation: index 3 is the transition
point where 1 begins.
```

If: first occurence of 1.

low mid high

{ 0, 0, 0, 0, 0, 1, 1, 1, 1 }
0 1 2 3 4 5 6 7 8

pa = -1

low mid high

{ 0, 0, 0, 0, 0, 1, 1, 1, 1 }
0 1 2 3 4 5 6 7 8

pa = -1 6

low high
mid

{ 0, 0, 0, 0, 0, 1, 1, 1, 1 }
0 1 2 3 4 5 6 7 8

pa = -1 5

high low

since low > high, STOP.

{ 0, 0, 0, 0, 0, 1, 1, 1, 1 }
0 1 2 3 4 5 6 7 8

pa = -1 5

Transition Point Code

```
int transitionPoint(int arr[], int n) {
    // code here
    int lo = 0;
    int hi = n-1;

    int pa = -1;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        if(arr[mid] == 0) {
            lo = mid + 1;
        } else {
            pa = mid;
            hi = mid -1;
        }
    }

    return pa;
}
```

$TC = O(\log_2 N)$

First Bad Version (Leetcode) (Asked in Google)

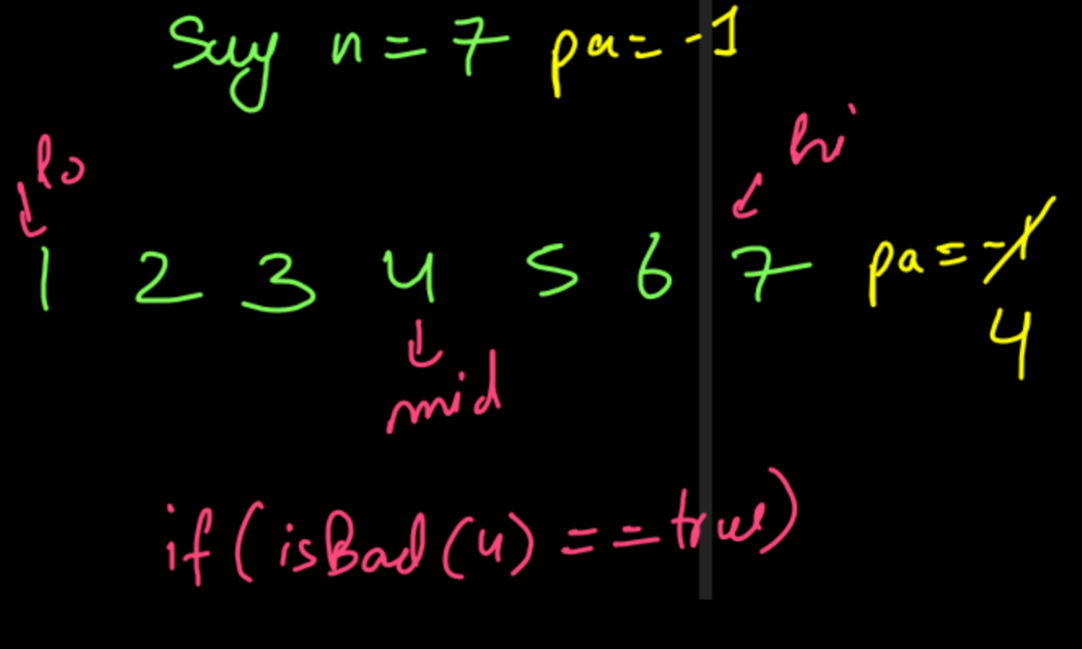
278. First Bad Version

Easy 3961 1444 Add to List Share

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.



Say n = 7 pa = -1

lo, hi

1 2 3 4 5 6 7 pa = ~~-1~~ 4

mid

if (isBad(4) == true)

mid

lo, hi

1 2 3 4 5 6 7

isBadVersion = false pa = 4

lo, hi

1 2 3 4 5 6 7

isBadVersion = true pa = ~~4~~ 3

hi, lo

since lo > hi = STOP

1 2 3 4 5 6 7

pa = 3 ✓

first Bad Version Code

```java
public class Solution extends VersionControl {
    public int firstBadVersion(int n) {

        int lo = 1;
        int hi = n;

        int pa = -1;
        while(lo <= hi) {
            int mid = lo + (hi-lo)/2;

            if(isBadVersion(mid)) {
                pa = mid;
                hi = mid -1;
            } else {
                lo = mid + 1;
            }
        }

        return pa;
    }
}
```

Guess Number Higher or Lower (leetcode)

374. Guess Number Higher or Lower

Easy 541 92 Add to List Share

We are playing the Guess Game. The game is as follows:

I pick a number from `1` to `n`. You have to guess which number I picked.

Every time you guess wrong, I will tell you whether the number I picked is higher or lower than your guess.

You call a pre-defined API `int guess(int num)`, which returns 3 possible results:

- `-1`: The number I picked is lower than your guess (i.e. `pick < num`).
- `1`: The number I picked is higher than your guess (i.e. `pick > num`).
- `0`: The number I picked is equal to your guess (i.e. `pick == num`).

Return *the number that I picked*.

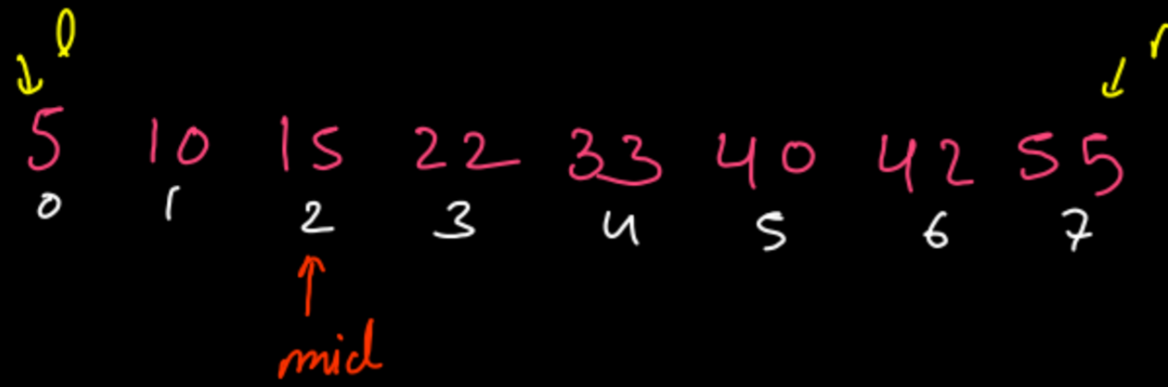
Almost same as prev question writing code directly -

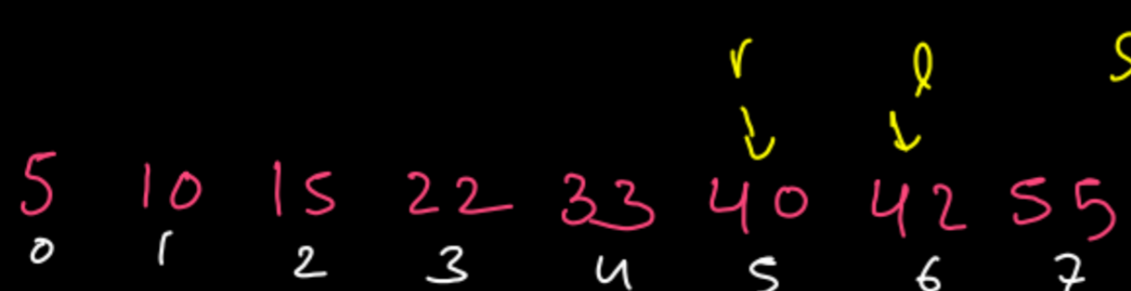
Guess No Higher or Lower Code

```java
public class Solution extends GuessGame {
    public int guessNumber(int n) {

        int lo = 1;
        int hi = n;

        while(lo <= hi) {

            int mid = lo + (hi-lo)/2;

            if(guess(mid) == 0) {
                return mid;
            } else if(guess(mid) == -1) {
                hi = mid - 1;
            } else {
                lo = mid + 1;
            }
        }

        return -1; //unreacheble statement
    }
}
```

Broken Economy (Ceil & Floor) (Pepcoding)

target = 41





Since $l > r$

STOP

arr[r] = floor

arr[l] = ceil

Broken Economy Code

```java
public static void ceilAndFloor(int[] arr, int target) {
    int ceil = Integer.MAX_VALUE; //bado me sabse chota
    int floor = Integer.MIN_VALUE; //choto me sabse bada

    int lo = 0;
    int hi = arr.length-1;

    while(lo <= hi) {
       int mid = lo + (hi-lo)/2;

       if(arr[mid] == target) {
           ceil = arr[mid];
           floor = arr[mid];
           break;
       } else if(arr[mid] < target) {
           if(arr[mid] > floor) {
               floor = arr[mid];
           }
           lo = mid + 1;
       } else {
           if(arr[mid] < ceil) {
               ceil = arr[mid];
           }
           hi = mid - 1;
       }
    }

    System.out.println(ceil);
    System.out.println(floor);
}
```

First And Last Position of Element in Sorted Array

{Leetcode}

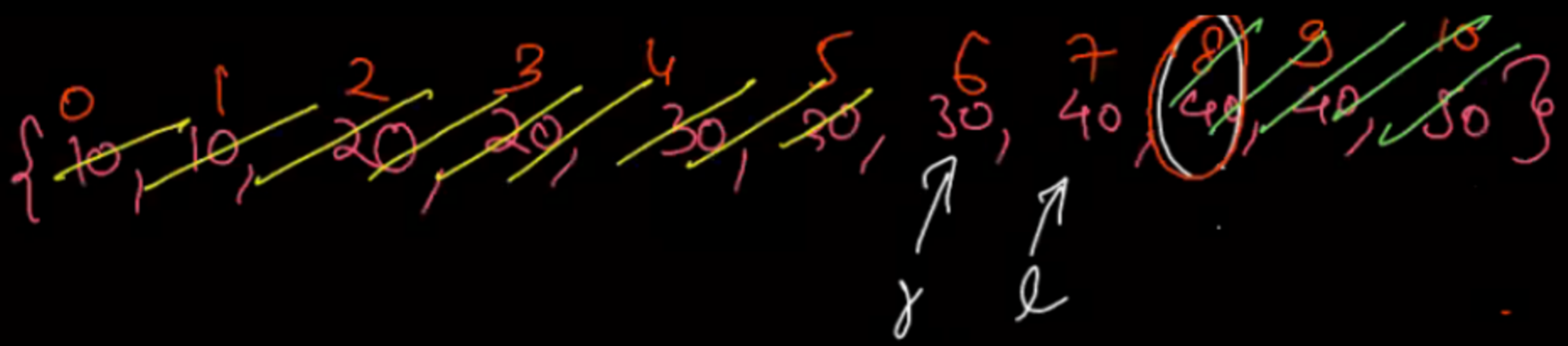
34. Find First and Last Position of Element in Sorted Array

Medium 8848 267 Add to List Share

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

You must write an algorithm with `O(log n)` runtime complexity.



First & Last Position Code

```java
public int[] searchRange(int[] nums, int target) {

    int[] ans = new int[2];
    ans[0] = firstIndex(nums,target);
    ans[1] = lastIndex(nums,target);

    return ans;
}
```

```java
private int firstIndex(int[] arr,int target) {
    int lo = 0;
    int hi = arr.length-1;

    int pans = -1;
    while(lo <= hi) {
        int mid = lo+ (hi-lo)/2;
        if(arr[mid] == target) {
            pans = mid;
            hi = mid - 1;
        } else if(arr[mid] < target) {
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }

    return pans;
}
```

```java
private int lastIndex(int[] arr, int target) {
    int lo = 0;
    int hi = arr.length-1;

    int pans = -1;
    while(lo <= hi) {
        int mid = lo+ (hi-lo)/2;
        if(arr[mid] == target) {
            pans = mid;
            lo = mid + 1;
        } else if(arr[mid] < target) {
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }

    return pans;
}
```

Search Insert Position

35. Search Insert Position

Easy 6404 365 Add to List Share

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

if element is found then return mid
else return left.

Search Insert Position Code

```java
public int searchInsert(int[] nums, int target) {

    int lo = 0;
    int hi = nums.length-1;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;
        if(nums[mid] == target) {
            return mid;
        } else if(nums[mid] < target) {
            lo = mid + 1;
        } else {
            hi = mid - 1;
        }
    }

    return lo;
}
```

Count Number of Occurrences (GfG)

lo = 1, hi = 6, target = 2

0	1	2	3	4	5	6
1	1	2	2	2	2	3

⇒ find firstIndex = 2

⇒ find LastIndex = 5

⇒ No of occ = li - fi + 1 = 5 - 2 + 1
= 4

Count Number of Occurrences Code

```java
int firstIndex(int[] arr, int n, int x) {
    int lo = 0;
    int hi = arr.length - 1;

    int pa = -1;
    while(lo <= hi) {
        int mid = lo+ (hi-lo)/2;

        if(arr[mid] == x) {
            pa = mid;
            hi = mid-1;
        } else if(arr[mid] > x) {
            hi = mid - 1;
        } else {
            lo = mid + 1;
        }
    }

    return pa;
}
```

```java
int lastIndex(int[] arr, int n, int x) {
    int lo = 0;
    int hi = arr.length - 1;

    int pa = -1;
    while(lo <= hi) {
        int mid = lo+ (hi-lo)/2;

        if(arr[mid] == x) {
            pa = mid;
            lo = mid + 1;
        } else if(arr[mid] > x) {
            hi = mid - 1;
        } else {
            lo = mid + 1;
        }
    }

    return pa;
}
```

```java
int count(int[] arr, int n, int x) {
    int fi = firstIndex(arr,n,x);
    int li = lastIndex(arr,n,x);

    if(fi == -1) {
        return 0;
    }

    return li - fi + 1;
}
```

Lower & Upper Bound {Ques not avail anywhere}

Lower Bound: If element is found, return first occurence & if not found then return just greater value (ceil)

Upper Bound: Element found or not found, return ceil always

So, upper Bound = ceil always. {also, first occ of ceil only}

{ 10, 19, 19, 20, 20, 49, 49, 49, 50, 50 } target = 20
  0   1   2   3   4   5   6   7   8   9

LB = 3 UB = 5

{ 10, 19, 19, 20, 20, 49, 49, 49, 50, 50 } target = 30
  0   1   2   3   4   5   6   7   8   9

LB = 5 UB = 5

{ 10, 19, 19, 20, 20, 49, 49, 49, 50, 50 } target = 5
  0   1   2   3   4   5   6   7   8   9

LB = 0 UB = 0

Search Insert Position is Lower Bound on Unique Elements

If we think now,

Search Insert position asked us to find the Lower Bound. because if the element is found we return its index else return just greater value (ceil).

↓

given in the ques that elements are distinct.

{ 10, 19 19, 20, 20, 49 49 40, 50, 50 }
0 1 2 3 4 5 6 7 8 9

target = 20

pa = 4 3

LB = 3

UB = 5

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

    //we have to return the first occ if found else ceil

    int lo = 0;
    int hi = arr.length -1;
    int pa = arr.length;

    while(lo <= hi) {
        if(arr[mid] >= target) {
            pa = mid;
            hi = mid -1;
        } else {
            lo = mid + 1;
        }
    }

    return pa;
}
```

mid = lo + (hi - lo)/2

```java
public static int upperBound(int[] arr, int target) {

    // we have to find the ceil only

    int lo = 0;
    int hi = arr.length-1;

    while(lo <= hi) {
        if(arr[mid] <= target) {
            lo = mid + 1;
        } else {
            hi = mid -1;
        }
    }

    return lo;
}
```

mid = lo + (hi - lo)/2

Closest Element {Exact not avail anywhere}

{ 10, 10, 10, 20, 25, 48, 48, 48, 50, 50 }
  0   1   2   3   4   5   6   7   8   9

target = 30

25 (index 4) 48 (index 5)

30 − 25 = 5 48 − 30 = 18

25 is closer

way of finding closest ele:

① Binary Search on Element
→ if search successful
⇒ return 0.

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

Cal Lowerbound.
ceil = Lb
floor = Lb − 1
} find which one is closer

for its code
go to Hecter's ques Lutcode (done in this notes)

K Closest Elements {leetcode}

target = 20
k = 3

l h
{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

① Apply binary Search
{ if element is found
floor = mid - 1;
ceil = mid + 1;

}

target = 20

l mid h

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

0 1 2 3 4 5 6 7 8 9

target = 20

l mid h

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

0 1 2 3 4 5 6 7 8 9

target = 20

mid

l h

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

0 1 2 3 4 5 6 7 8 9

target = 20

l h → mid

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

0 1 2 3 4 5 6 7 8 9

target = 20

k = 3

f c

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

0 1 2 3 4 5 6 7 8 9

k = 3 2 [20]

compare f & c values

target = 20

k = 3

f c

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

0 1 2 3 4 5 6 7 8 9

k = 2 1 [20, 25]

target = 20

k = 3

f f c

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

0 1 2 3 4 5 6 7 8 9

k = 2 1 0 [20, 25, 10]

target = 30
k = 3

{ 10, 10, 10, 20, 25, 48, 48, 48, 50, 50 }
  l   h

Apply binary Search
if element is not found

floor = hi;

ceil = lo;

target = 30
k = 3

l, mid, h

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}
0 1 2 3 4 5 6 7 8 9

target = 30
k = 3

l, mid, h

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}
0 1 2 3 4 5 6 7 8 9

mid

target = 30
k = 3

l h

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}
0 1 2 3 4 5 6 7 8 9

target = 30
k = 3

h l

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}
0 1 2 3 4 5 6 7 8 9

ceil = arr[lo] → 48

floor = arr[hi] = 25

target = 30
k = 3

f f f c

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}
0 1 2 3 4 5 6 7 8 9

k = 3 2 [25]

target = 30
k = 3

f f f c

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}
0 1 2 3 4 5 6 7 8 9

k = 3 2 [25, 20]

1

target = 30
k = 3

f f f f c

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}
0 1 2 3 4 5 6 7 8 9

k = 3 2 [25, 20, 48]

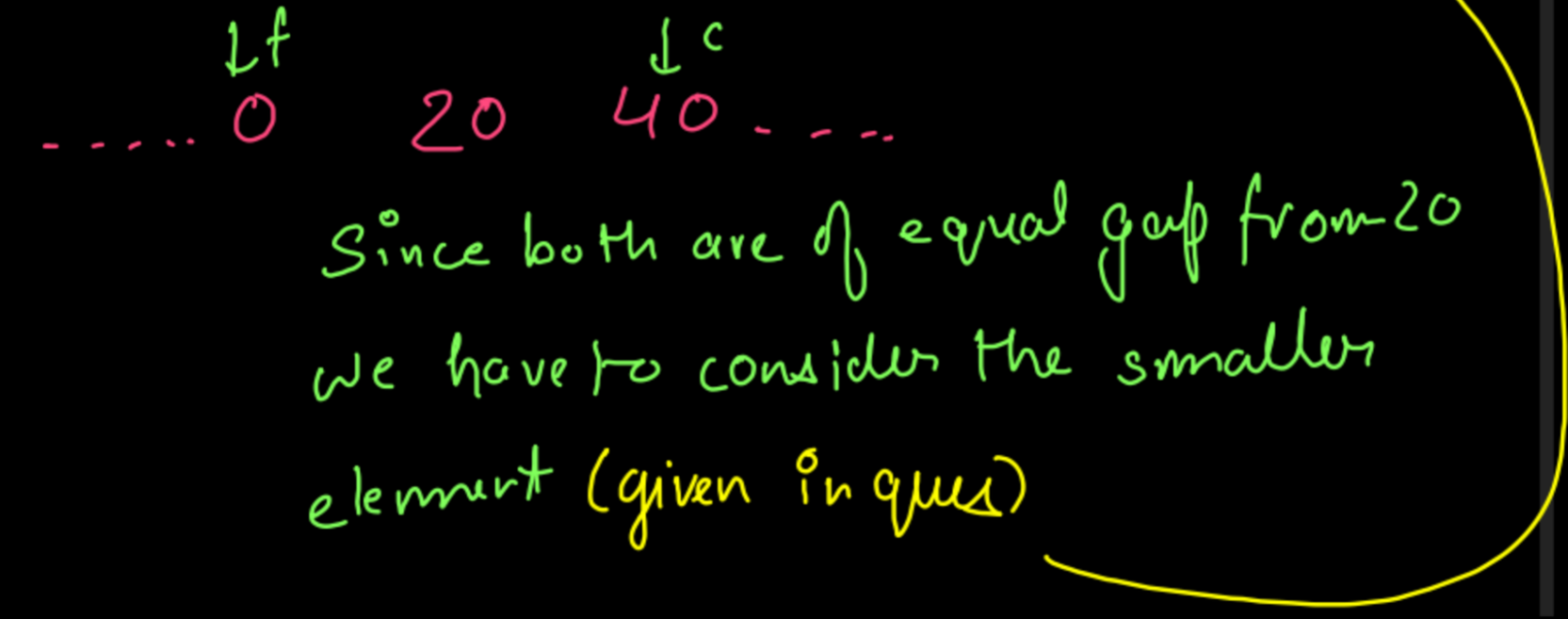
658. Find K Closest Elements

Medium 3691 389 Add to List Share

Given a **sorted** integer array `arr`, two integers `k` and `x`, return the `k` closest integers to `x` in the array. The result should also be sorted in ascending order.

An integer `a` is closer to `x` than an integer `b` if:

- `|a - x| < |b - x|`, or
- `|a - x| == |b - x|` and `a < b`



My code for K Closest Elements

```java
public List<Integer> findClosestElements(int[] arr, int k, int x) {

    int ceil = lowerBound(arr,x);
    int floor = ceil -1;

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

    while(floor >= 0 && ceil < arr.length && k-- > 0) {
        if(Math.abs(x-arr[floor]) < Math.abs(arr[ceil]-x)) {
            ans.add(arr[floor]);
            floor--;
        } else if(Math.abs(x-arr[floor]) > Math.abs(arr[ceil]-x)){
            ans.add(arr[ceil]);
            ceil++;
        } else {
          if(arr[floor] < arr[ceil]) {
              ans.add(arr[floor]);
              floor--;
          }  else {
              ans.add(arr[ceil]);
              ceil++;
          }
        }
    }
```

TC = lowerbound + Two pointer + Sorting the ArrayList

```java
while(floor >= 0 && k-- > 0) {
    ans.add(arr[floor]);
    floor--;
}

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

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

$$TC = \log_2 N + k + k\log_2 k$$

$$TC = O(\log_2 N + k + k\log_2 k)$$

Heaters (Leetcode 475)

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

heaters = [1, 8, 5]

Heater ka min radius nikalna tough ho sakta hai.

Rather, num har ek house se min dist par rakhe heater nikaal lenge.

2 → 1

3 → 2

11 → 1

9 → 1

11 → 3 → Unka max is our answer.

Since we want to find closest heater to every house, heaters array must be sorted.

⇒ Traverse in houses array. Pick house [i] & find its closest heater. Check if it is greater than curr max, update the maximum.

Code for Heaters

h_1 = heaters.length
h_2 = houses.length

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

        int maxRadius = Integer.MIN_VALUE;
        for(int i=0;i<houses.length;i++) {
            int curr = closest(heaters,houses[i]);
            maxRadius = Math.max(Math.abs(curr-houses[i]),maxRadius);
        }

        return maxRadius;
    }
```

→ $h_1 \log_2 h_1$

$O(h_2)$

$TC = O(h_1 \log_2 h_1 + h_2 + \log_2 h_1)$

```
    private int closest(int[] heaters, int house) {
        int lb = lowerBound(heaters,house);
        if(lb == heaters.length) return heaters[lb-1];
        else if(lb == 0) return heaters[0];

        else if(Math.abs(house-heaters[lb]) < Math.abs(house-heaters[lb-1])) {
            return heaters[lb];
        }

        else return heaters[lb-1];
    }
```

→ $O(\log_2 h_1)$

Category:

Root & Square

Sqrt(x) (LC 69)

pa = 0;

35

l mid h

1 2 3 . 17 . . 35

if(17 * 17 == 35)

l mid h

1 2 3 . 7 . . 16

if(7 * 7 == 35)

l mid h

1 2 3 4 5 6

if(3 * 3 == 35) pa = 3

l mid h

4 5 6 if(5 * 5 == 35) pa = 5

l h mid

6 if(6 * 6 == 35)

l > h

return pa;

↳ 5 (floor Sqrt)

Sqrt(x) Code

```java
public int mySqrt(int x) {

    if(x == 0) return 0;
    long lo = 1;
    long hi = x;

    long pa = 0;

    while(lo <= hi) {
        long mid = lo + (hi-lo)/2;

        if(mid * mid == x) return (int)mid;
        else if(mid * mid < x) {
            pa = mid;
            lo = mid + 1;
        } else {
            hi = mid -1;
        }
    }

    return(int) pa;
}
```

Square Root → fractional No

target = 60

Precision = 6

① Calculate floor square root → 7

2.1 → 7.0 | 7.1 | 7.2 | 7.3 ..... 7.9

2.2 → 7.x1 | 7.x2 | 7.x9

2.3 → 7.xy1 | 7.xy2 | 7.xy9

} This will be done using Linear Search

precision ↑

$(\log_2 n + p * 10)$

Square Root fractional No Code {With Precision}

```java
public double sqrtWithPrecision(int n, int precision) {
    double ans = floorSqrt(n);
    double i= 0.1; //initial precision

    while(precision-- > 0) {
        while(ans * ans <= n) ans += i;
        ans = ans-i;
        i = i/10;
    }

    return ans;
}
```

⇒ Instead, we can specify the precision
i.e if precison = 3

while (h-l > 0.0001)

if left & right are double values & we have to apply binary search, we can't directly write while(l<=r)

↑
for double l<=r is mostly not going to happen

Valid Perfect Square (LC 367)

Use the prev ques to calculate floor Sqrt. If sqrt*sqrt gives the input no, perfect square else not.

→ Most optimised

```
public boolean isPerfectSquare(int num) {

    if(num == 1) return true;

    int sqrt = mySqrt(num);

    if(sqrt*sqrt == num) return true;

    return false;
}
```

one more way

$100 = 2 \times 5 \times 2 \times 5$

$50 = 2 \times 5 \times 5$

All factors have even freq

odd freq of 2

Hence, 100 is perfect sq

N^{th} Root

N^{th} Root

No is M

```java
public int NthRoot(int n, int m)
{
    int low = 1, high = m;

while(low <= high){
    int mid = (low + high)/2;
    int mul = (int)Math.pow(mid,n);
    if(mul == m) return mid;
    else if(mul > m) high = mid - 1;
    else low = mid + 1;

}
return -1;

}
```

Example 1:

```
Input: n = 2, m = 9
Output: 3
Explanation: 3^2 = 9
```

Example 2:

```
Input: n = 3, m = 9
Output: -1
Explanation: 3rd root of 9 is not
integer.
```

n=2 m=9

① lo=1 hi=9 mid=4 pow(4,2)==9 high=mid-1

② lo=1 hi=3 mid=2 pow(2,2)==9 lo=mid+1

③ lo=3 hi=3 mid=3 pow(3,2)==9 ✓ return 3

Category :

Rotated Sorted Array

Minimum in Rotated Sorted Array {LC: 153}

0 1 2 3 4 5 ⇒ k = 0

5 0 1 2 3 4 ⇒ k = 1

4 5 0 1 2 3 ⇒ k = 2

3 4 5 0 1 2 ⇒ k = 3

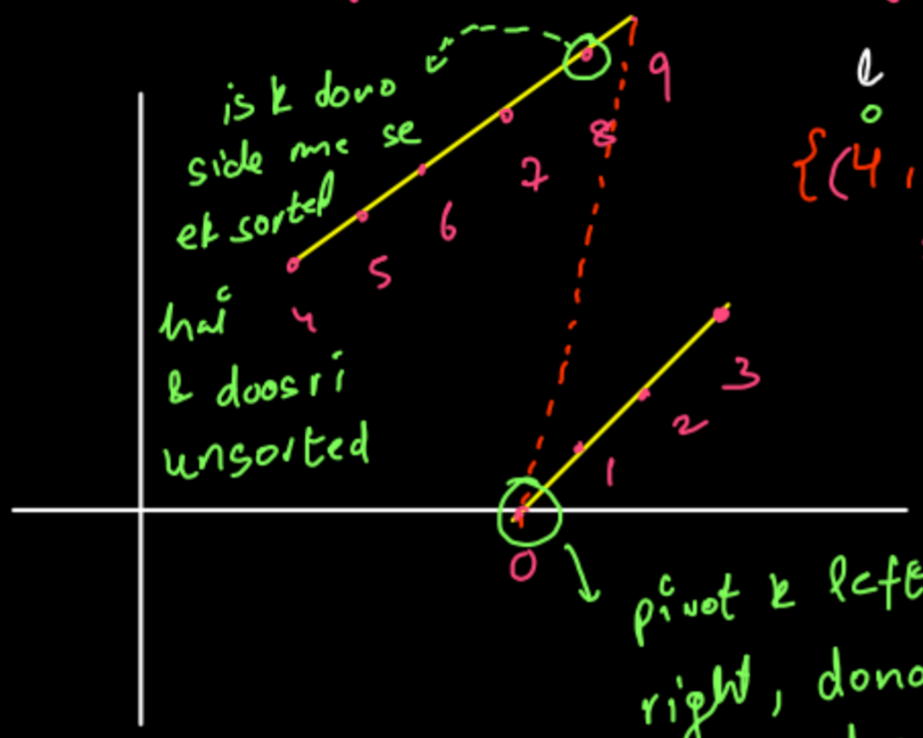
2 3 4 5 0 1 ⇒ k = 4

1 2 3 4 5 0 ⇒ k = 5

{Min can also be called as pivot as the array seems to be rotated around it}

(Elements will be Unique)

Solve using Modified Binary Search



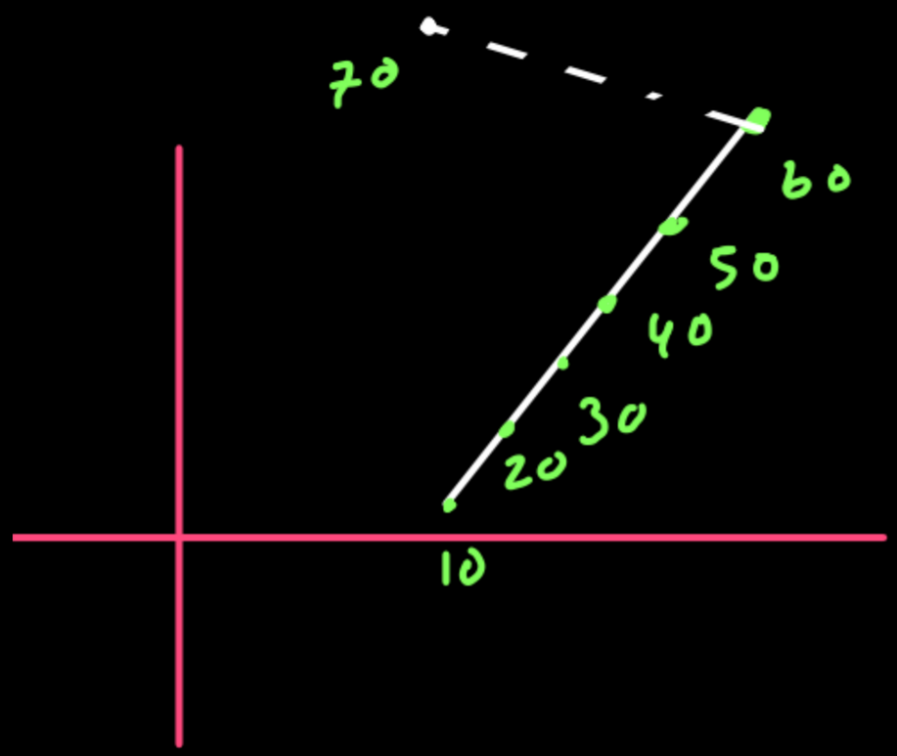
l mid h
0 1 2 3 4 5 6 7 8 9
{4, 5, 6, 7, 8, 9, 0, 1, 2, 3}
sorted unsorted
(pivot will lie here)

mid l → mid
mid h
0 1 2 3 4 5 6 7 8 9
{4, 5, 6, 7, 8, 9, 0, 1, 2, 3}

⇒ if(arr[l] < arr[mid])
⇓
left range is sorted {l = mid+1}

else left range is unsorted {h = mid-1}

```
if(mid == max)
    return mid+1;
if(mid == min)
    return mid;
```



0 1 2 3 4 5 6

70 10 20 30 40 50 60

l mid h

20 < 30 < 40

⇓

less than 30 (so, 30 is not mid)

Now, find unsorted region

mid

0 1 2 3 4 5 6

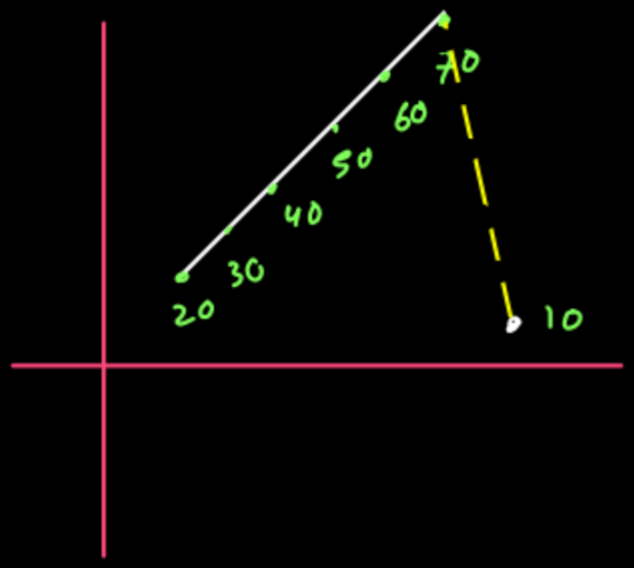
70 10 20 30 40 50 60

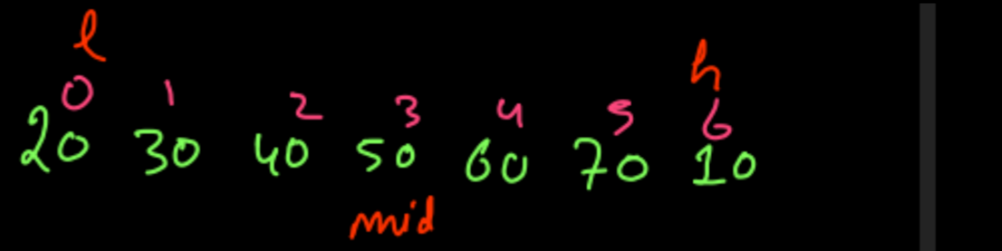
l h

since arr[mid+1] > arr[mid]

&& arr[mid-1] > arr[mid]

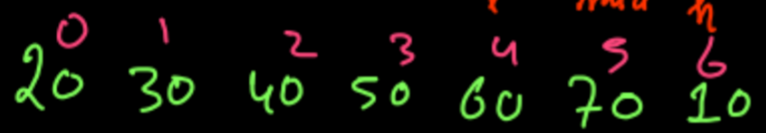
return arr[mid];



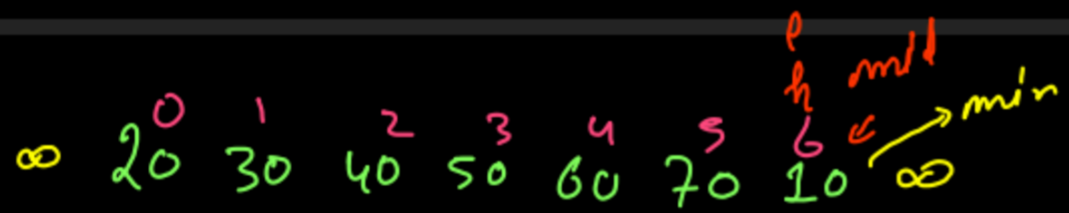


arr[mid-1] < arr[mid] < arr[mid+1]

arr[h] < arr[mid] ⇒ right part is unsorted



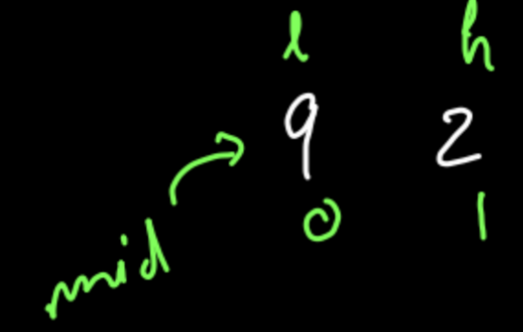
arr[h] > arr[mid]



return arr[mid];

{ Both corners ∞ for checking min. They will be taken as -∞ if checking max }

① If only 2 elements are remaining



if (arr [l] < arr [mid]) ⇒ it is equal

we can't decide which half sorted.

② Will not work for duplicates.

Code (LC: 153)

```java
private boolean isMin(int[] nums, int mid) {
        int lVal = (mid - 1 > 0) ? nums[mid-1] : Integer.MAX_VALUE;
        int rVal = (mid + 1 < nums.length) ? nums[mid + 1] : Integer.MAX_VALUE;

        if(nums[mid] < lVal && nums[mid] < rVal) return true;

        return false;
    }

    private boolean isMax(int[] nums, int mid) {
        int lVal = (mid - 1 > 0) ? nums[mid-1] : Integer.MIN_VALUE;
        int rVal = (mid + 1 < nums.length) ? nums[mid + 1] : Integer.MIN_VALUE;

        if(nums[mid] > lVal && nums[mid] > rVal) return true;

        return false;
    }
```

```java
public int findMin(int[] nums) {

    if(nums[0] < nums[nums.length-1]) return nums[0];

    int lo = 0;
    int hi  = nums.length-1;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        if(isMin(nums,mid)) { //is minimum
            return nums[mid];
        }

        else if(isMax(nums,mid)) { //is maximum
            return nums[mid + 1]; //this will be min
        }

        else if(nums[lo] < nums[mid]) { //is left part sorted-> discard the left part
            lo = mid + 1;
        }

        else if(nums[mid] < nums[hi]) { //is right part sorted-> discard the right part
            hi = mid - 1;
        }
    }

    return 0;
}
```

Pivot in Rotated Sorted Array - 2

{ 5, 5, 5, 5, 5, 5, 5, 0, 0, 0, 1, 1, 1 }
l = 0, mid = 6, h = 12
0 1 2 3 4 5 6 7 8 9 10 11 12

If you apply the condition arr[low] ≤ mid || arr[mid] ≤ high to check for sorted array, the previous approach will not work. This is shown below:

{ 5, 5, 5, 5, 5, 5, 5, 0, 0, 0, 1, 1, 1 }
l = 0, mid = 6, h = 12
0 1 2 3 4 5 6 7 8 9 10 11 12

Since arr[l] ≤ arr[mid] ⇒ left part is sorted

{ ~~5, 5, 5, 5, 5, 5, 5~~, 0, 0, 0, 1, 1, 1 }
l = 7, mid = 9, h = 12
0 1 2 3 4 5 6 7 8 9 10 11 12

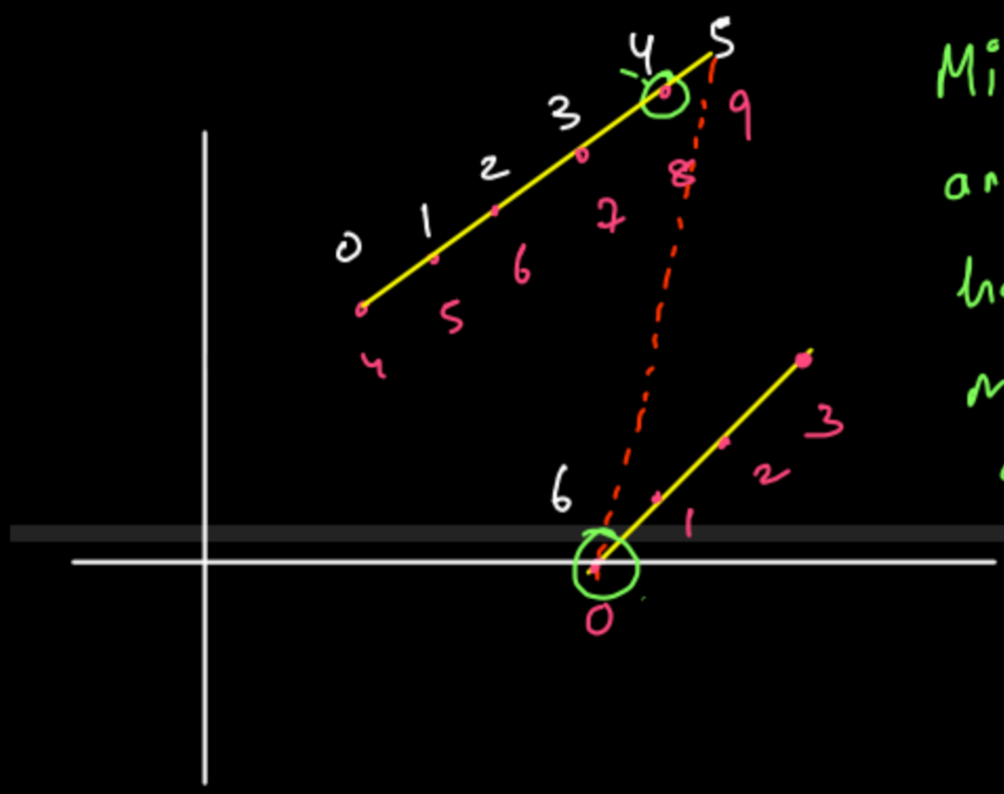
We discarded 0s but 0 is actually the pivot.

find Rotation Count (GfG)

Given an ascending sorted rotated array **Arr** of distinct integers of size **N**. The array is right rotated **K** times. Find the value of **K**.

find min / pivot in rotated sorted array {for unique ele in this ques}

Its index will be the answer.



Min is at index 6 but if the array was sorted, it should have been at index 0. This means it has moved 6 steps away from its sorted pos or array has been rotated by k = 6.

Code for find Rotation Count

```
int findKRotation(int arr[], int n) {
    // code here

    int pivotIndex = findMin(arr);
    return pivotIndex;
}
```

Searching in Rotated Sorted Array

33. Search in Rotated Sorted Array

Medium 12164 816 Add to List Share

There is an integer array `nums` sorted in ascending order (with **distinct** values).

Prior to being passed to your function, `nums` is **possibly rotated** at an unknown pivot index `k` (`1 <= k < nums.length`) such that the resulting array is `[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]` (**0-indexed**). For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the array `nums` **after** the possible rotation and an integer `target`, return *the index of* `target` *if it is in* `nums`, *or* `-1` *if it is not in* `nums`.

You must write an algorithm with `O(log n)` runtime complexity.

{4, 5, 6, 7, 8, 9, 10, 1, 2, 3}

target = 5

first Approach:

l

{4, 5, 6, 7, 8, 9, 10, 1, 2, 3}

h

find the pivot

BS in Left Sorted part

BS in Right Sorted part

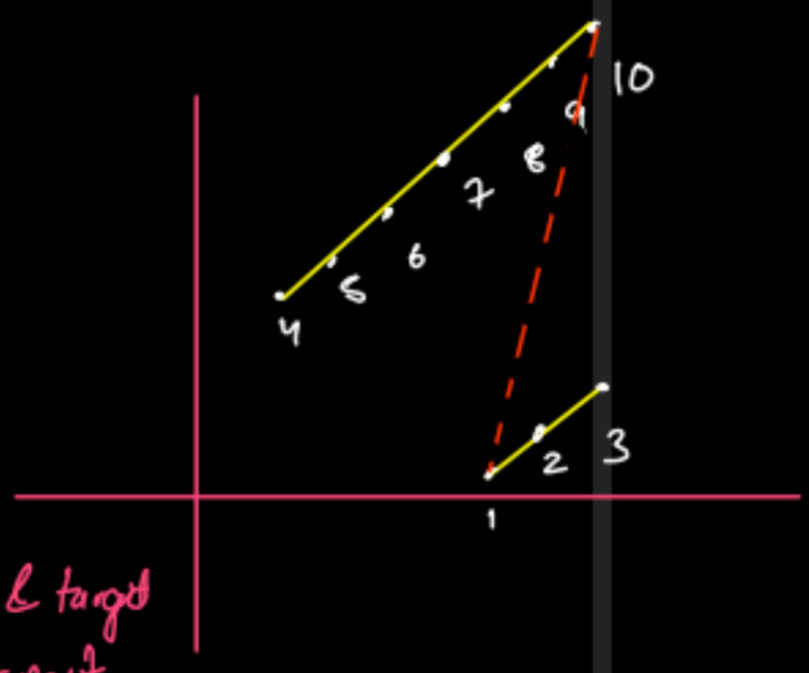
Solving in one BS

```
        l                 mid                 h
{ 4, 5, 6, 7, 8, 9, 10, 1, 2, 3 }
  0  1  2  3  4  5  6  7  8  9
```

```
if(arr[mid] == target) return mid;

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

Left part is sorted & target is in left part

Right is sorted and target is also in right.



Search in Rotated Sorted Array (Code)

```java
public int search(int[] arr, int target) {

    int lo = 0;
    int hi = arr.length-1;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        if(arr[mid] == target) {
            return mid;
        }
        if(arr[lo] <= arr[mid]) {
            if(target >= arr[lo] && target < arr[mid]) {
                hi = mid -1;
            } else {
                lo = mid + 1;
            }
        }
        else  {
            if(target > arr[mid] && target <= arr[hi]) {
                lo  = mid + 1;
            } else {
                hi = mid -1;
            }
        }
    }

    return -1;
}
```

(arr[lo] = target or target > mid) ③ ≤ 6 7 0 1 2

(same reason as above)

target = 3

l m h

4 5 6 7 0 1 2 (4 < 7)

0 1 2 3 4 5 6

l m h

4 5 6 7 0 1 2 (4 < 5)

0 1 2 3 4 5

lh → mid

4 5 6 7 0 1 2 if (4 < 4)

0 1 2 3 4 5

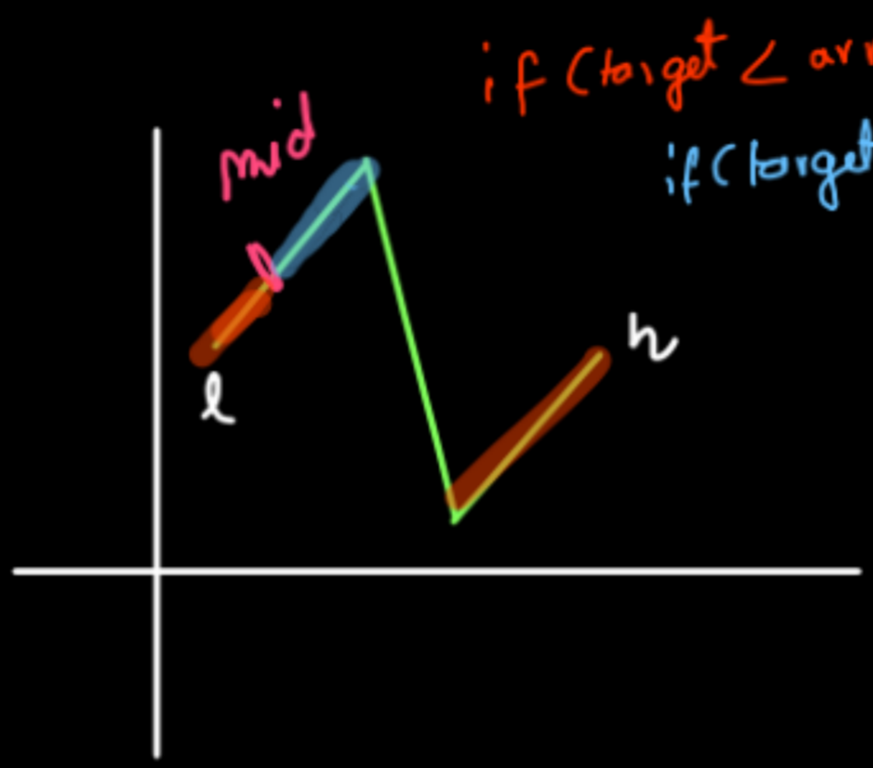
(left part not sorted)

h l

4 5 6 7 0 1 2 if (4 ≥ 4)

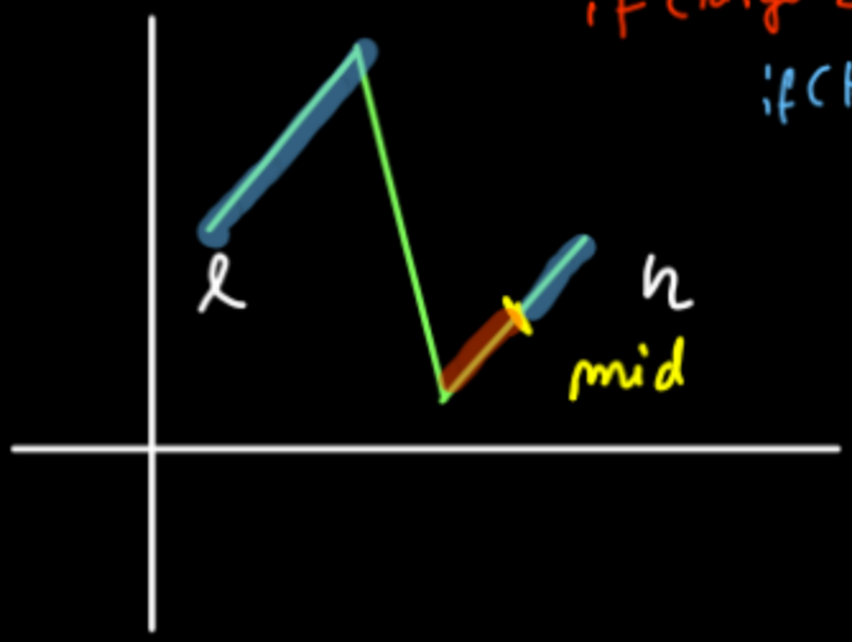
0 1 2 3 4 5 6 left part sorted

return -1



if (target < arr[mid])

if (target > arr[mid])



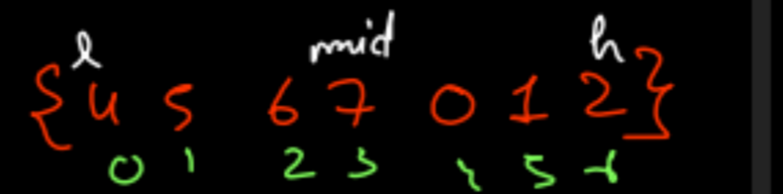
if (target < arr[mid])

if (target > arr[mid])

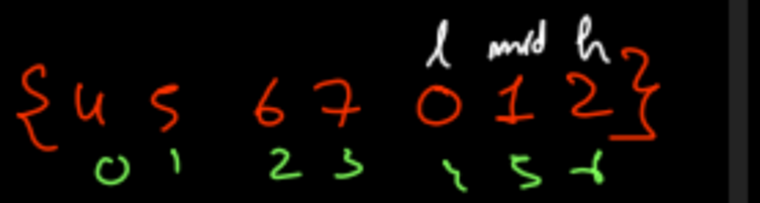
target = 3

```java
public int search(int[] arr, int target) {

    int lo = 0;
    int hi = arr.length-1;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        if(arr[mid] == target) {
            return mid;
        }
        if(arr[lo] <= arr[mid]) {
            if(target >= arr[lo] && target < arr[mid]) {
                hi = mid -1;
            } else {
                lo = mid + 1;
            }
        }
        else  {
            if(target > arr[mid] && target <= arr[hi]) {
                lo  = mid + 1;
            } else {
                hi = mid -1;
            }
        }
    }

    return -1;
}
```

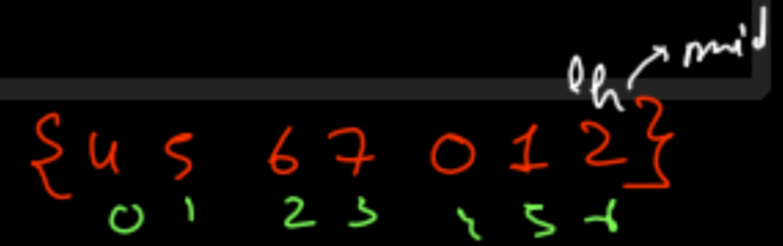
why ==



left range is sorted but target does not lie in the range



left range is sorted but target does not lie in the range

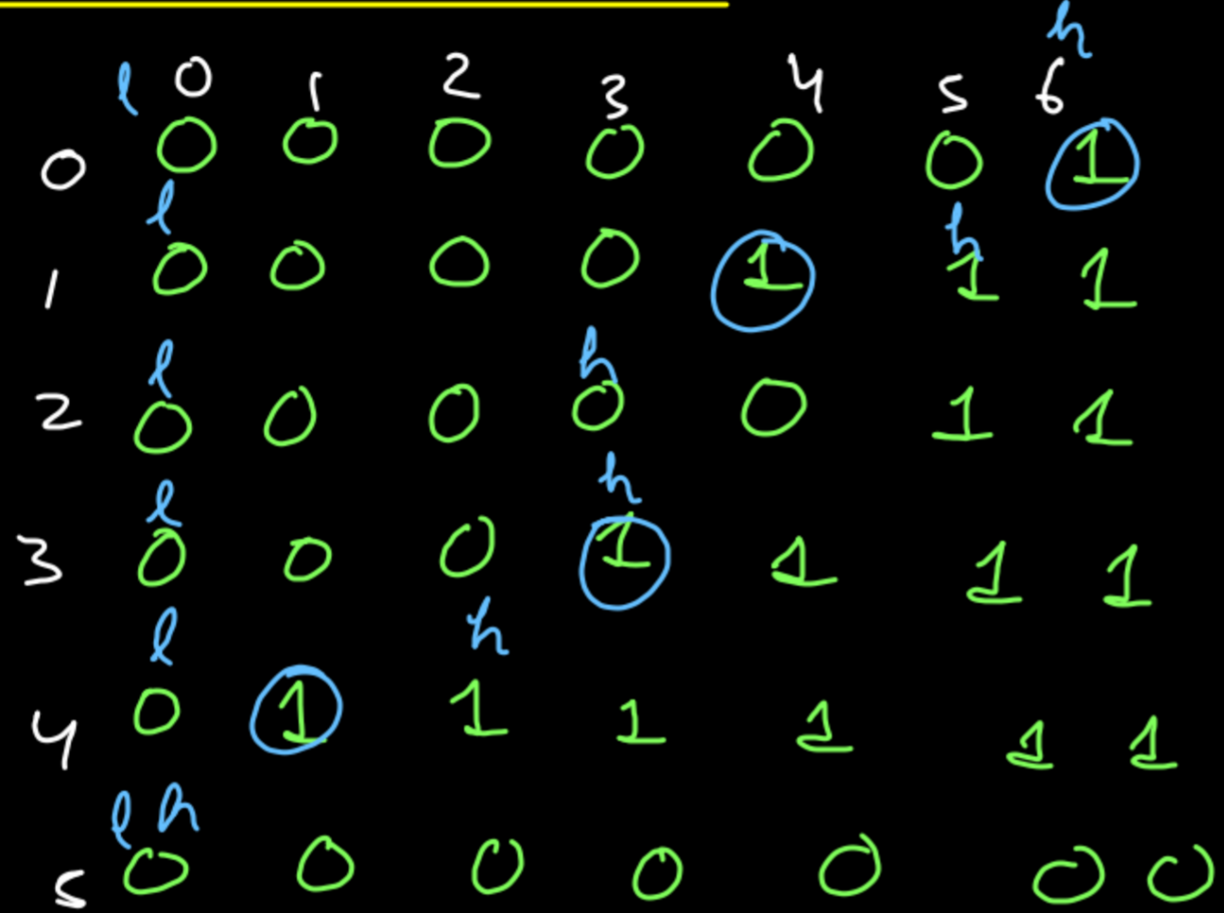


Both conditions did not execute & lo remains less than hi.
This causes an ∞ loop.

{ if (arr[l] < arr[mid]) ∝
  if (arr[mid] < arr[hi]) ∝

Hence, ya to first cond^n me = lagao ya second me
ya donno me varna 1 element k case me TLE aaega.

Row with Maximum 1s



no of ones =
M (no of cols) -
Transition Point
(No of 0s)

Best case : $O(\log_2 m)$

Worst case: $O(n \log(m))$

Code

↓ Normal BS

```
int rowWithMax1s(int arr[][], int n, int m) {
    // code here
    int lo = 0, hi = m-1;

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

↓ Optimized because of this

```
int rowWithMax1s(int arr[][], int n, int m) {
    // code here
    int hi = m-1;

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

    return ans;
}
```

Count Os in a Sorted Matrix

0 0 0 → calculate tp (-1)^3

0 0 1 → calculate tp (2)

0 1 1 → calculate tp (1)

Code

```java
int transitionPoint(int[][] arr,int row,int m) {
    int lo = 0;
    int hi = m-1;

    int pa = m;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        if(arr[row][mid] == 1) {
            pa = mid;
            hi = mid - 1;
        } else {
            lo = mid + 1;
        }
    }

    return pa;

}
```

```java
/*you are required to complete this method*/
int countZeros(int A[][], int N)
{
    int totalZeroes = 0;
    // Your code here
    for(int i=0;i<N;i++) {
        int currZeroes = transitionPoint(A,i,N);
        totalZeroes += currZeroes;
    }

    return totalZeroes;
}
```

Search in a Sorted 2-D Matrix

{ I already know staircase search hence adding other method }

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

target = 10

mid

l h

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

{1, 3, 5, 7, 10, 11, 16, 20, 23, 30, 34, 60}

Row idx -> cell / cols

Col idx -> cell % cols

row = 1

col = 1

arr[row][col]

```java
public boolean searchMatrixHelper(int[][] matrix, int target) {
    int lo = 0;
    int n = matrix.length, m = matrix[0].length;
    int hi = n*m-1;

    while(lo <= hi) {
        int midCell = lo + (hi-lo)/2;

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

        if(matrix[rowIdx][colIdx] == target) return true;
        else if(matrix[rowIdx][colIdx] < target) lo = midCell + 1;
        else hi = midCell - 1;
    }

    return false;
}

public boolean searchMatrix(int[][] matrix, int target) {

    return searchMatrixHelper(matrix,target);
}
```

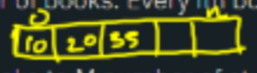
Category:

Binary Search on Answer

++ V Imp

Book Allocation Problem

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



2. You have to allocate 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 minimum of those in all the other permutations and print this minimum value.
3. Each book will be allocated to exactly one student. Each student has to be allocated at least one book.
4. Note: Return -1 if a valid assignment is not possible, and allotment should be in contiguous order.

N books

i^{th} book → A_i no of pages

M students

Minimize the maximum pages.

Ek book ko ek se zyada students nahi padenge. } Aisi koi book nahi hogi jise ek se zyada bache padhe.

Har baccha kamse kam ek book padega. } Ek bhi aisa baccha nahi hoga jo ek bhi book na padhe.

B1 B2 B3 B4
10 20 30 40

Kisi student ko B_1 & B_3 nahi mil sakti. Agar b_1 mili hai to b_2 milegi (contiguous) uske baad b_3 mil sakti hai.

0	1	2	3
20	10	30	40

books = 4

Students = 2

Allocations : (for 2 students)

st1	st2
20 {20	10 30 40 } 80
30 { 20 10	30 40 } 70
60 {20 10 30	40 } 40 ⇒ Best configuration

max load = 60

Allocations : for 3 students

st1	st2	st3
20	10 30	40 ⇒ 40
20 10	30	40 ⇒ 40
20	10	30 40 ⇒ 70

Binary Search on Pages

low
⇓
max no of pages in the array

high
⇓
sum of all the pages of all the books.

{ Kyuki agar kisi ko ye book assign hoti hai to total load usme ya to itna ya is se zyada hi hoga }

Binary Search on Pages

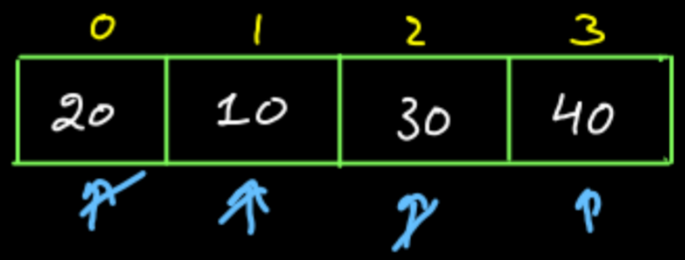
high = 20 + 10 + 30 + 40
= 100

low
40

mid = 70

pa = 70

high
100



st = 2

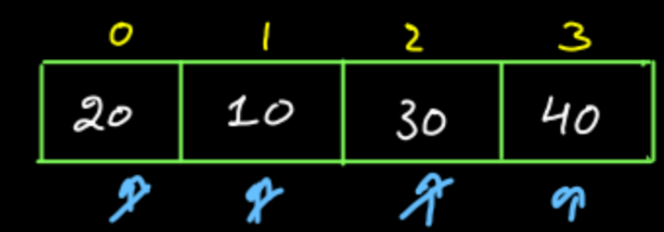
pg = (20 + 10 + 30) + (40)
s1 s2

Since st < 2
This ans is possible

low | high | pa

40 | 69 | 70

mid = 54



St = 1 < 3

pg = (20+10) + (30) + (40)

s1 s2 s3

Ab agar 54 he ans nahi hai to is se chota to kya he ans hoga - That is why lo = mid + 1

↳ (Lalach zyada kar liya)

Since st > 2

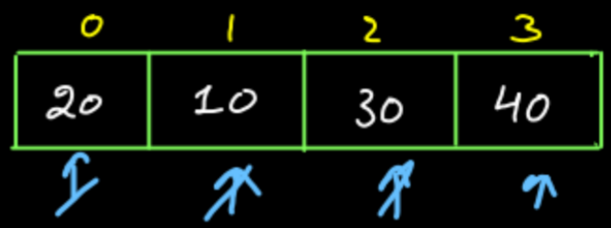
54 cannot be an answer

low | high | pa

55 | 69 | ~~70~~ 62

mid = 62

hi = mid - 1



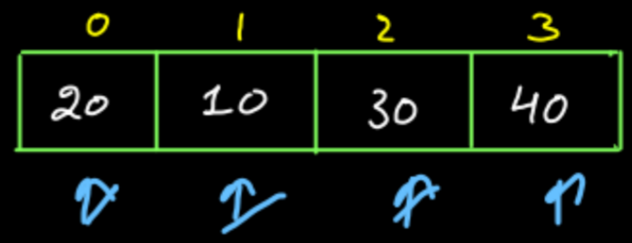
st = ~~1~~ 2

Pg = ∅(20 + 10 + 30) + (40)
S1 S2

Since st ≤ 2

∴ 62 can be an answer.

low	high	pa
55	61	62



mid = 58

st = 1 2 3

Pg = 0.6 (20+10) + (30) + (40)
S1 S2 S3

Zyada lalach kar liya

Since st > 2 So, 58 cannot be an answer.

lo = mid + 1

low
59

high
61

pa
~~62~~
60

mid = 60

hi = mid - 1

low = high = 59

↳ 59 will not be our ans & after this, lo > high - 60 is the final ans.

0	1	2	3
20	10	30	40

st = 1

pg = 0 + (20 + 10 + 30) + (40)
S1 S2

Since st ≤ 2

∴ 60 can be our answer

```java
private static boolean isPossible(int maxLoad,int[] arr, int students) {
  int st = 1;
  int pg = 0;

  for(int i=0;i<arr.length;i++) {
      if(pg + arr[i] <= maxLoad) {
          pg += arr[i];
      } else {
          st++;
          pg = arr[i];
      }
  }

  if(st <= students) return true;
  return false;
}
```

```java
public static int minPages(int[]arr, int m) {
  //write your code here
  int max = Integer.MIN_VALUE;
  int sum = 0;

  for(int i=0;i<arr.length;i++) {
      if(arr[i] > max) max = arr[i];

      sum += arr[i];
  }

  int lo = max;
  int hi = sum;
  int pa = 0;

  while(lo <= hi) {
      int mid = lo + (hi-lo)/2;

      if(isPossible(mid,arr,m) == true) {
          pa = mid;
          hi = mid - 1;
      } else {
          lo = mid + 1;
      }
  }
  return pa;
}
```

Split Array Largest Sum {leetcode: 410} {Hard}

Given an array `nums` which consists of non-negative integers and an integer `m`, you can split the array into `m` non-empty continuous subarrays.

Write an algorithm to minimize the largest sum among these `m` subarrays.

Books

Students

Same as prev ques

0	1	2	3
20	10	30	40

Minimize the max

Pages → this was also the sum of subarrays

```java
public boolean isPossible(int[] nums, int maxSum, int m) {

    int subArrays = 1;
    int sum = 0;

    for(int i=0;i<nums.length;i++) {
        if(sum + nums[i] <= maxSum) {
            sum += nums[i];
        } else {
            sum = nums[i];
            subArrays++;
        }
    }

    if(subArrays <= m) return true;
    return false;
}
```

```java
public int splitArray(int[] arr, int m) {

    int max = Integer.MIN_VALUE;
    int sum = 0;

    for(int i=0;i<arr.length;i++) {
        if(arr[i] > max) max = arr[i];

        sum += arr[i];
    }

    int lo = max;
    int hi = sum;
    int pa = 0;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        if(isPossible(arr,mid,m) == true) {
            pa = mid;
            hi = mid - 1;
        } else {
            lo = mid + 1;
        }
    }
    return pa;
}
```

Capacity to ship packages within D days ${LC : 1011} ${Medium}

1011. Capacity To Ship Packages Within D Days

Medium 3602 80 Add to List Share

A conveyor belt has packages that must be shipped from one port to another within `days` days.

The i^{th} package on the conveyor belt has a weight of `weights[i]`. Each day, we load the ship with packages on the conveyor belt (in the order given by `weights`). We may not load more weight than the maximum weight capacity of the ship.

Return the least weight capacity of the ship that will result in all the packages on the conveyor belt being shipped within `days` days.

Again, same as book allocation Problem.

Exact same code will work.

Painters' Partition

Medium 139 7 Add to favorites

Asked In: GOOGLE CODENATION

Given 2 integers **A** and **B** and an array of integars **C** of size **N**.

Element **C[i]** represents length of **ith** board.

You have to paint all **N** boards **[C0, C1, C2, C3 ... CN-1]**. There are **A** painters available and each of them takes **B** units of time to paint `1 unit` of board.

Calculate and return minimum time required to paint all boards under the constraints that **any painter will only paint contiguous sections of board.**

- 2 painters cannot share a board to paint. That is to say, a board cannot be painted partially by one painter, and partially by another.
- A painter will only paint contiguous boards. Which means a configuration where painter 1 paints board 1 and 3 but not 2 is invalid.

Return the ans % 10000003

Basically the potential ans (pa) will be multiplied by B (time taken for painting unit length)

Total min time = Min (max x Length) * time taken to paint unit length.

```java
public boolean isPossible(int[] nums, int maxSum, int m) {

    int subArrays = 1;
    int sum = 0;

    for(int i=0;i<nums.length;i++) {
        if(sum + nums[i] <= maxSum) {
            sum += nums[i];
        } else {
            sum = nums[i];
            subArrays++;
        }
    }

    if(subArrays <= m) return true;
    return false;
}
```

```java
public class Solution {
    static long m =  10000003;
    public int paint(int A, int B, int[] C) {
        long max = Long.MIN_VALUE;
        long sum = 0;

        for(int i=0;i<C.length;i++) {
            if(C[i] > max) max = C[i];

            sum += C[i];
        }

        long lo = max;
        long hi = sum;
        long pa = 0;

        if(A == C.length) return (int)((B*lo)%m);
        if(A == 1) return (int)((B*hi)%m);

        while(lo <= hi) {
            long mid = lo + (hi-lo)/2;

            if(isPossible(C,(int)mid,A) == true) {
                pa = mid;
                hi = mid - 1;
            } else {
                lo = mid + 1;
            }
        }
        return (int)((B*pa)%m);
    }
```

no of painters = no of boards to paint.

max time will be max length out of all boards * B.

when painter = 1 all boards (sum of all lengths) will be painted by him only.

Koko Eating Bananas

875. Koko Eating Bananas

Medium 3663 167 Add to List Share

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.*

h = 8

k = ?

lo = 1 hi = 11

This is because we are given that ans will always exist

Since it is given that ans will always exist ;
let us take a max speed say 1000 bananas / hr .

1 + 1 + 1 + 1 = 4

0	1	2	3
3	6	7	11

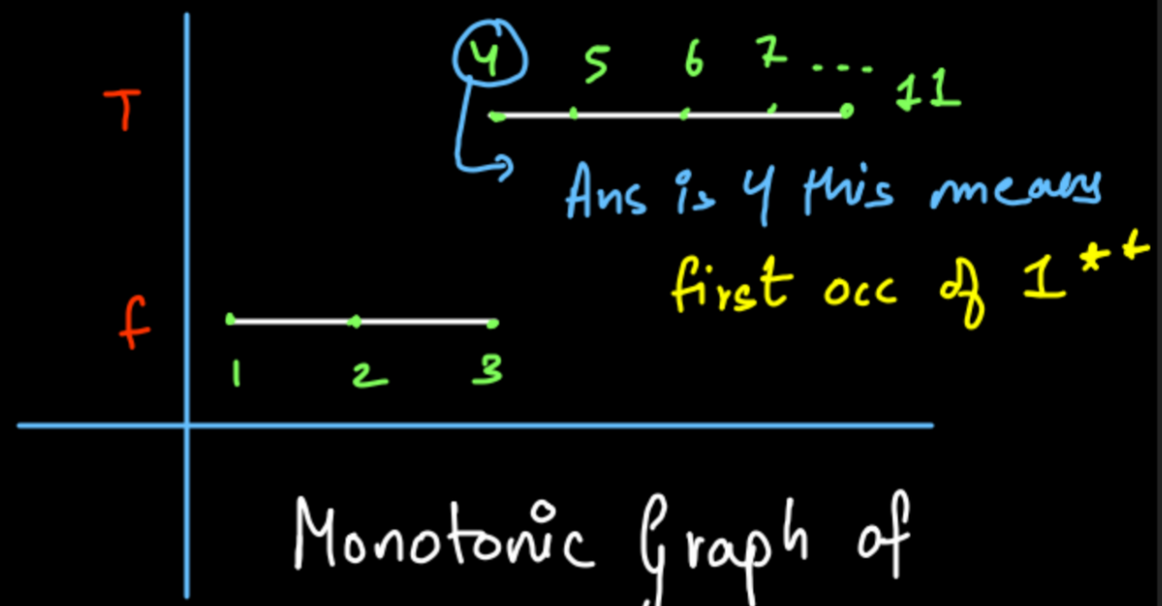
Still, we will take min 4 hours to eat all bananas as there are 4 piles. Hence in the ques, h which is given to us can never be less than the length of the array .

How can we think of BS ?

h = 8h

0	1	2	3
3	6	7	11

11 = 1+1+1+1 = 4 (Y)
10 = 1+1+1+2 = 5 (Y)
9 = 1+1+1+2 = 5 (Y)
8 = 1+1+1+2 = 5 (Y)
7 = 1+1+1+2 = 5 (Y)
6 = 1+1+2+2 = 6 (Y)
5 = 1+2+2+3 = 8 (Y)
4 = 1+2+2+3 = 8 (Y)
3 = 1+2+3+4 = 10 (N)
2 = 2+3+4+6 = 15 (N)
1 = 3+6+7+11 = 27 (N)



Ans is 4 this means first occ of 1**

Monotonic Graph of bananas eating speed.

```java
public boolean isPossible(int midSpeed,int []piles, int h) {

    int currHours = 0;

    for(int i=0;i<piles.length;i++) {
        currHours = currHours + piles[i]/midSpeed;
        if(piles[i] % midSpeed != 0) currHours++;
    }

    if(currHours <= h) return true;
    return false;
}
```

```java
public int minEatingSpeed(int[] piles, int h) {

    int lo = 1;

    int max = Integer.MIN_VALUE;
    for(int i=0;i<piles.length;i++) {
        max = Math.max(piles[i],max);
    }

    int hi = max;
    int pa = piles.length;
    while(lo <= hi) {
        int midSpeed = lo + (hi-lo)/2;

        if(isPossible(midSpeed,piles,h)) {
            pa = midSpeed;
            hi = midSpeed -1;
        } else {
            lo = midSpeed +1;
        }
    }

    return pa; → return paj
```

Smallest Divisor Threshold

1283. Find the Smallest Divisor Given a Threshold

Medium 1209 145 Add to List Share

piles → nums, hours → threshold

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.

Exactly same as Koko Eating Bananas.

→ minSpeed of eating bananas

```java
public int smallestDivisor(int[] nums, int threshold) {
    int lo = 1;

    int max = Integer.MIN_VALUE;
    for(int i=0;i<nums.length;i++) {
        max = Math.max(nums[i],max);
    }

    int hi = max;
    int pa = nums.length;
    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        if(isPossible(mid,nums,threshold)) {
            pa = mid;
            hi = mid -1;
        } else {
            lo = mid +1;
        }
    }

    return pa;
```

```java
public boolean isPossible(int mid,int []nums, int threshold) {

    int curr = 0;

    for(int i=0;i<nums.length;i++) {
        curr = curr + nums[i]/mid;
        if(nums[i] % mid != 0) curr++;
    }

    if(curr <= threshold) return true;
    return false;
}
```

Aggressive Cows

AGGRCOW - Aggressive cows

#binary-search

Farmer John has built a new long barn, with N (2 <= N <= 100,000) stalls. The stalls are located along a straight line at positions x1,...,xN (0 <= xi <= 1,000,000,000).

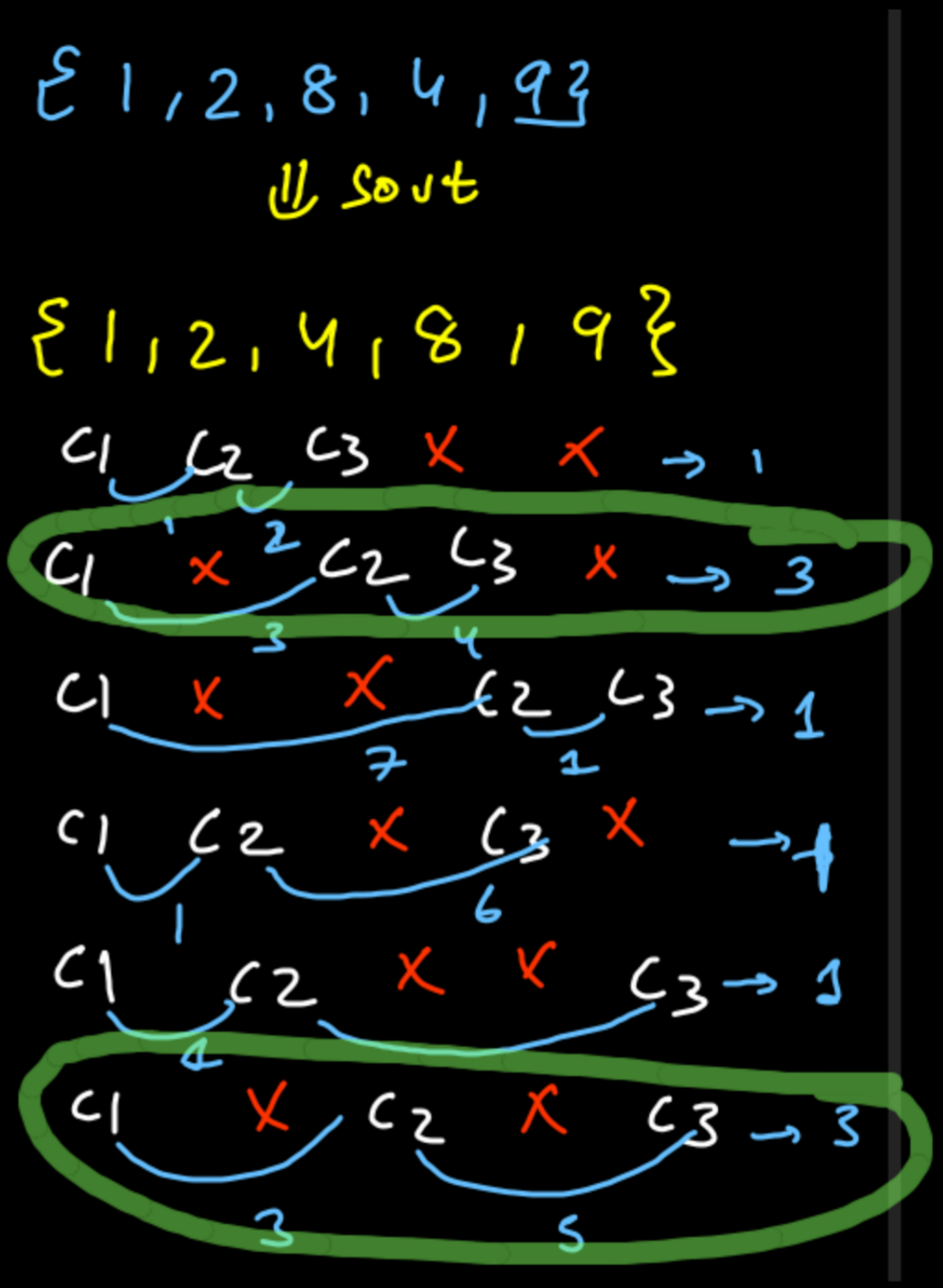
His C (2 <= C <= N) cows don't like this barn layout and become aggressive towards each other once put into a stall. To prevent the cows from hurting each other, FJ wants to assign the cows to the stalls, such that the minimum distance between any two of them is as large as possible. What is the largest minimum distance?

Input

t – the number of test cases, then *t* test cases follows.
* Line 1: Two space-separated integers: N and C
* Lines 2..N+1: Line i+1 contains an integer stall location, xi

Output

For each test case output one integer: the largest minimum distance.



Why first cow is fixed at the first position?

Greedy

1st case

If we keep C1 away from starting it, max dist blw 2 cows is shown here.

2nd case

y will always be greater than x

If C1 is kept at the starting it, max dist blw 2 cows increases.

Binary Search on Adjacent Distance

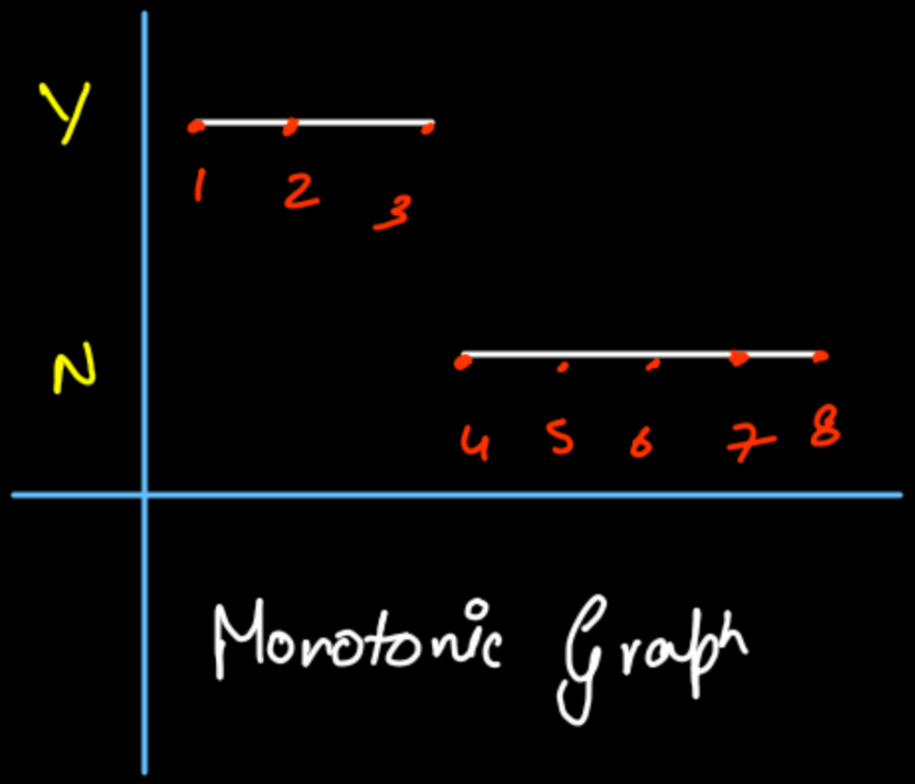
low — high

arr[1] - arr[0] ... arr[n-1] - arr[0]

1 ... 8

{1, 2, 4, 8, 9} Min adj distance

c1 c2



Monotonic Graph

1 → c1 c2 c3 x x Yes

2 → c1 x c2 c3 x x Yes

3 → c1 x c2 c3 x y Yes

4 → No

5 → No

6 → No

7 → No

8 → No

```java
private static boolean isPossible(int[] stalls,int mid, int cows) {
        int placedCows = 1;
        int placedAt = stalls[0];

        for(int i=1;i<stalls.length;i++) {
                if(placedAt + mid <= stalls[i]) {
                        placedAt = stalls[i];
                        placedCows++;
                }
        }

        if(placedCows < cows) return false;
        return true;
}
```

```java
public static int maximizeMinDist(int[] stalls,int cows) {

        Arrays.sort(stalls);

        int lo = stalls[1]-stalls[0];
        int hi = stalls[stalls.length-1] - stalls[0];
        int pa = 0;

        while(lo <= hi) {
                int mid = lo + (hi-lo)/2;

                if(isPossible(stalls,mid,cows)) {
                        pa = mid;
                        lo = mid + 1; //because we have to maximize
                } else {
                        hi = mid-1;
                }
        }
        return pa;
}
```

Category:

Nearly Sorted Array

Search in Nearly Sorted Array {K = 1 for this Ques}

Nearly / k-Sorted array : Har element apni sorted position par ho, ya use k aage ho ya k peeche.

l mid h

20	10	40	30	50	60	80	70	90
0	1	2	3	4	5	6	7	8

target = 70

mid

l l h

20	10	40	30	50	60	80	70	90
0	1	2	3	4	5	6	7	8

$O(\log_2 n * (2k+1))$

$k = 1$

l mid h

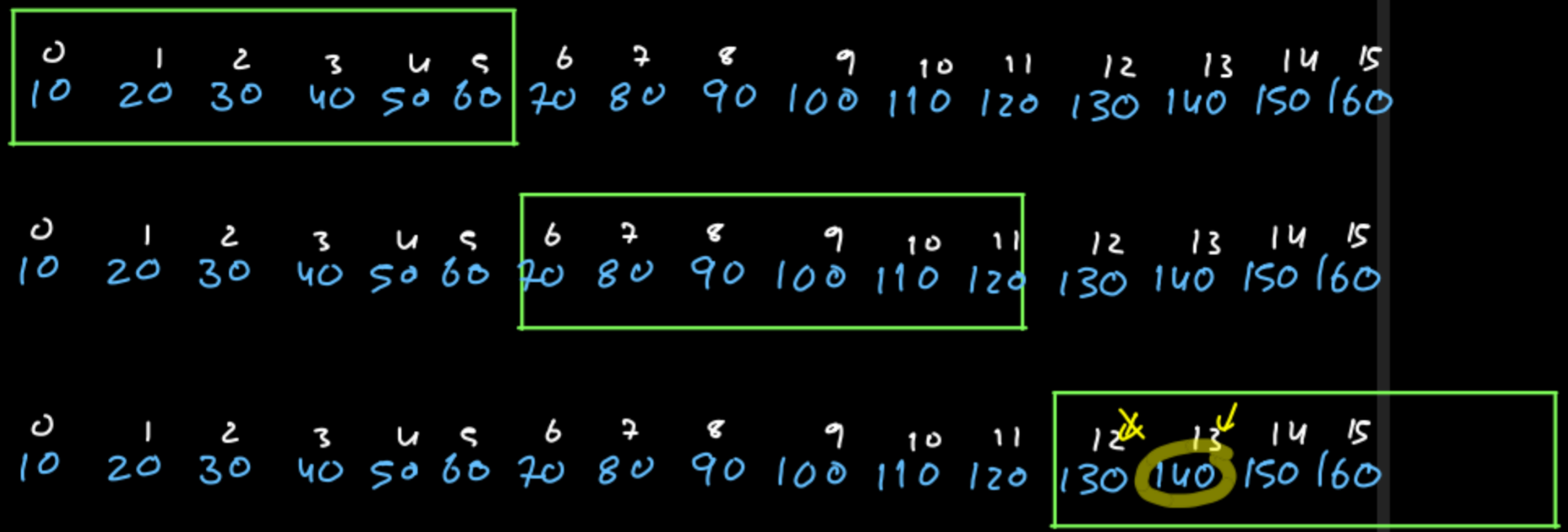
20	10	40	30	50	60	80	70	90
0	1	2	3	4	5	6	7	8

comparing 0 with 3 elements

```java
public static int search(int[] arr,int k,int target) {
    int lo = 0;
    int hi = arr.length-1;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        int lVal = (mid-1 >= 0) ? arr[mid - 1] : Integer.MIN_VALUE;
        int rVal = (mid + 1 < arr.length) ? arr[mid + 1]  : Integer.MAX_VALUE;

        if(arr[mid] == target) return mid;
        else if(lVal == target) return mid - 1;
        else if(rVal == target) return mid + 1;

        if(arr[mid] > target) hi = mid-2;
        else lo = mid + 2;
    }

    return -1;
}
```

Jump Search

target = 140



window of size k keeps on shifting till we get the window where the element could be present. When we reach that window, apply Linear Search.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160

Best Case : first window → first Ele O(1)
→ Last Ele O(K)

Total No of window = N/k

Worst Case : $\frac{N}{k} + k$
↳ Linear Search in a Window

↳ Skipping windows

WC i.e $\frac{N}{k} + k$ will be minimized if $k = \sqrt{N}$

Proof that $k = \sqrt{N}$ will minimize the WC complexity

$$f(k) = \frac{N}{k} + k$$

To find minima

$$\frac{df(k)}{dk} = 0$$

$$\frac{df}{dk} = -\frac{N}{k^2} + 1$$

$$\Rightarrow -\frac{N}{k^2} + 1 = 0$$

$$\Rightarrow +\frac{N}{k^2} = +1$$

$$N = k^2$$

$$k = \pm\sqrt{N}$$

double diff to check whether the one is minima or maxima

$$\frac{d^2f}{dk^2} = \frac{N}{k}$$

$$\left.\frac{d^2f}{dk^2}\right]_{k=\sqrt{N}} = \frac{N}{\sqrt{N}} = \sqrt{N}$$

Since $\frac{d^2f}{dk^2} > 0$ for $k = \sqrt{N}$

$\Rightarrow k = \sqrt{N}$ is the point of minima

Hence Proved

Step Search { Jump Search with variable sized window }

dynamic { size keeps on changing }

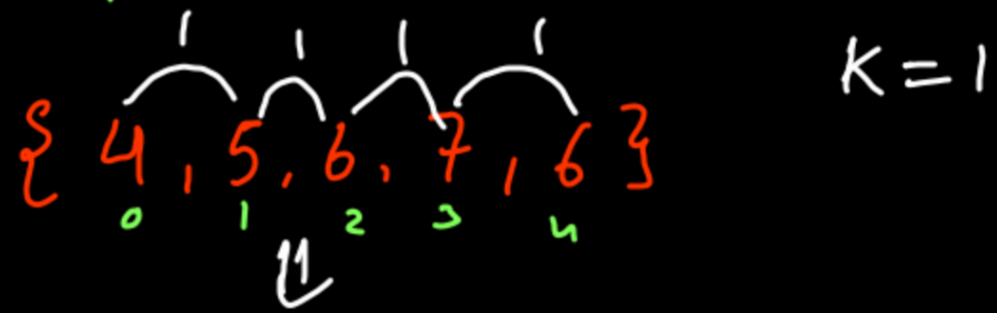
Searching in an array where adjacent differ by at most k

Easy Accuracy: **71.96%** Submissions: **5434**

Points: **2**

A step array is an array of integer where each element has a difference of at most k with its neighbor. Given a key x, we need to find the index value of x if multiple elements exist, return the first occurrence of the key.

{ Neighbors will have atmost k difference blw each other }



We can clearly see that array may or may not be sorted.

Example 1

$k = 3$

target = 28

{ 10, 12, 15, 16, 19, 21, 23, 25, 28 }

0 1 2 3 4 5 6 7 8

$\frac{abs\{target - arr[i]\}}{k}$ = jump size

$\frac{28 - 10}{3} = \frac{18}{3} = 6$

$i = i + 6$

{ 10, 12, 15, 16, 19, 21, 23, 25, 28 }

0 1 2 3 4 5 6 7 8

$\frac{28 - 23}{3} = \frac{5}{3} = 1$

{ 10, 12, 15, 16, 19, 21, 23, 25, 28 }

0 1 2 3 4 5 6 7 8

$\frac{28 - 25}{3} = \frac{3}{3} = 1$

Example: 2

k = 3
target = 24

{15, 13, 16, 14, 17, 20, 19, 22, 24, 23}

$\frac{24-15}{3} = \frac{9}{3} = 3$

{15, 13, 16, 14, 17, 20, 19, 22, 24, 23}

$\frac{24-14}{3} = \frac{10}{3} = 3$

{15, 13, 16, 14, 17, 20, 19, 22, 24, 23}

$\frac{24-19}{3} = \frac{5}{3} = 1$

{15, 13, 16, 14, 17, 20, 19, 22, 24, 23}

$\frac{24-22}{3} = \frac{2}{3} = 0$

Infinite Loop

{15, 13, 16, 14, 17, 20, 19, 22, 24, 23}

if Jump Size == 0
Jump Size ++;

```java
// Function for finding maximum and value pair
public static int search (int arr[], int n, int x, int k) {
    //Complete the function

    int idx = 0;

    while(idx < arr.length) {
        if(arr[idx] == x) return idx;

        int jumpSize = ((int)Math.abs(x-arr[idx]))/k;
        if(jumpSize == 0) jumpSize++;

        idx += jumpSize;
    }

    return -1;
}
```

WC : O(N)

Avg Case : O(√N)

Best Case : O(1)

Category: Infinite Array

{Unbounded Binary Search}

Unbounded Binary Search {Infinite Sorted Array}

{kind of fenwick Tree, sqrt decomposition, binary lifting}

tar = 28

{ l h
{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, ... ∞ }

{ 1, 2, l 3, 4 h, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, ... ∞ }

{ 1, 2, 3, 4, l 5, 6, 7, 8 h, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, ... ∞ }

{ 1, 2, 3, 4, 5, 6, 7, 8, l 9, 10, 11, 12, 13, 14, 15, 16 h, 17, 18, 19, 20, 21, 22, ... ∞ }

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, l 17, 18, 19, 20, 21, 22, ... 32 h ∞ }

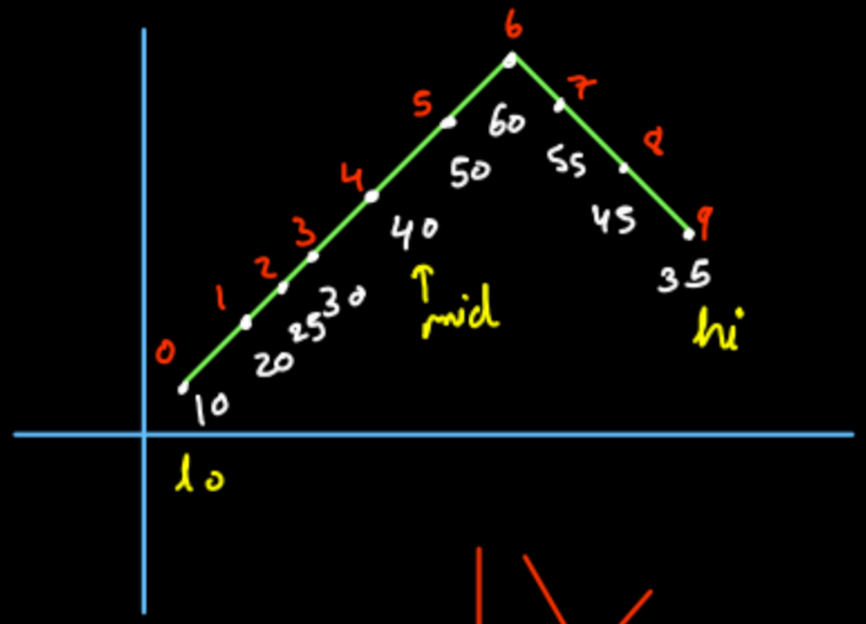
Apply Bin Search

```java
public static int binarySearch(int[] arr, int lo, int hi, int target) {

    if(lo > hi) return -1; //unsuccessful search

    int mid = lo +(hi-lo)/2;

    if(arr[mid] == target) return mid;
    else if(arr[mid] > target) return binarySearch(arr,lo,mid - 1,target);
    else return binarySearch(arr,mid + 1,hi,target);
}

//This method will be invoked as binarySearchInfinite(arr,0,1,target)

public static binarySearchInfinite(int[] arr, int lo, int hi, int target) {

    if(lo > hi) return -1; //this line will never execute as lo will never be less than high in case of unsuccessful search

    if(target >= arr[lo] && target <= arr[hi]) {
        return binarySearch(arr,lo,hi,target);
    } else {
        binarySearchInfinite(arr,hi + 1,hi * 2,target);
    }
}
```

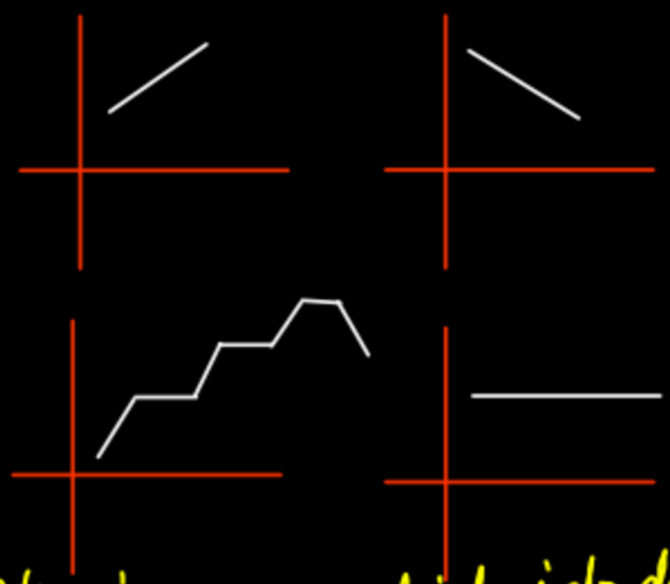
Category : Bitonic Array

Bitonic Array {Peak Element}

↳ Brute force : Linear Search (O(N))

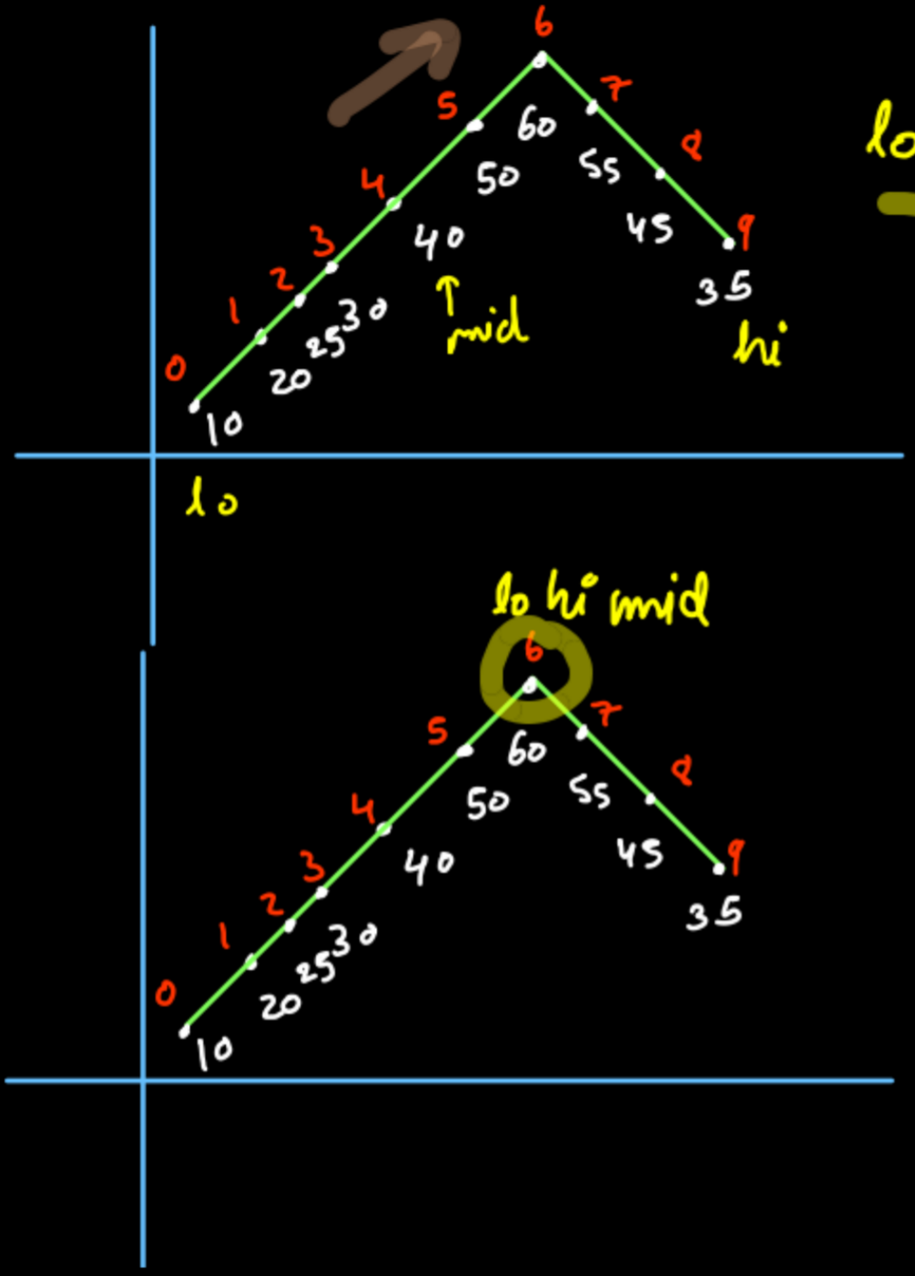


Some bitonic Arrays

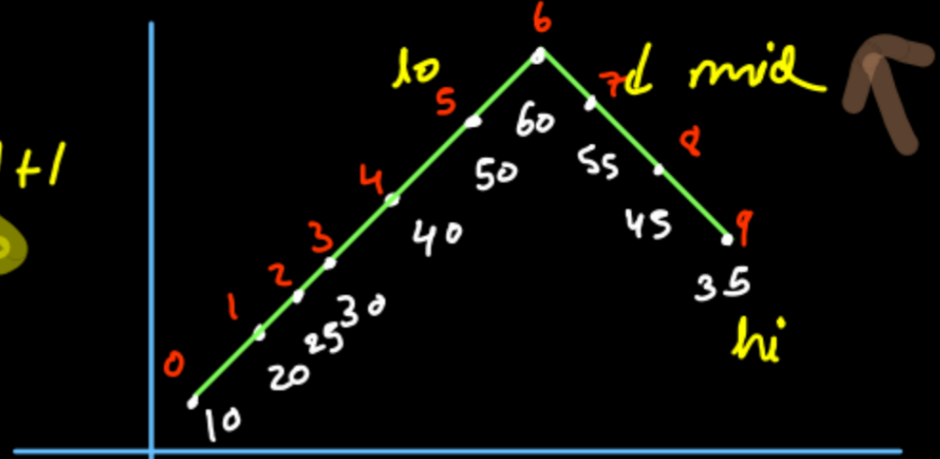


This is not bitonic arrays

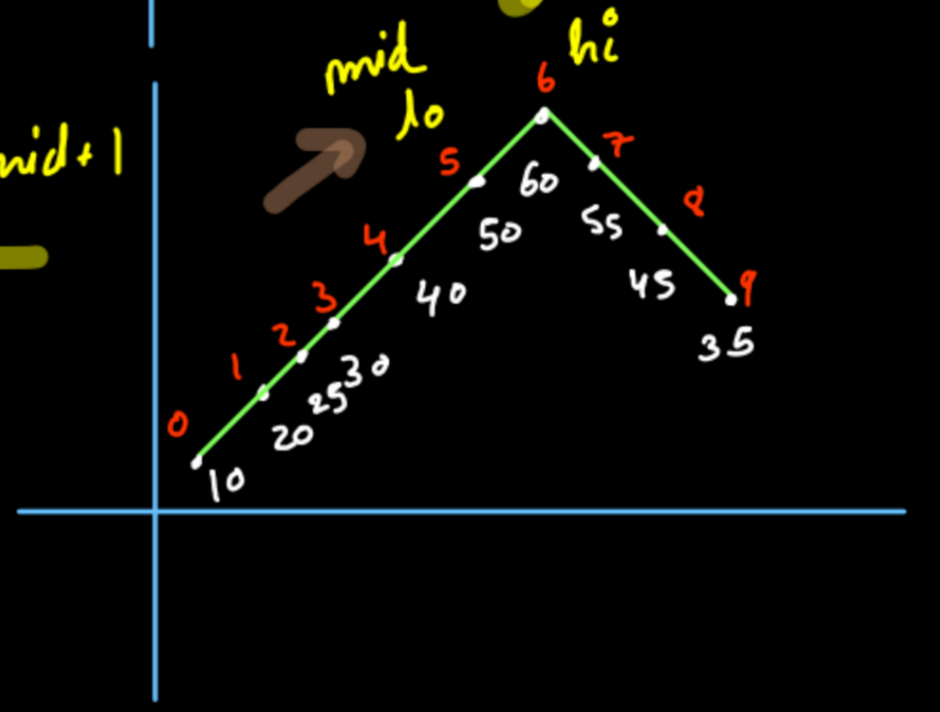
Bitonic array vahi thai do decrease hone R baad kabhi increase na kare.



lo = mid + 1

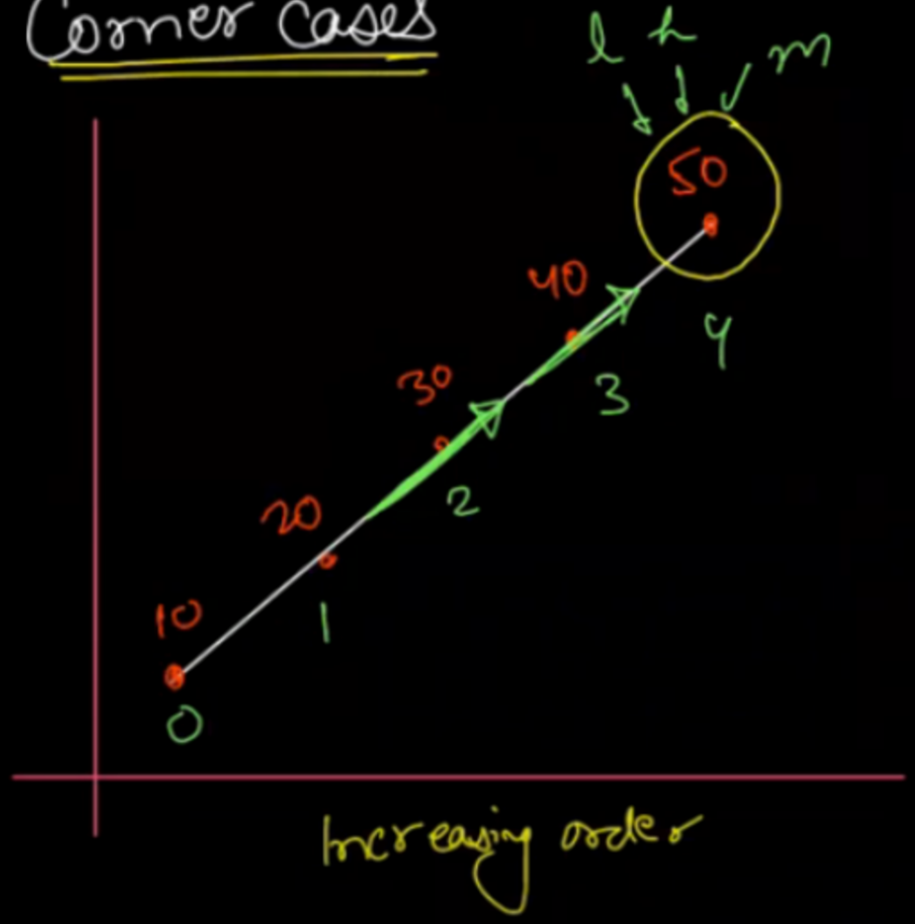


hi = mid - 1

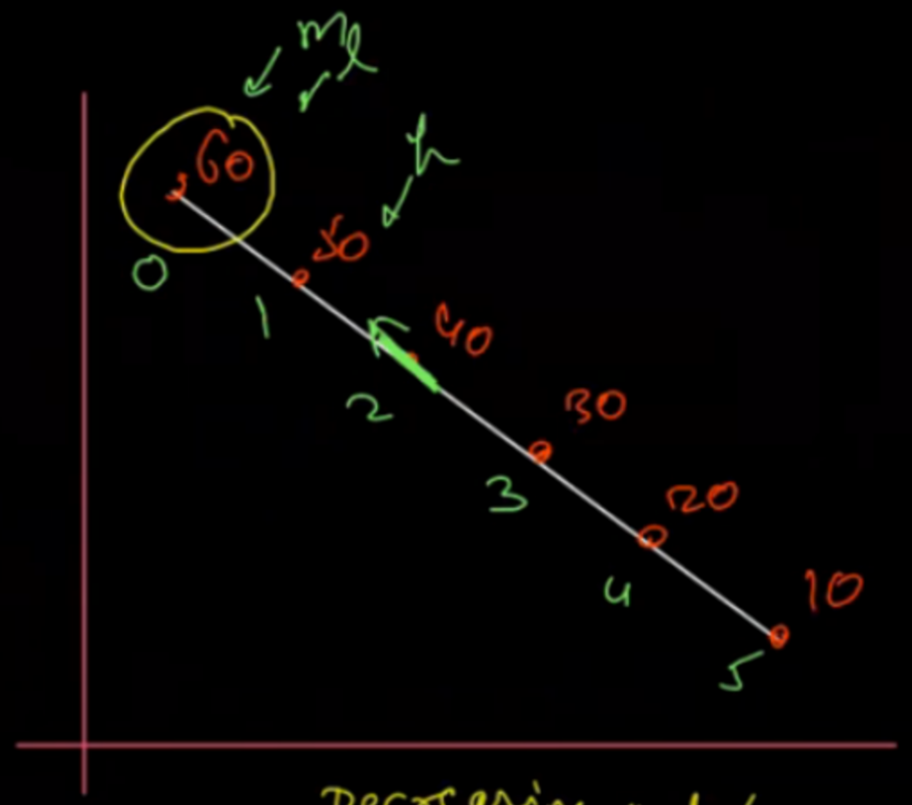


lo = mid + 1

Corner cases



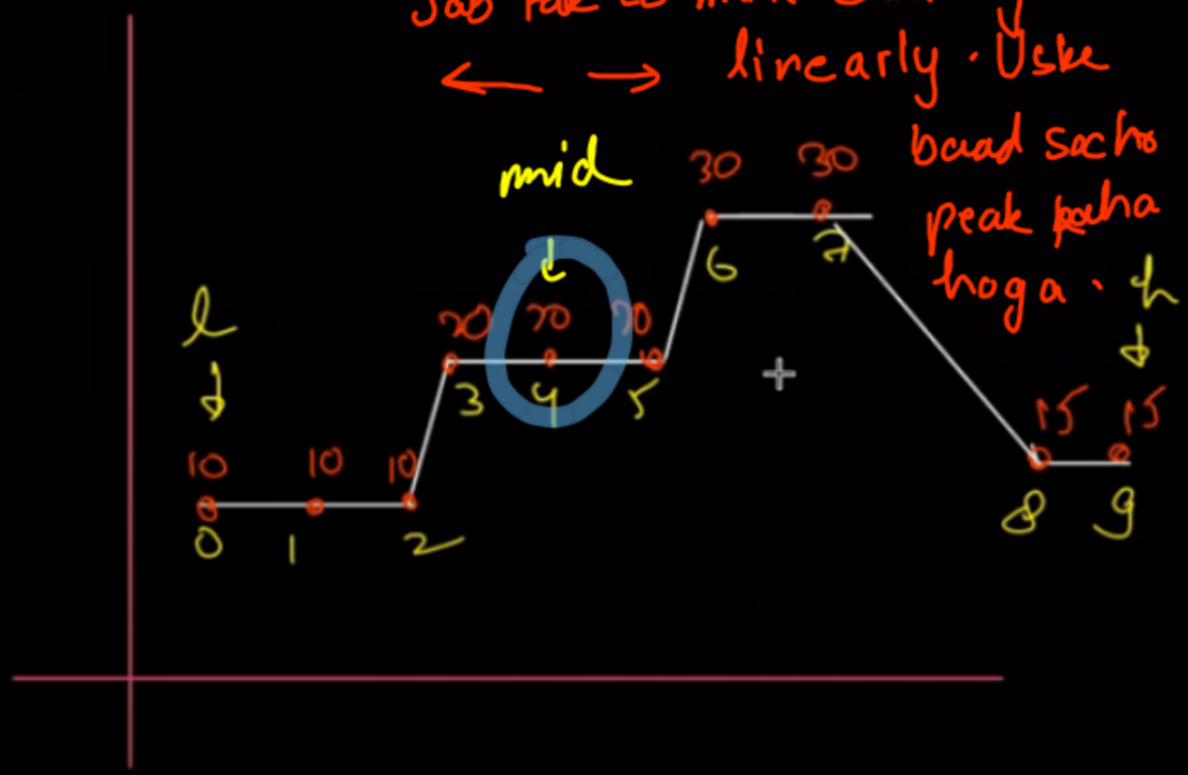
Increasing order



Decreasing order

```java
public int peakIndexInMountainArray(int[] arr) {
    int lo = 0;
    int hi = arr.length-1;

    while(lo <= hi) {
        int mid = lo +(hi-lo)/2;

        int lVal = mid - 1 >=0 ? arr[mid-1] : Integer.MIN_VALUE;
        int rVal = mid + 1 < arr.length ? arr[mid + 1] : Integer.MAX_VALUE;

        if(arr[mid] >lVal && arr[mid] > rVal) {
            return mid;
        } else if(arr[mid + 1] > arr[mid]) {
            lo = mid +1;
        } else if(arr[mid + 1] < arr[mid]) {
            hi = mid - 1;
        }
    }

    return -1;
}
```

Our code will fail in case of duplicate elements.



Left aur right, done
me he equal elements
hai. So, increasing
nature kis taraf hai
ye pata he nahi
chalega. So, we can't
decide which side will
be the peek.

```java
public int peakIndexInMountainArray(int[] arr) {
    int lo = 0;
    int hi = arr.length-1;

    while(lo <= hi) {
        int mid = lo +(hi-lo)/2;

        int lVal = mid - 1 >=0 ? arr[mid-1] : Integer.MIN_VALUE;
        int rVal = mid + 1 < arr.length ? arr[mid + 1] : Integer.MIN_VALUE;

        if(arr[mid] >lVal && arr[mid] > rVal) {
            return mid;
        } else if(rVal > arr[mid] && arr[mid] > lVal) {
            lo = mid +1;
        } else if(rVal < arr[mid] && arr[mid] < lVal)  {
            hi = mid - 1;
        } else {
            //O(n) worst case
            if(arr[lo] <= arr[hi]) lo++;
            else hi--;
        }
    }

    return -1;
}
```

Handled Duplicates

O(N) → TC

find Peak Element

162. Find Peak Element

Medium 5192 3465 Add to List Share

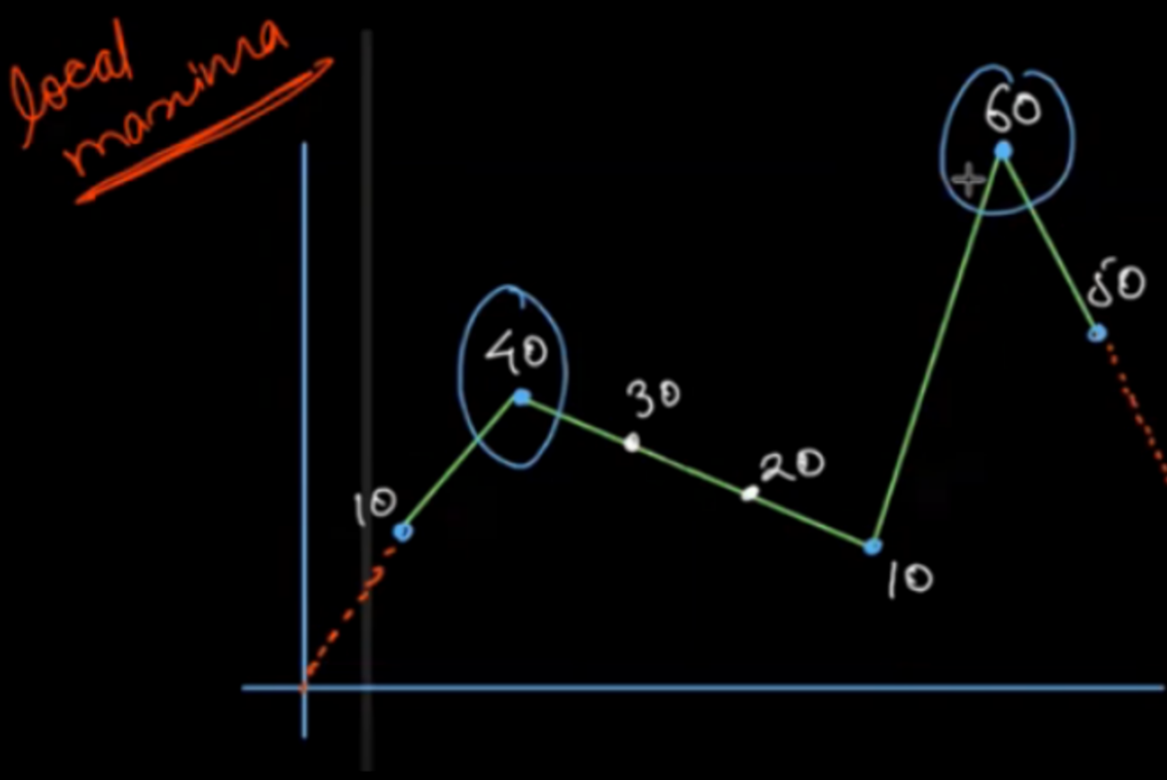
A peak element is an element that is strictly greater than its neighbors.

Given an integer array `nums`, find a peak element, and return its index. If the array contains multiple peaks, return the index to **any of the peaks**.

You may imagine that `nums[-1] = nums[n] = -∞`.

You must write an algorithm that runs in `O(log n)` time.

They are asking for a local maxima. Global maxima is nothing but the max value in the array & it is not possible to find out the max value in an array in O(logn) time.



$$\{\underset{0}{10}, \underset{1}{40}, \underset{2}{30}, \underset{3}{20}, \underset{4}{10}, \underset{5}{60}, \underset{6}{50}\}$$

l — mid — h

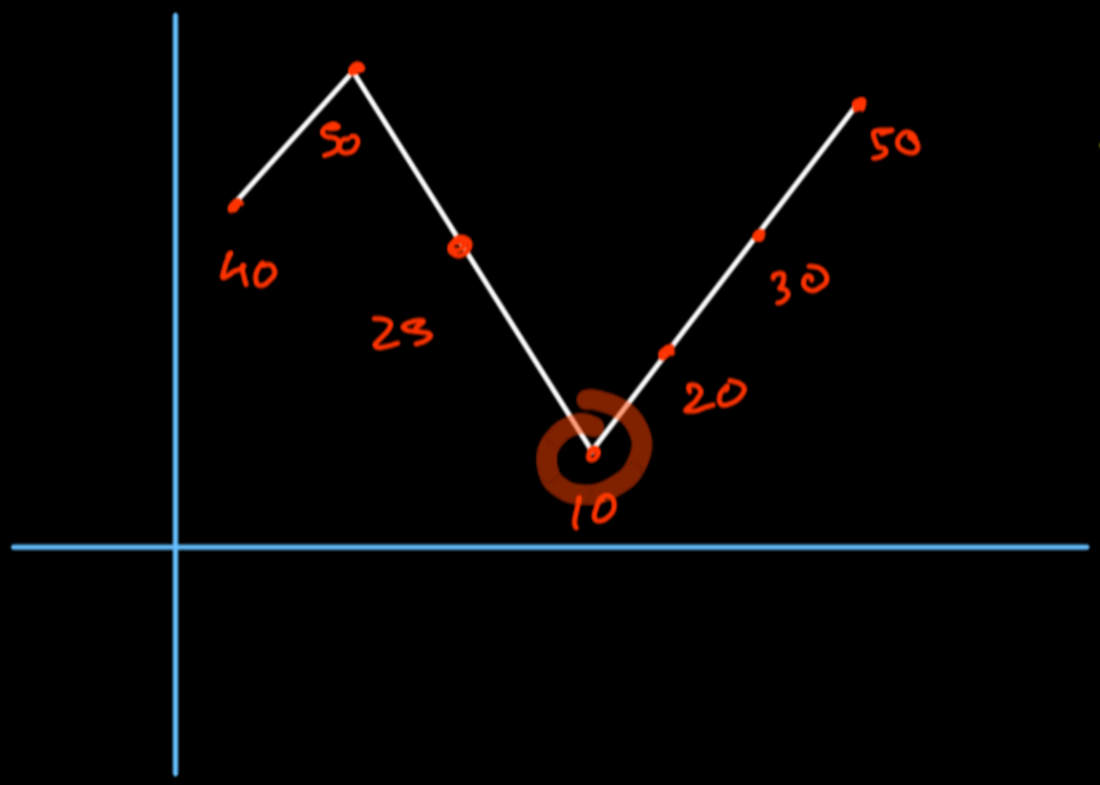
Since arr[mid-1] > arr[mid], so we can say that there will definitely be a peak there. So, we move to the left.

$$\{\underset{0}{10}, \underset{1}{40}, \underset{2}{30}, \underset{3}{20}, \underset{4}{10}, \underset{5}{60}, \underset{6}{50}\}$$

l mid h

⇒ peak found ⇒ local maxima

Special Case: Mid on Valley



l mid h

{40, 50, 25, 10, 20, 30, 50}

0 1 2 3 4 5 1

⇒ ∵ arr[mid-1] > arr[mid]
& arr[mid+1] > arr[mid]
peak will be on both the sides. So, we can go on either side.

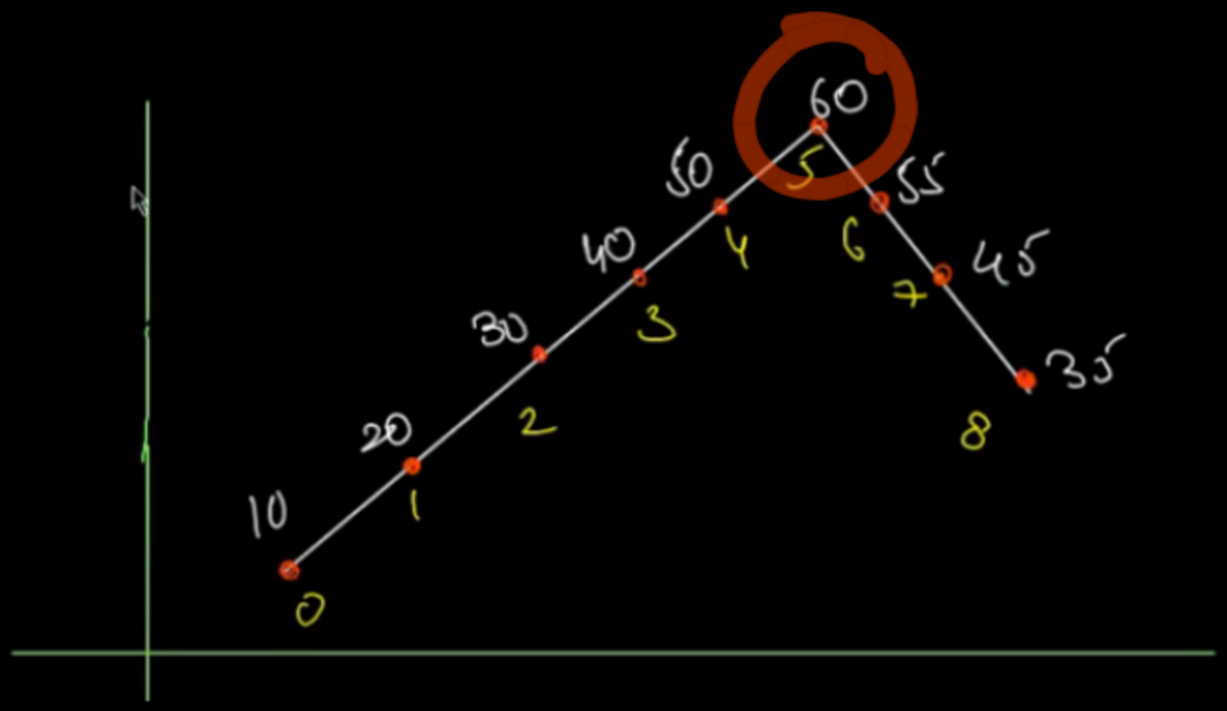
A4 Min one peak element will surely be present because of the condition that arr[-1] = arr[n] = $-\infty$.

=> We will need to keep arr[-1] = arr[n] = $-\infty$ of Long i.e Long.MIN_VALUE because arr is an array of Integers & it can also contain Integer.MIN_VALUE.

Code for Peak Element

```java
public int findPeakElement(int[] arr) {

    int lo = 0;
    int hi = arr.length - 1;

    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;

        long lVal = mid -1 >= 0 ? arr[mid-1] : Long.MIN_VALUE;
        long rVal = mid + 1 < arr.length ? arr[mid + 1] : Long.MIN_VALUE;

        if(arr[mid] > lVal && arr[mid] > rVal) return mid;
        else if(arr[mid] < arr[mid + 1]) lo = mid +1;
        else hi = mid - 1;
    }

    return -1;
}
```

Search in Bitonic Array



⇒ find the peak element
& if it is our ans, return peakIdx. If our target is greater than peak, can't find it else left & right both are sorted, apply bin search on each of them.

Code

$TC = 3\log_2 N$

```java
public int peakIndexInMountainArray(int[] arr) {
    int lo = 0;
    int hi = arr.length-1;

    while(lo <= hi) {
        int mid = lo +(hi-lo)/2;

        int lVal = mid - 1 >=0 ? arr[mid-1] : Integer.MIN_VALUE;
        int rVal = mid + 1 < arr.length ? arr[mid + 1] : Integer.MIN_VALUE;

        if(arr[mid] >lVal && arr[mid] > rVal) {
            return mid;
        } else if(rVal > arr[mid] && arr[mid] > lVal) {
            lo = mid +1;
        } else if(rVal < arr[mid] && arr[mid] < lVal)  {
            hi = mid - 1;
        } else {
            //O(n) worst case
            if(arr[lo] <= arr[hi]) lo++;
            else hi--;
        }
    }

    return -1;
}
```

```java
public int binarySearch(int[] arr, int lo, int hi, int target, boolean isInc) {

   while( lo <= hi) {

        int mid = lo + (hi-lo)/2;

        if(arr[mid] == target) return mid;
        else if(isInc == true) {
            if(arr[mid] > target) hi = mid - 1;
            else lo = mid + 1;
        } else {
            if(arr[mid] > target) lo = mid + 1;
            else hi = mid -1;
        }
   }


    return -1;
}
```

```java
   public int solve(int[] A, int B) {
       int peakIdx = peakIndexInMountainArray(A);
       if(A[peakIdx] == B) return peakIdx;
       else if(B > A[peakIdx]) return -1;
       int left = binarySearch(A,0,peakIdx-1,B,true);
       if(left != -1) return left;
       int right = binarySearch(A,peakIdx + 1,A.length-1,B,false);
       return right;
   }
```

Extra Practice Questions

Unique Element in a Sorted Array

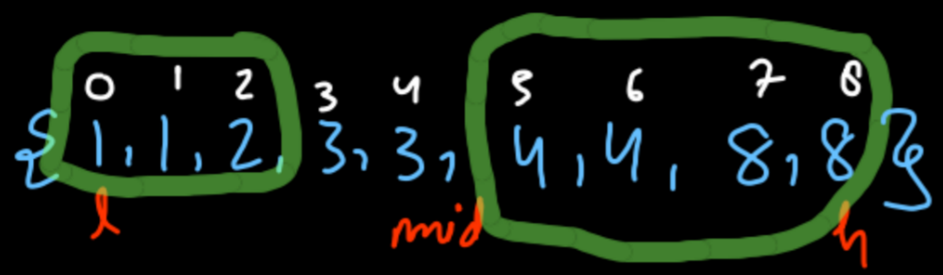
{1, 1, 2, 3, 3, 4, 4, 8, 8}

Approach: 1 {Using XOR} TC = O(N)

xor = 1 ^ 1 ^ 2 ^ 3 ^ 3 ^ 4 ^ 4 ^ 8 ^ 8

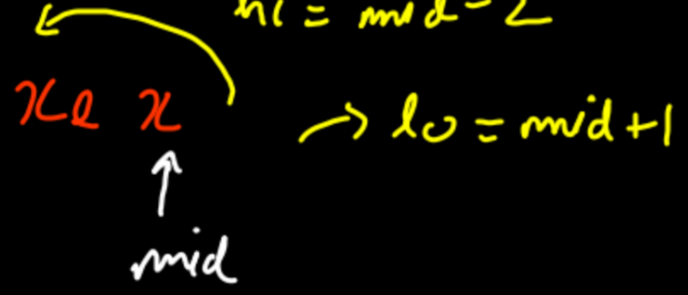
= 0 ^ 2 ^ 0 ^ 0 ^ 0 = 2

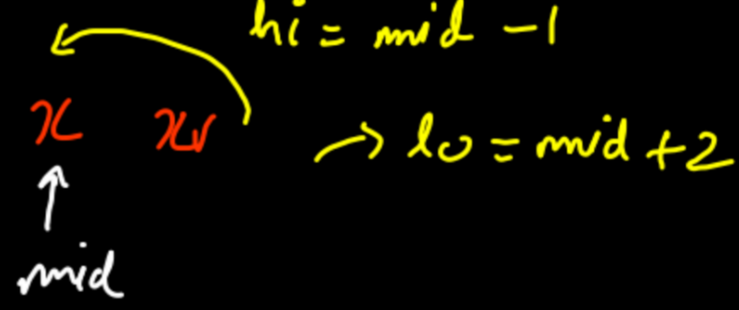
Since the given array is sorted, we can apply BS.



$TC = O(\log_2 N)$

In which ever part there are odd no of elements, unique element will be present there.





When $x_l = x$

Left Range : mid - lo - 1

Right Range : hi - mid

When $x = x_l$

Left Range = mid - lo

Right Range = hi - mid - 1

Code

```java
public int singleNonDuplicate(int[] nums) {
    int lo = 0, hi = nums.length-1;
    while(lo <= hi) {
        int mid = lo + (hi-lo)/2;
        int lVal = mid - 1 >= 0 ? nums[mid - 1] : -1;
        int rVal = mid + 1 < nums.length ? nums[mid + 1] : -1;

        //this is the unique element
        if(nums[mid] != lVal && nums[mid] != rVal) {
            return nums[mid];
        }

        if(nums[mid] == lVal) {
            int lCount = mid - lo - 1;
            if(lCount % 2 == 1) {
                hi = mid - 2;
            } else {
                lo = mid + 1;
            }
        } else {
            int rCount = hi - mid - 1;
            if(rCount % 2 == 1) {
                lo = mid + 2;
            } else {
                hi = mid -1;
            }
        }
    }

    return -1;
}
```

Missing Element in AP

0	1	2	3	4	5	6	7	8
2	4	8	10	12	14	16	18	20

0 1 2 3 4 5 6 7 8 (n-1)

a a+d a+2d . . . . . a+(n-1)d

⇒ At i^{th} index, The AP term should be a+id

Solve using Binary Search

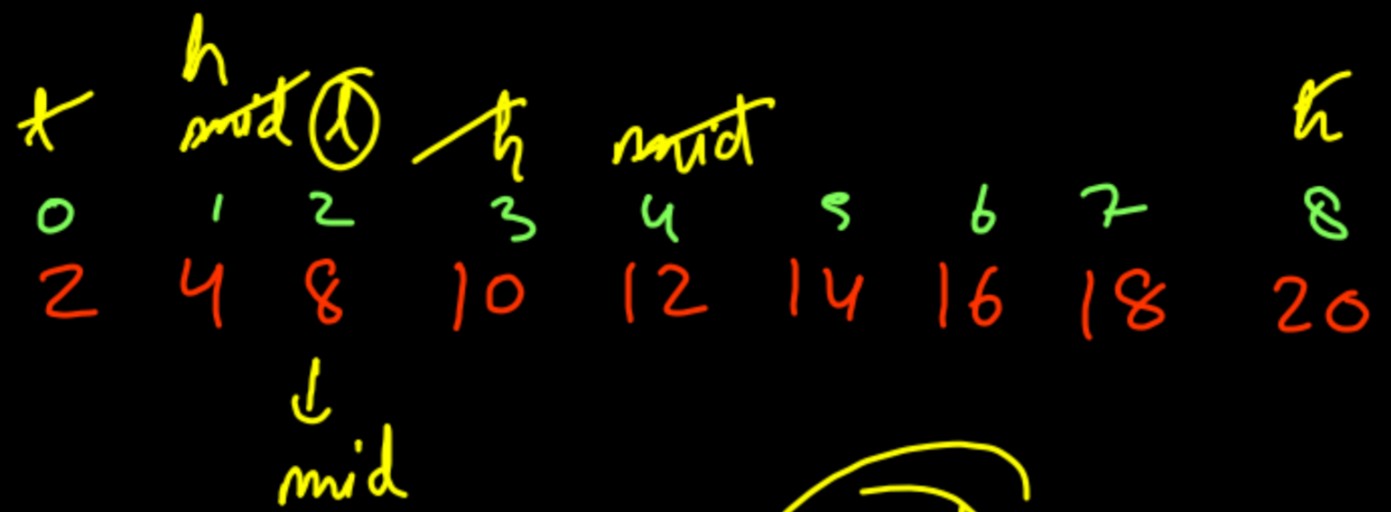
l				mid				h
0	1	2	3	4	5	6	7	8
2	4	8	10	12	14	16	18	20

if a+4*d = 2+4*2 = 10 == arr[mid]

Since arr[mid] > 10 (a+id)

h = mid - 1

else l = mid - 1



To calculate d, the last term should be = a + nd

$$20 = 2 + 8 * d$$

$$18 = 8d \Rightarrow \boxed{d = 2}$$

```java
class Solution {
    int findMissing(int[] arr, int n) {
        // code here
        int lo = 0;
        int hi = n-1;

        int a = arr[0];
        int d = (arr[n-1] - arr[0]) / n;

        while(lo <= hi) {
            int mid = lo + (hi-lo)/2;

            if((a + mid*d) == arr[mid]) {
                lo = mid + 1;
            } else {
                hi = mid - 1;
            }
        }

        return a+lo*d;
    }
}
```

Category: MEDIAN

Median of 2 Sorted Arrays {Same Size}

al am ah

{ a_0 a_1 a_2 a_3 a_4 }

{ b_0 b_1 b_2 b_3 b_4 }

bl bm bh

am = bm {$\frac{am+bm}{2}$ or am or bm}

am < bm discard left in a[] & right in b[]

am > bm discard right in a[] & left in b[]

(1) Brute force :- Merge

2 6 7 9 10 (N)

1 3 5 8 8 (N)

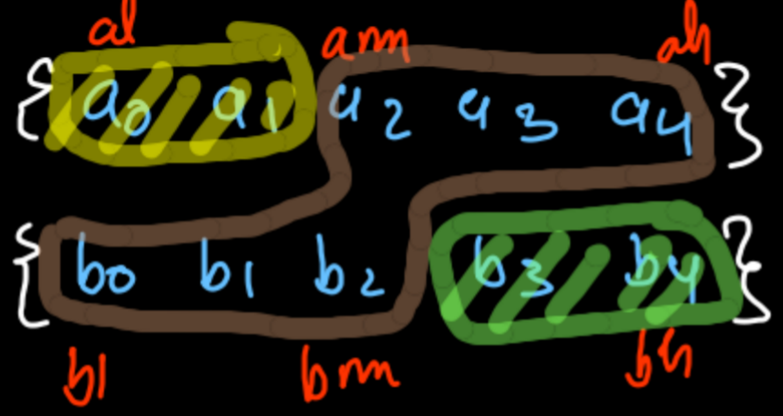
1 2 3 5 6 7 8 8 9 10 (2N)

$\frac{6+7}{2} = 6.5$

Time: $O(2n) + O(1) = O(2n)$

Space: $O(2n)$

$a_m < b_m$ Case



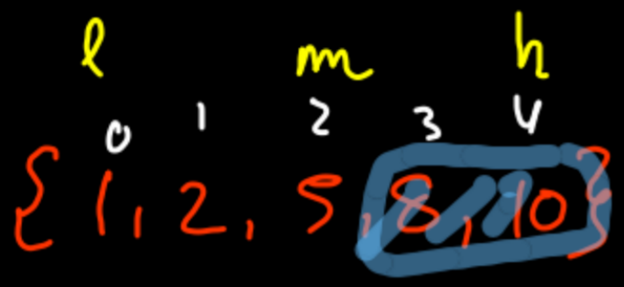
$a_m < b_m$

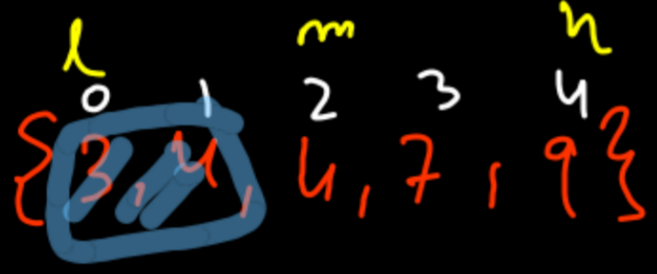
a_0 and a_1 are less than a_m and b_3 & b_4 are greater than b_m

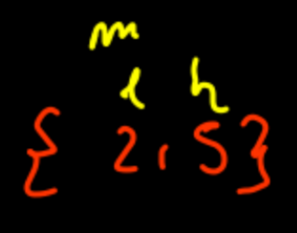
Only $b_0 b_1$ & $a_3 a_4$ can lie b/w a_m & b_m

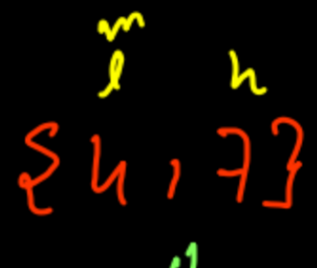
hence, median will be among these elements.

Example Dry Run

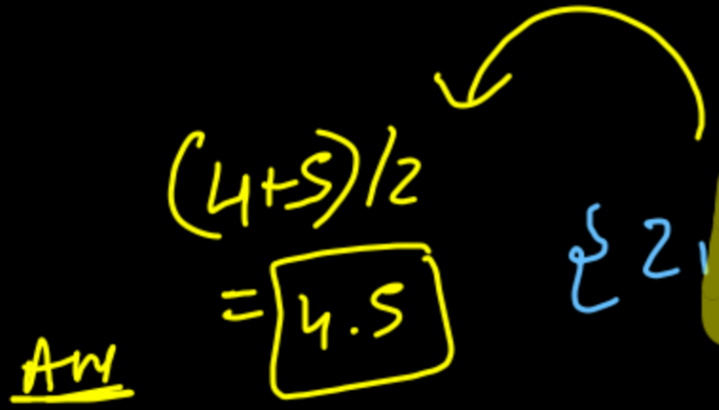




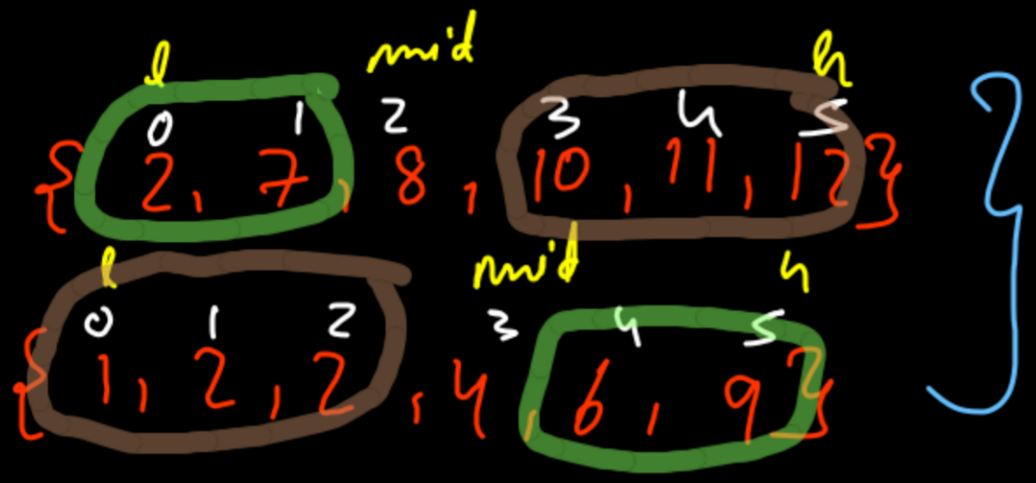




When only 2 elements in each array are left, sort them & find the median



Case of Even Arrays



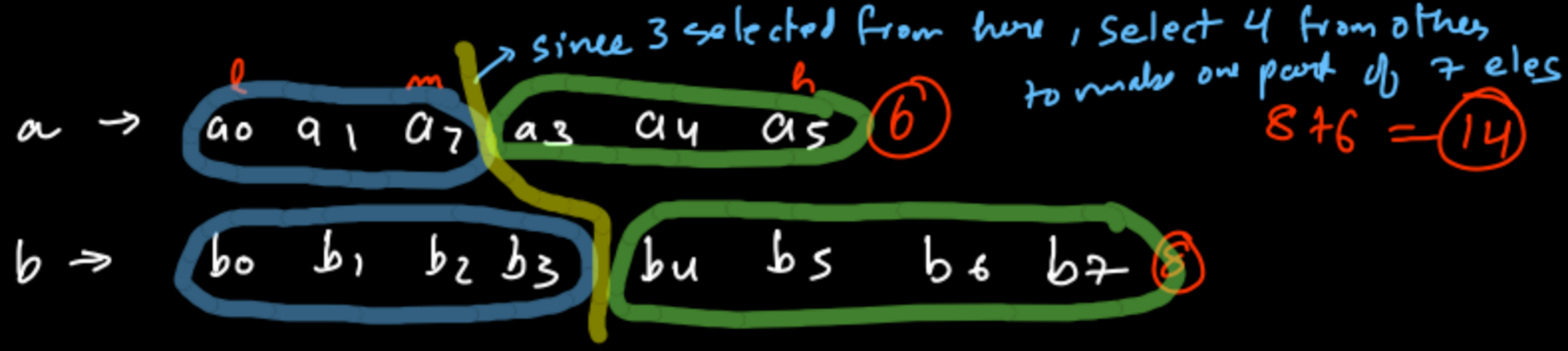
In case of even sized array take floor as the mid for one array & ceil for other to have equal elements to discard on either side.

{0+5}/2 → 2 → arr a mid

→ 3 → arr b mid

{ Refer to code from GfG }

Median of 2 Sorted Arrays {Unequal Size} {Electrode Hard}

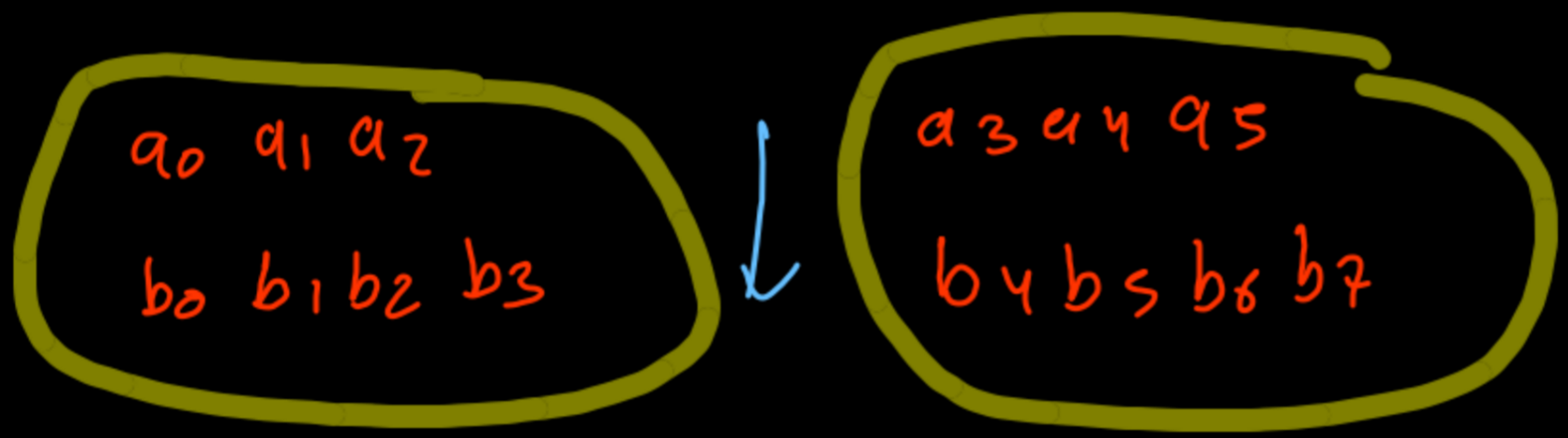


Sorted Array {contains elements of a & b only}

To find median, the gist is to divide the array in 2 equal parts and the largest ele of first part & smallest of second part will contribute to the median. & In case of even size of the overall array

$a \rightarrow a_0\ a_1\ a_2\ a_3\ a_4\ a_5$

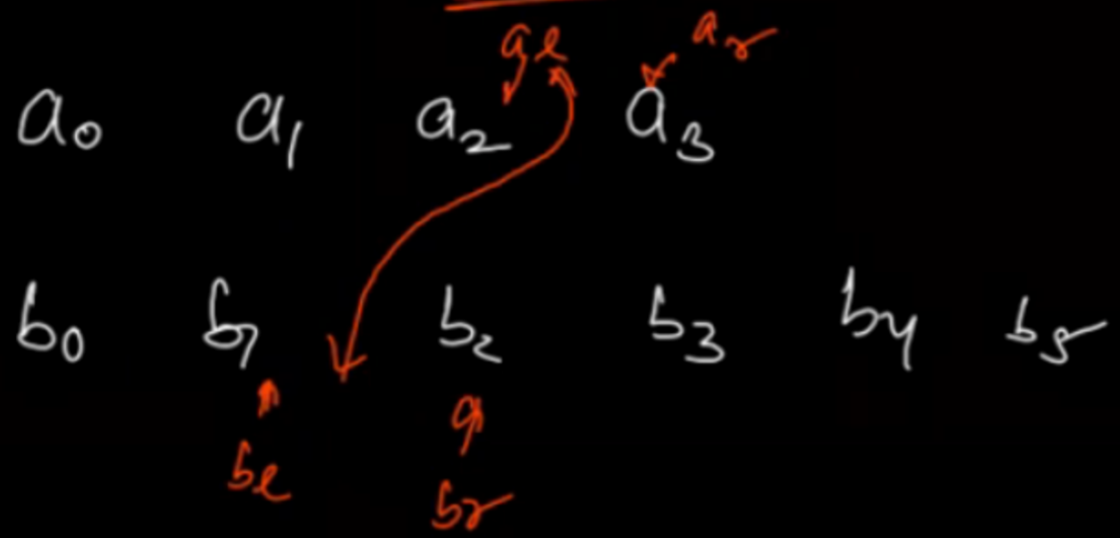
$b \rightarrow b_0\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7$



$a_0 \leq b_4$

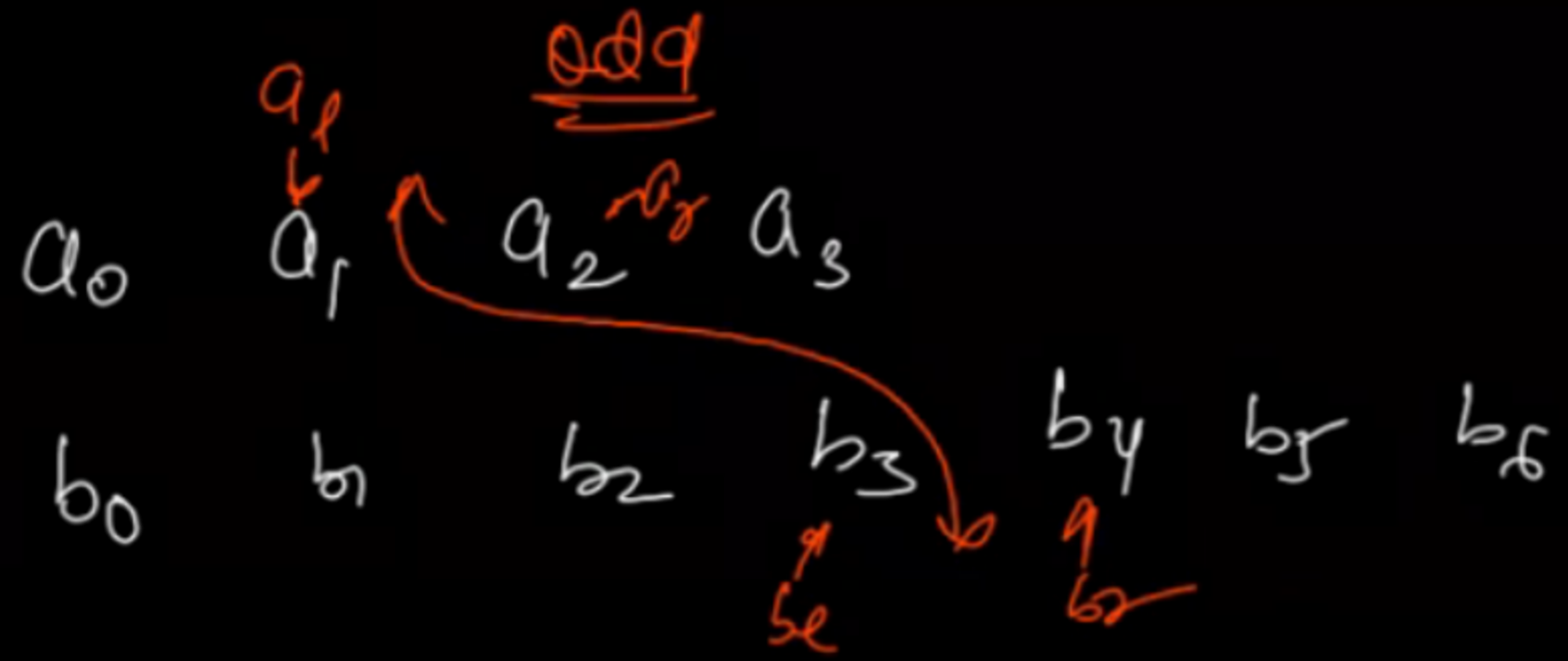
$b_0 \leq a_3$

Even

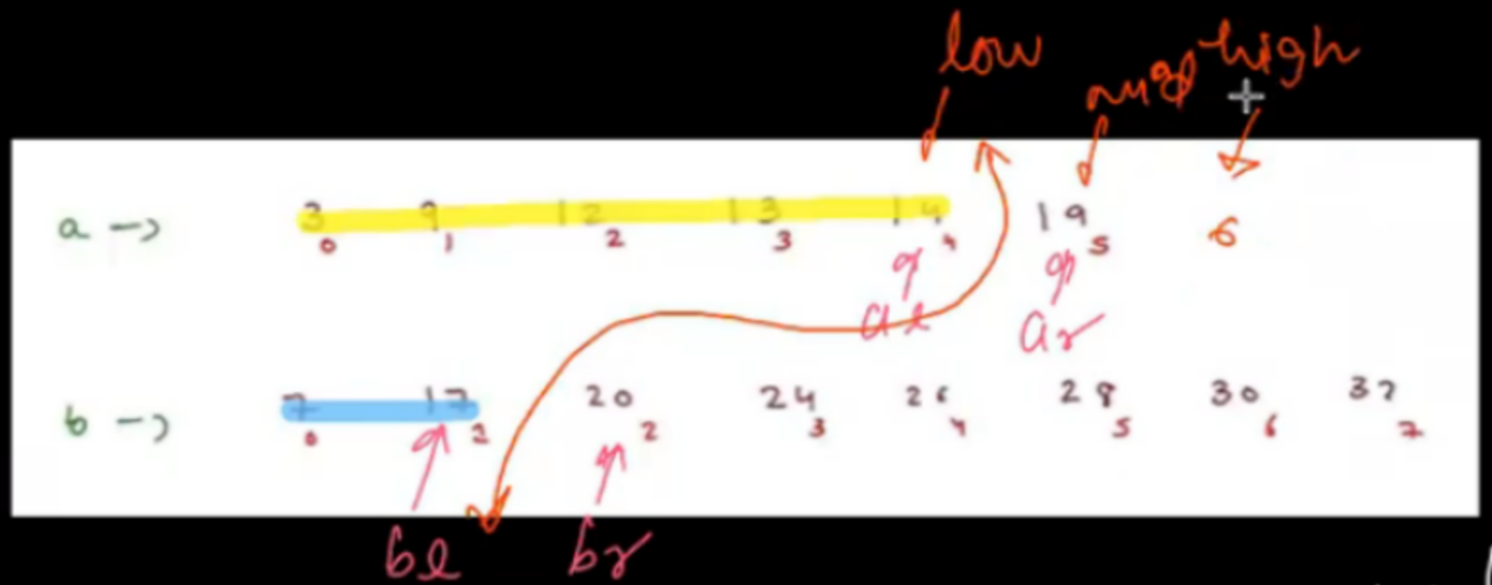


Median = $\{a_l, b_l, a_r, b_r\}$

+ Median of above sorted array



Median = $+ \max(a_l, b_l)$



low + high = 14

7l, 7r

violate → al <= br

high = mid - 1

bl <= ar

violate

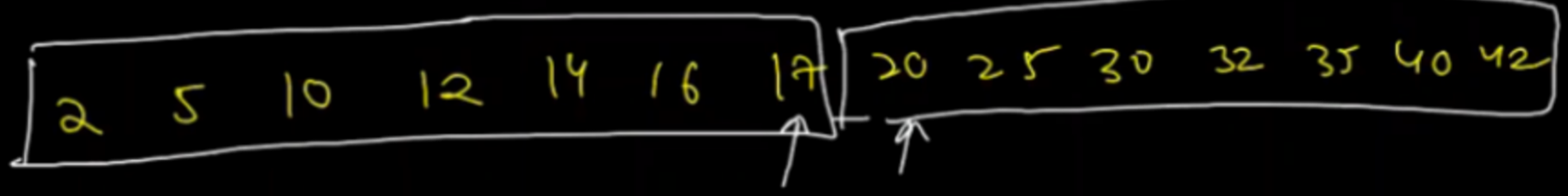
↳ low = mid + 1

14, 17, 19, 20

(17 + 19)/2 = 18

Corner case ①

$O(\log_2 N)$



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

        int lo = 0;
        int hi = nums1.length;
        int N = (nums1.length + nums2.length);
        int Nby2 = (N + 1) / 2;

        while(lo <= hi) {
            int mid = lo + (hi-lo)/2;

            int al = (mid -1 >= 0) ? nums1[mid - 1] : Integer.MIN_VALUE;
            int ar = (mid < nums1.length) ? nums1[mid] : Integer.MAX_VALUE;
            int bl = (Nby2 - mid -1 >= 0) ? nums2[Nby2 - mid -1] : Integer.MIN_VALUE;
            int br = (Nby2 - mid < nums2.length) ? nums2[Nby2 - mid] : Integer.MAX_VALUE;

            if(al <= br && bl <= ar) {
                //return median
                if(N % 2 == 1) {
                    return Math.max(al,bl);
                } else {
                     int[] arr = {al,bl,ar,br};
                     Arrays.sort(arr);
                     return (arr[1] + arr[2])/2.0;

                }
            }

            else if(al > br) {
                hi = mid -1;
            } else {
                lo = mid + 1;
            }
        }

        return 0.0;
    }
}
```