

Binary Search - lecture 4

Binary Search on Answer (V. IMP.)

① Book Allocation

- Split Array Largest Sum
- Capacity to ship Packages
- Painter's Partition

- ② Koko Eating Bananas
→ Smallest divisor threshold
- ③ Aggressive Cows
- ④ Woodcutting - SKO

{SPOJ}

More Questions to Practice

[Minimize Max Distance](#)

[Prata Spoj](#)

[Minimum Bouquets](#)

[Minimized Max Distance to Any Store](#)

[Magnetic Force Between Balls](#)

[Minimum Speed to Arrive on Time](#)

Book Allocation

$$book = 4 \quad stud = 2$$

Minimize the man^W
no. of pages

- ① Each stud should have atleast 1 book

20	10	30	40
----	----	----	----

stud ②

20	80	80
(20)	(10, 30, 40)	
30	70	
(20, 10)	(30, 40)	70

stud
③

20	10	70
(20)	(10)	(30, 40)
30	30	40
(20, 10)	(30)	(40)

20	40	40
(20)	(10, 30)	(40)

- ② we should allocate each book to exactly 1 student

③ Continuous allocation

④ Unbreakable item

Binary Search on pages { Books Array }

books = stud
 low = 40 high = 100
 $\text{mid} = 70 \quad Y$ ans = 100 70

$[40, 100]$

low = man of array

high = sum of array

$20 + 10 + 30 + 40$

low = 40 high = 69
 $\text{mid} = 54 \quad N$ ans = 70

$[40, 69]$

low = 55 high = 69
 $\text{mid} = 62 \quad Y$ ans = 70 62

$[55, 69]$

low = 55 high = 61
 $\text{mid} = 58 \quad N$ ans = 62

stud = 52

low = 59 high = 61
 $\text{mid} = 60 \quad Y$ ans = 60

stud = 12
 $\text{pages} = (20+10+30)(40)$

low = 60
 $\text{high} = 59$
 $\text{mid} = 59$ ans = 60

low = max of array high = sum of array
ans = high

while(low <= high) {

mid = low + (high - low)/2;

if (ispossible (arr, stud, mid) == true) {

 ans = mid; high = mid - 1;

} else {

 low = mid + 1;

}

return ans;

20	10	30	40
----	----	----	----

stud = 3

is possible

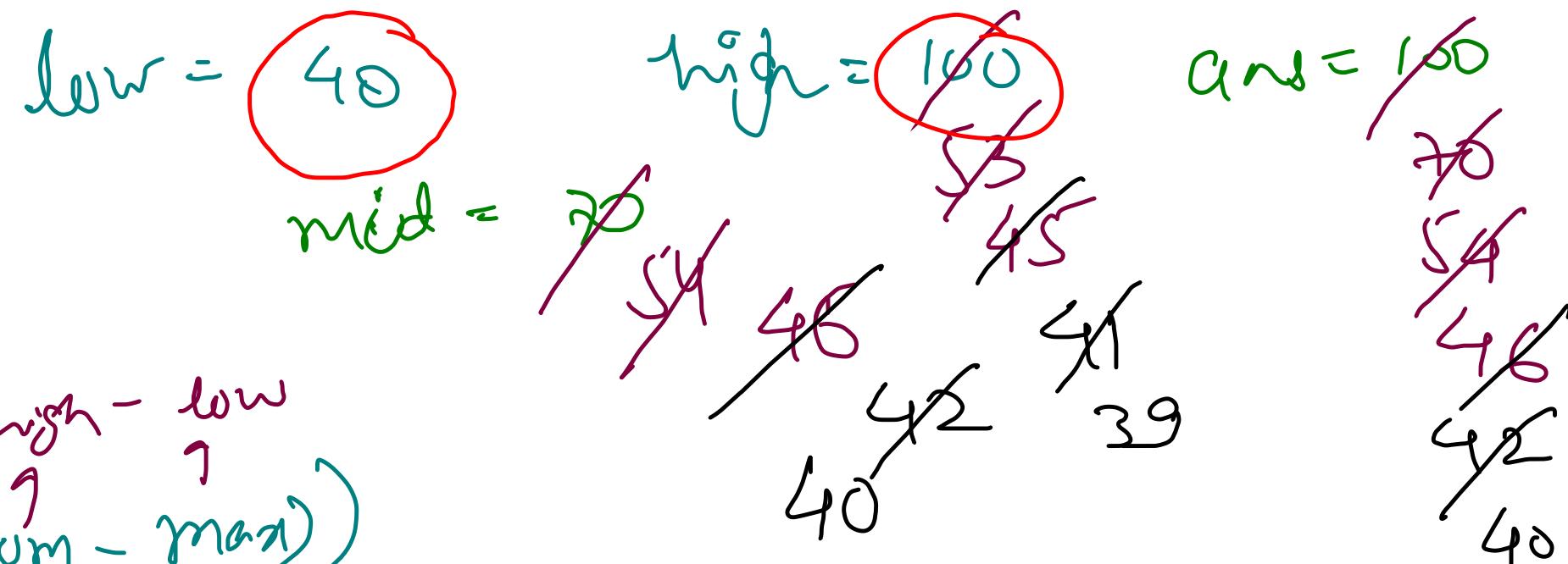
$$O(2^n + n * \log_2(\frac{\text{sum} - \text{max}}{\text{high} - \text{low}}))$$

\downarrow
low & high

```
public static boolean isPossible(int[] pages, int books, int maxLoad, int totalStud){
    int currStud = 1, currPages = 0;

    for(int i=0; currStud <= totalStud && i<books; i++){
        if(currPages + pages[i] <= maxLoad){
            currPages += pages[i];
        } else {
            currStud++;
            currPages = pages[i];
        }
    }

    if(currStud > totalStud) return false;
    return true;
}
```



```
public static int findPages(int[] pages, int books, int students)
{
    int low = maxOfArray(pages, books);
    int high = sumOfArray(pages, books);
    int ans = high;

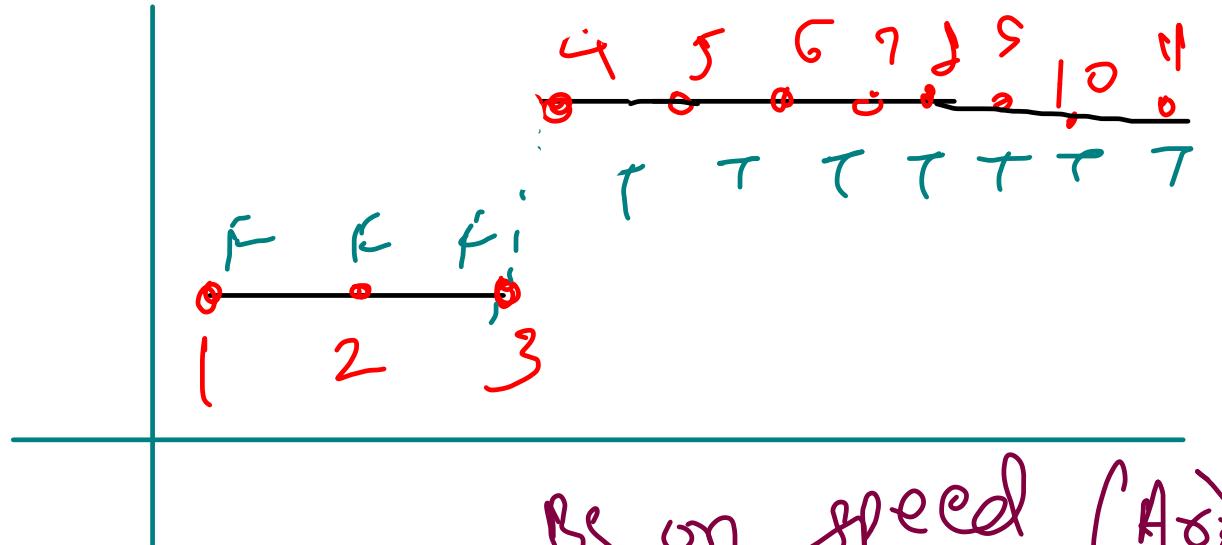
    while(low <= high){
        int mid = low + (high - low) / 2;

        if(isPossible(pages, books, mid, students) == true){
            ans = mid;
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    return ans;
}
```

Koko Eating Banana's

low = 1 high = 11



Be on speed (A~~o~~Xay)

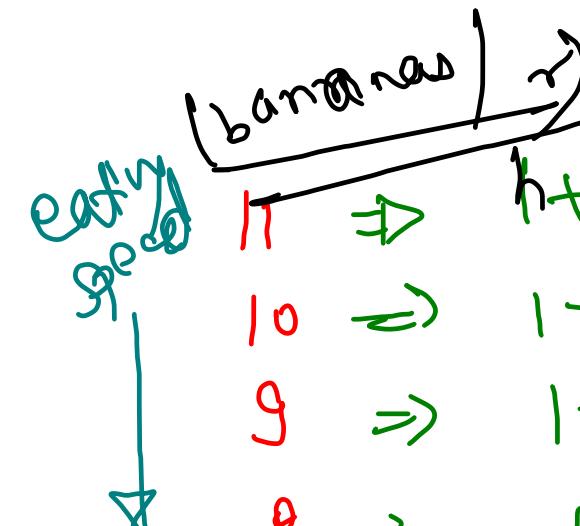
Koko loves to eat bananas. There are n piles of bananas, the i^{th} pile has $\text{piles}[i]$ bananas. The guards have gone and will come back in h hours.

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.

3	6	7	11
---	---	---	----

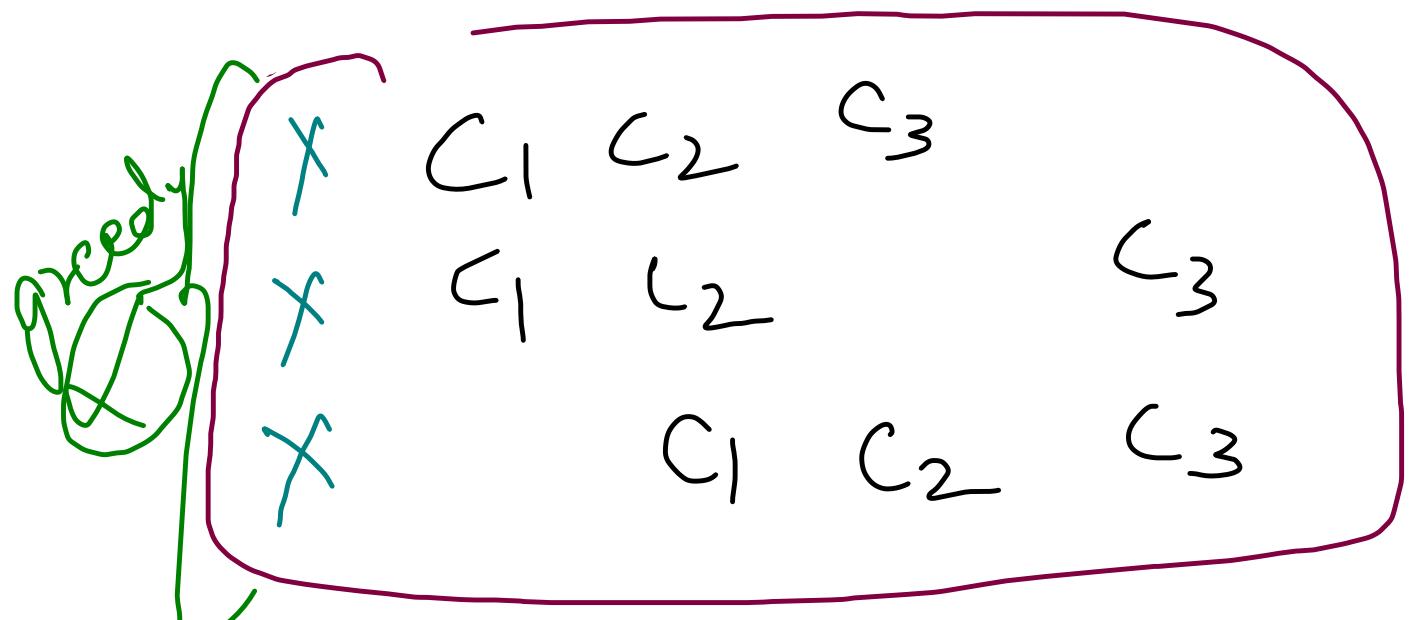


$11 \Rightarrow h + 1 + 1 + 1 = 1$ $10 \Rightarrow 1 + 1 + 1 + 2 = 5$ $9 \Rightarrow 1 + 1 + 1 + 2 = 5$ $8 \Rightarrow 1 + 1 + 1 + 2 = 5$ $7 \Rightarrow 1 + 1 + 1 + 2 = 5$ $6 \Rightarrow 1 + 1 + 2 + 2 = 6$ $5 \Rightarrow 1 + 2 + 2 + 3 = 8$ $4 \Rightarrow 1 + 2 + 2 + 3 + 2 = 10$ $3 \Rightarrow 2 + 3 + 4 + 6 = 15$ $2 \Rightarrow 3 + 6 + 7 + 11 = 27$	$h = 8$ $a_n = X$ 1 2 3 4 5 6 7 8 9 10 11 N
--	--

Aggressive Cows

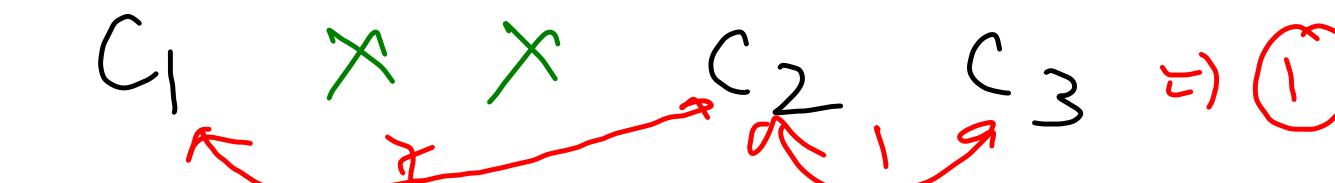
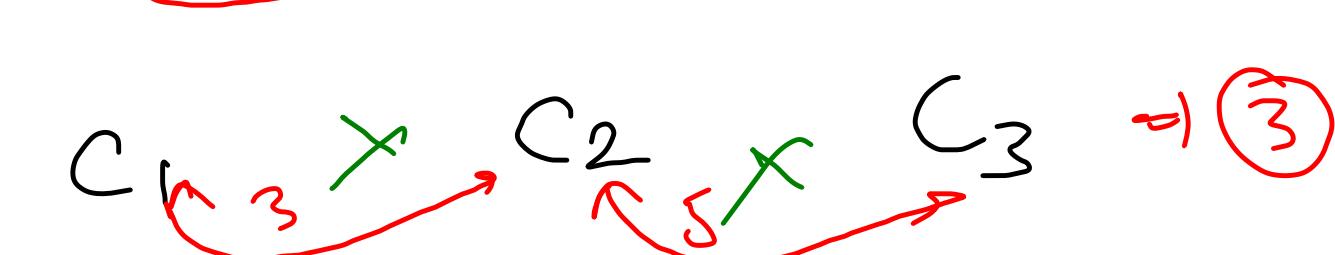
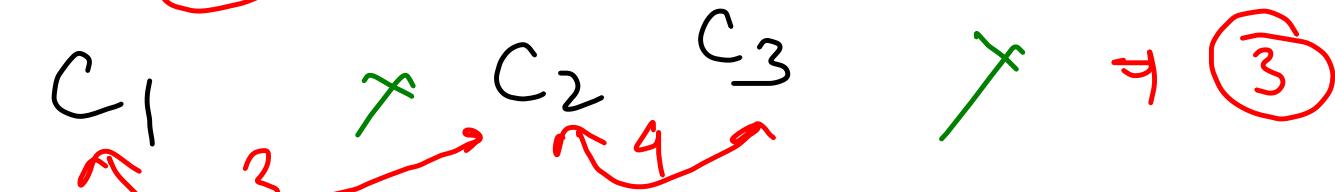
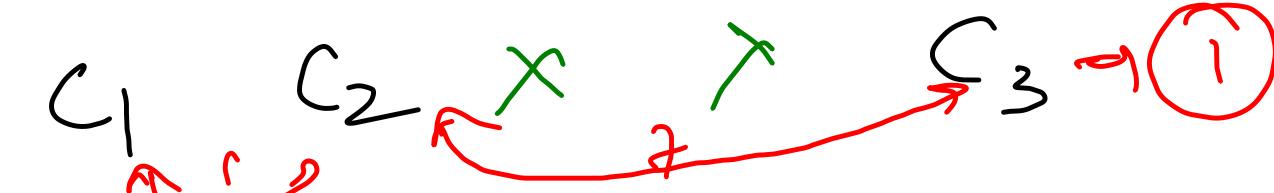
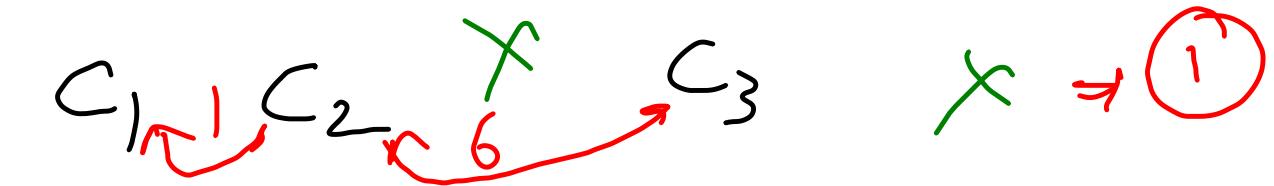
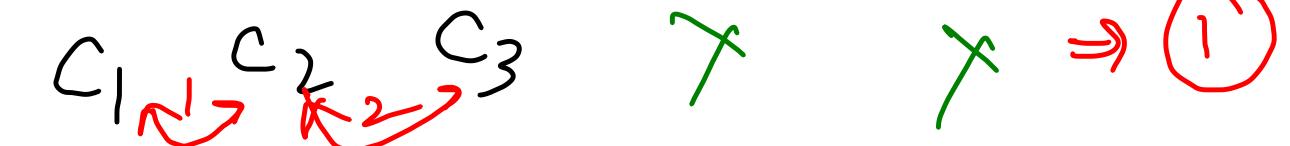
Farmer John has built a new long barn, with N ($2 \leq N \leq 100,000$) stalls. The stalls are located along a straight line at positions x_1, \dots, x_N ($0 \leq x_i \leq 1,000,000,000$).

His C ($2 \leq C \leq N$) 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?

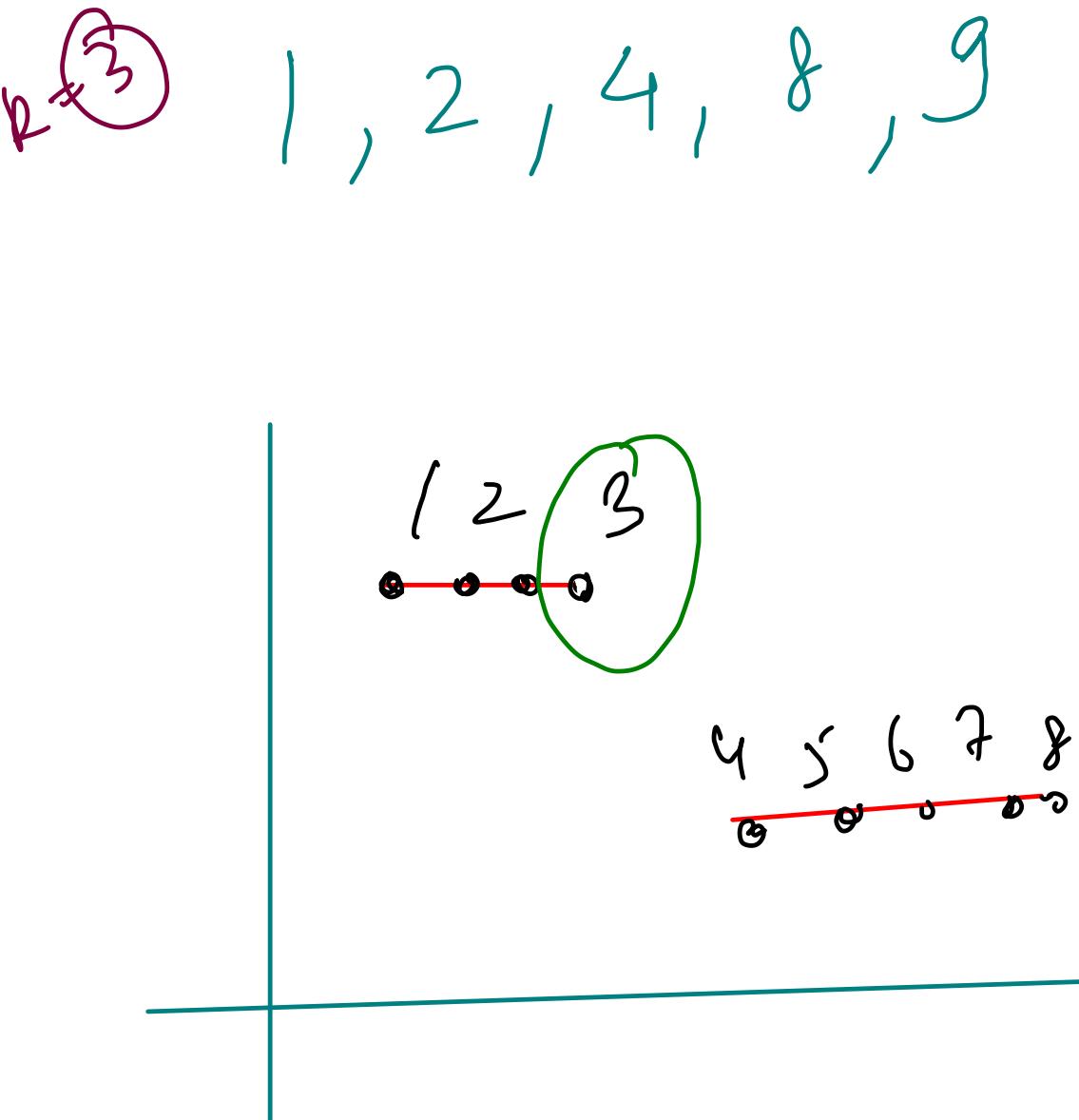


cows = ③

{ 1, 2, 4, 8, 9 }



Binary Search on Adjacent distance

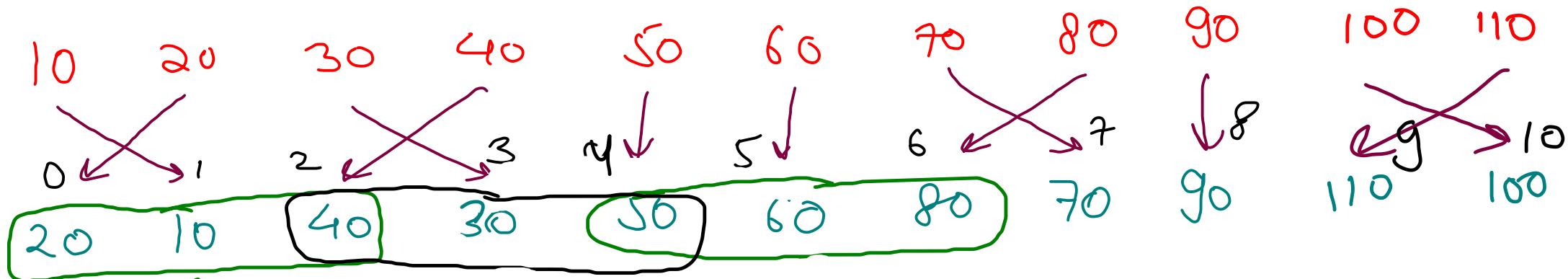


Binary Search - Lecture 5

- ① Search in nearly sorted Array
- ② Jump/ Step Search
- ③ Unbounded/ Infinite sorted Array
- ④ Staircase Search (search in matrix - I)
- ⑤ Search in Matrix - II

Search in Nearly Sorted Array

target = 35
R = 1



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

```
public static int solve(int[] arr, int target) {
    int low = 0, high = arr.length - 1;
    while(low <= high){
        int mid = low + (high - low) / 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(target == arr[mid]) return mid;
        if(target == lval) return mid - 1;
        if(target == rval) return mid + 1;

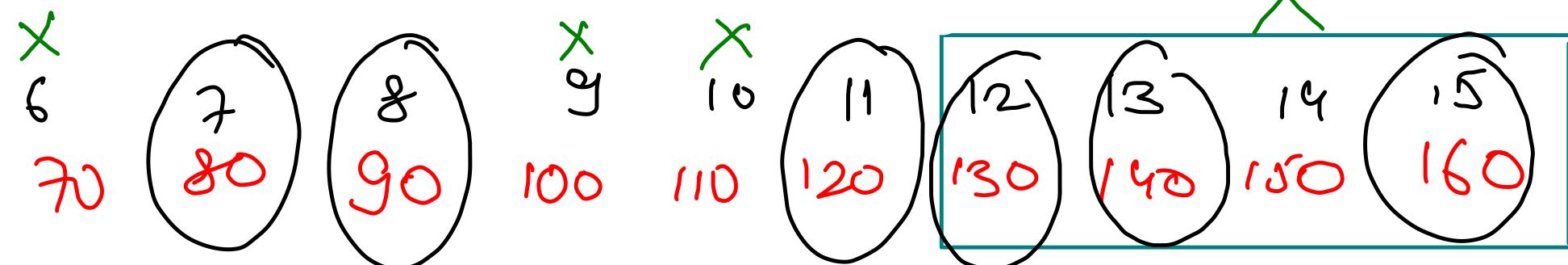
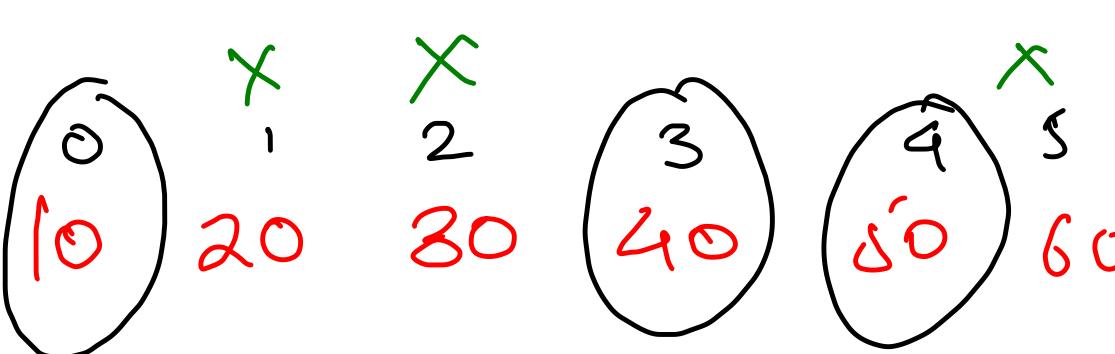
        if(target < arr[mid]) high = mid - 2;
        else low = mid + 2;
    }

    return -1;
}
```

Jump Search

HS $\Rightarrow O(N)$
 BS $\Rightarrow O(\log N)$

target = 140



$$\text{Total windows} = \left\lceil \frac{N}{k} \right\rceil = \sqrt{N}$$

k = window size

N = no of elements

Best case \Rightarrow First window

first elec(1) 10
 last elec 60
 Δk

Worst case $\Rightarrow \left\lceil \frac{N}{k} \right\rceil + k$
 skipping $\frac{N}{k}$ windows
 Linearch search in last window

$$k = \sqrt{N} \Rightarrow \left\lceil \frac{N}{\sqrt{N}} \right\rceil + \sqrt{N} = 2\sqrt{N} \Rightarrow O(\sqrt{N})$$

Step Search

$\{ \text{Index } X \rightarrow \text{Property} \}$
 $\{ \text{Element/Value } \rightarrow \text{Property} \}$

target = 28
 $k = 3$

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

$k = 3$
 $\{ \boxed{15, 13, 16}, \boxed{14, 17, 20}, \boxed{19, 22}, 24, 23 \}$
 target = 14

$\frac{24+14}{3} = 3$
 24 21 20 18 16 17 14

target = 14

```

public static int search (int arr[], int n, int target, int k) {
  int idx = 0;
  while(idx < n){

    if(arr[idx] == target) return idx;

    int minJump = (Math.abs(target - arr[idx])) / k;
    if(minJump == 0) minJump = 1;

    idx = idx + minJump;
  }
  return -1;
}
  
```

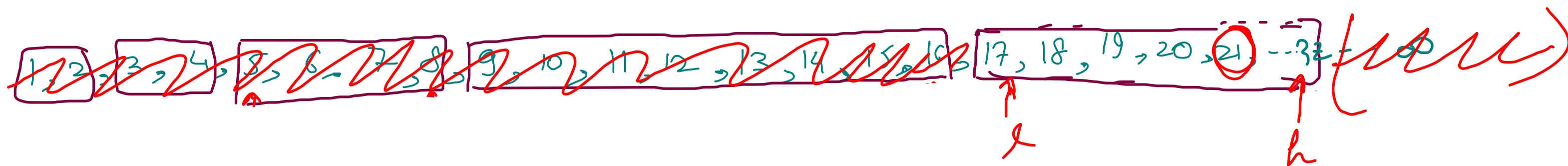
worst case $\Rightarrow O(N)$

avg case $\Rightarrow O(\sqrt{n})$

best case $\Rightarrow O(1)$

(Infinite Sorted Array)
Unbounded Binary Search

target = 21



[1, 2]

[9, 16]

$$T(N) = T\left(\frac{2}{3}N\right) + k$$

[3, 4]

[17, 32]

[5, 8]

Doubling the window size will take least time as compared to fixed size

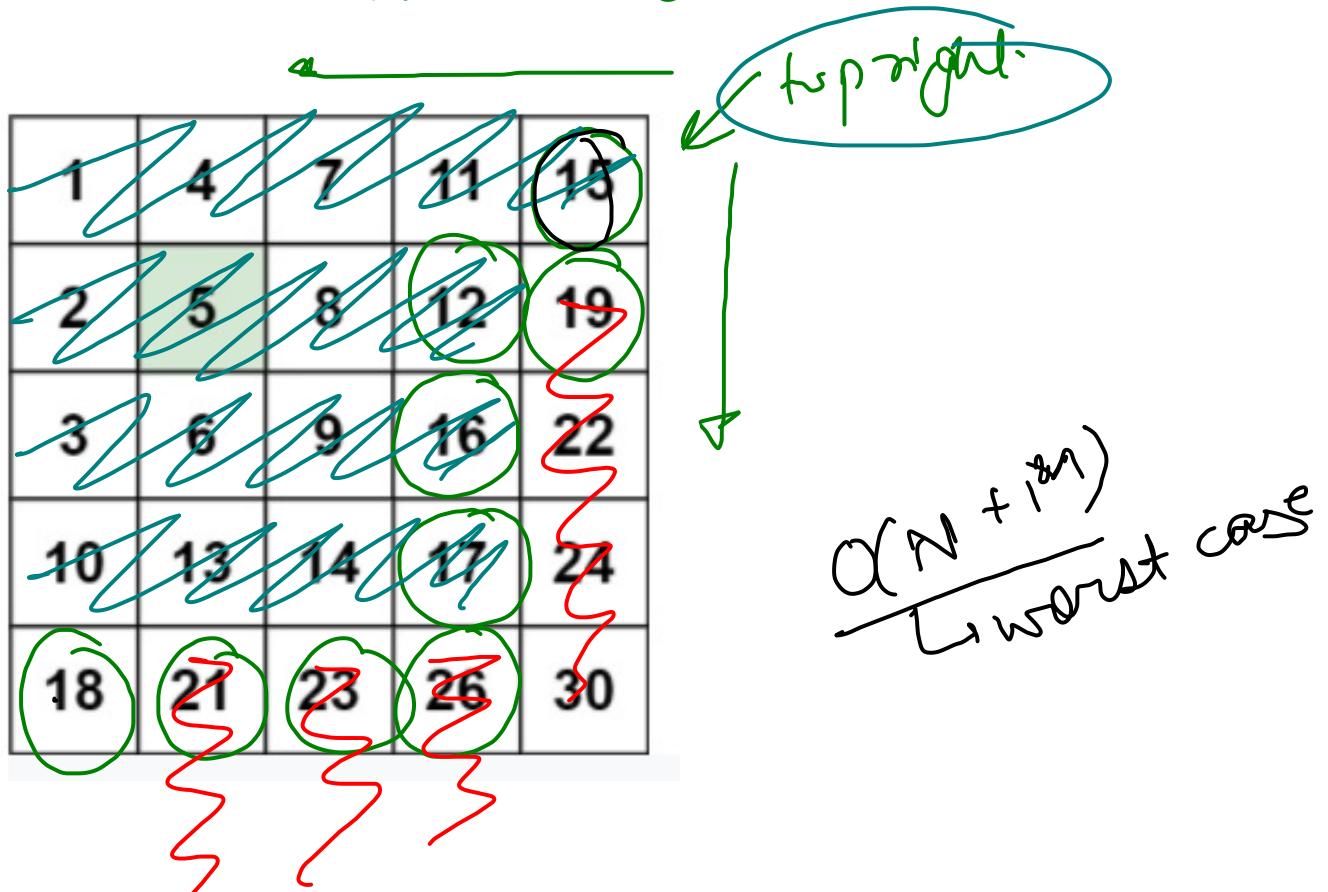
```
public static int binarySearch(int[] arr, int low, int high, int target){
    if(low > high) return -1;

    int mid = low + (high - low) / 2;
    if(arr[mid] == target) return mid;
    if(target < arr[mid]) return binarySearch(arr, low, mid - 1, target);
    return binarySearch(arr, mid + 1, high, target);
}

public static int binarySearchInfinite(int[] arr, int low, int high, int target){
    if(low > high) return -1;

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

// Invocation -> binarySearch(arr, 0, n - 1, target)
// Invocation -> binarySearchInfinite(arr, 0, 1, target)
```

Row \rightarrow SortedColumn \rightarrow Sorted

```

public boolean searchMatrix(int[][] matrix, int target) {
    // Staircase Search -> O(N + M) in Worst Case
    int row = 0, col = matrix[0].length - 1; // top right corner

    while(row < matrix.length && col >= 0){
        if(matrix[row][col] == target) return true;

        if(target < matrix[row][col]) col--;
        else row++;
    }

    return false;
}

```

① Brute force

Binary search on each row $\sim O(n \log m)$

② Staircase search

\hookrightarrow Divide & Conquer

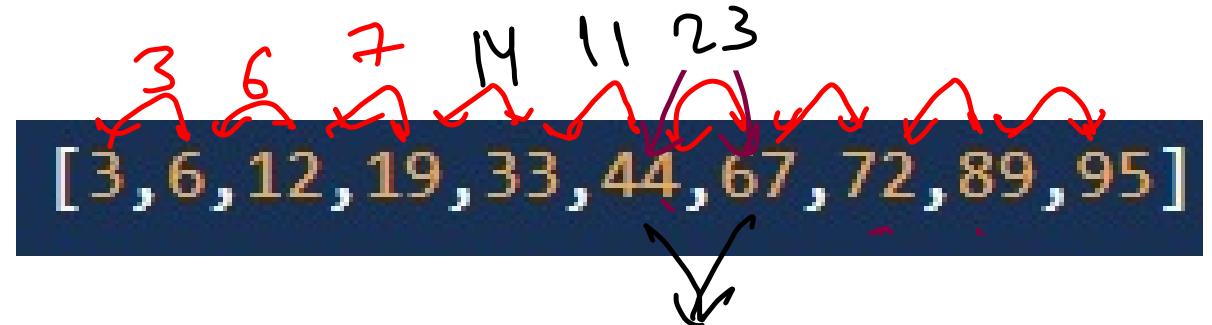
target = 18

$\left\{ \begin{array}{l} \text{arr[mid]} > \text{target} \quad (\text{d--}) \\ \quad \quad \quad \text{discard column} \\ \text{arr[mid]} < \text{target} \\ \quad \quad \quad \text{discard row++} \end{array} \right.$

\hookrightarrow unsuccessful search
 \Rightarrow out of matrix

848 · Minimize Max Distance to Gas Station

$p = 2$



low - ③

high - ② ③

mid = $\frac{2+3}{2} = 1\ 3\ N$

mid = 18 4

mid $\rightarrow \frac{14+17}{2} = 15\ 1$
mid < 1

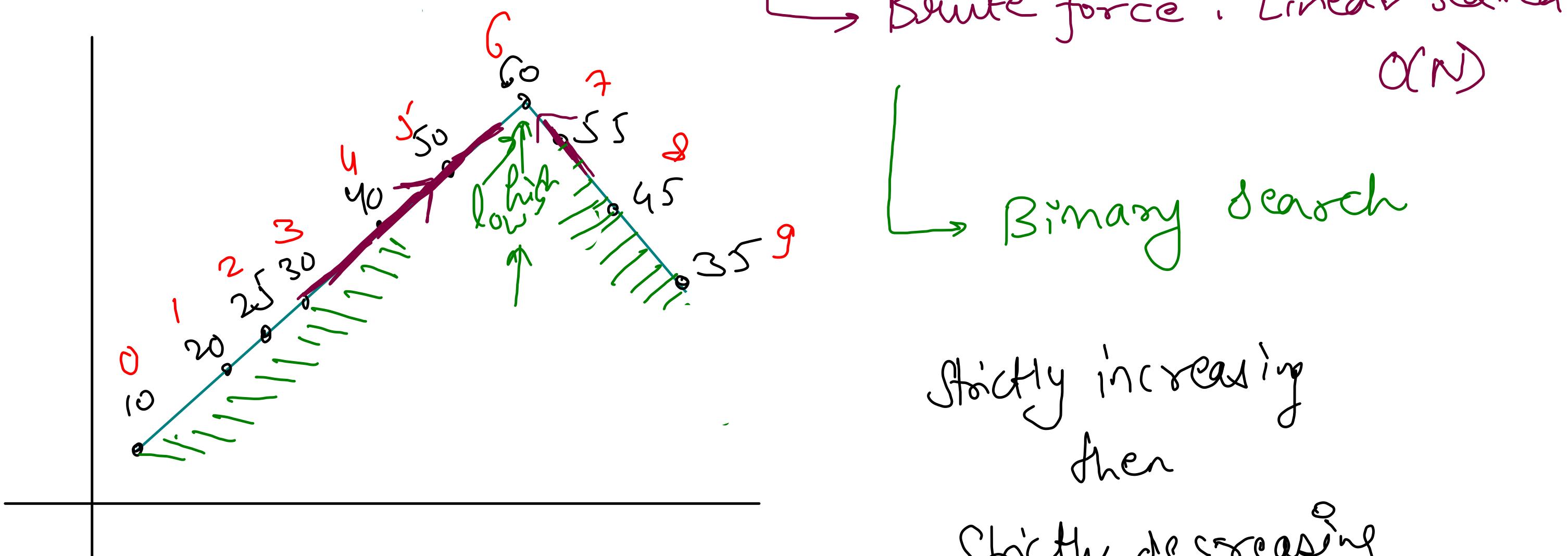
```
bool possible(double x, vector<int> &stations, int K)
{
    int count = 0;
    for (int i = 1; i < stations.size(); i++)
        count += (int) ((stations[i] - stations[i - 1]) / x);
    return count <= K;
}
```

```
double minmaxGasDist(vector<int> &stations, int K)
{
    double low = 0, high = 1e8;
    while (high - low > 1e-6)
    {
        double mid = (low + high) / 2.0;
        if (possible(mid, stations, K)) high = mid;
        else low = mid;
    }
    return low;
}
```

Binary Search - lecture 6

- ① Maximum Element in Bitonic Array
 - ② Search Element in Bitonic Array
 - ③ Peak Element
 - ④ Unique Element - Sorted Array
 - ⑤ Missing Element of Arithmetic Progression
- } Bitonic Array

Bitonic Array (Peak Element) → Maximum



Find Peak Element

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 $\text{nums}[-1] = \text{nums}[n] = -\infty$.

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

→ There can be more than one peaks

→ We do not have to return global maxima. We can return any local maxima

UNSORTED ARRAY

-2^{63}

long. MIN-VALUE

→ $\text{nums}[-1] = \text{nums}[n] = -\infty$

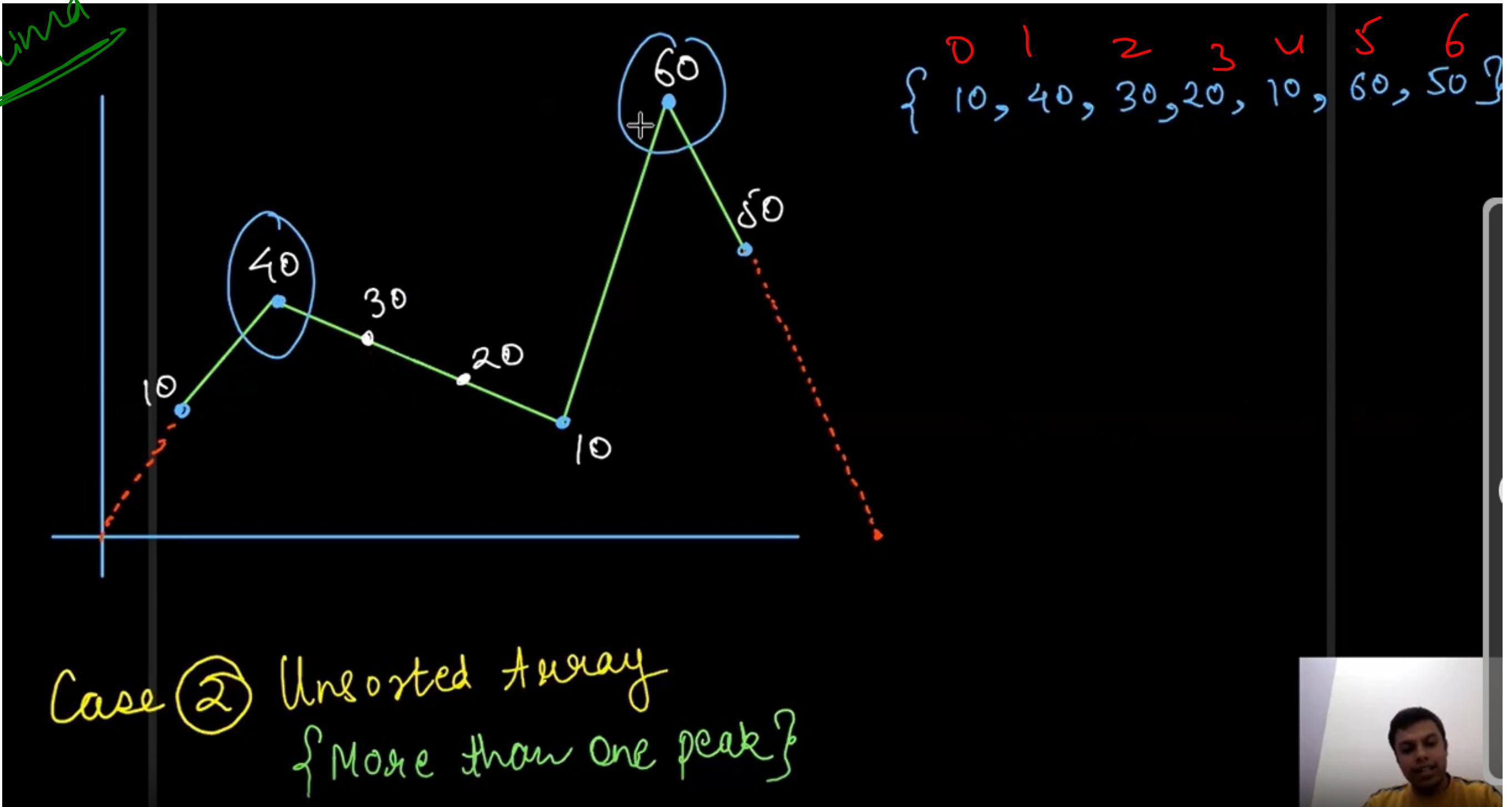
→ No two adjacent elements are equal

→ There will be at least one peak element.

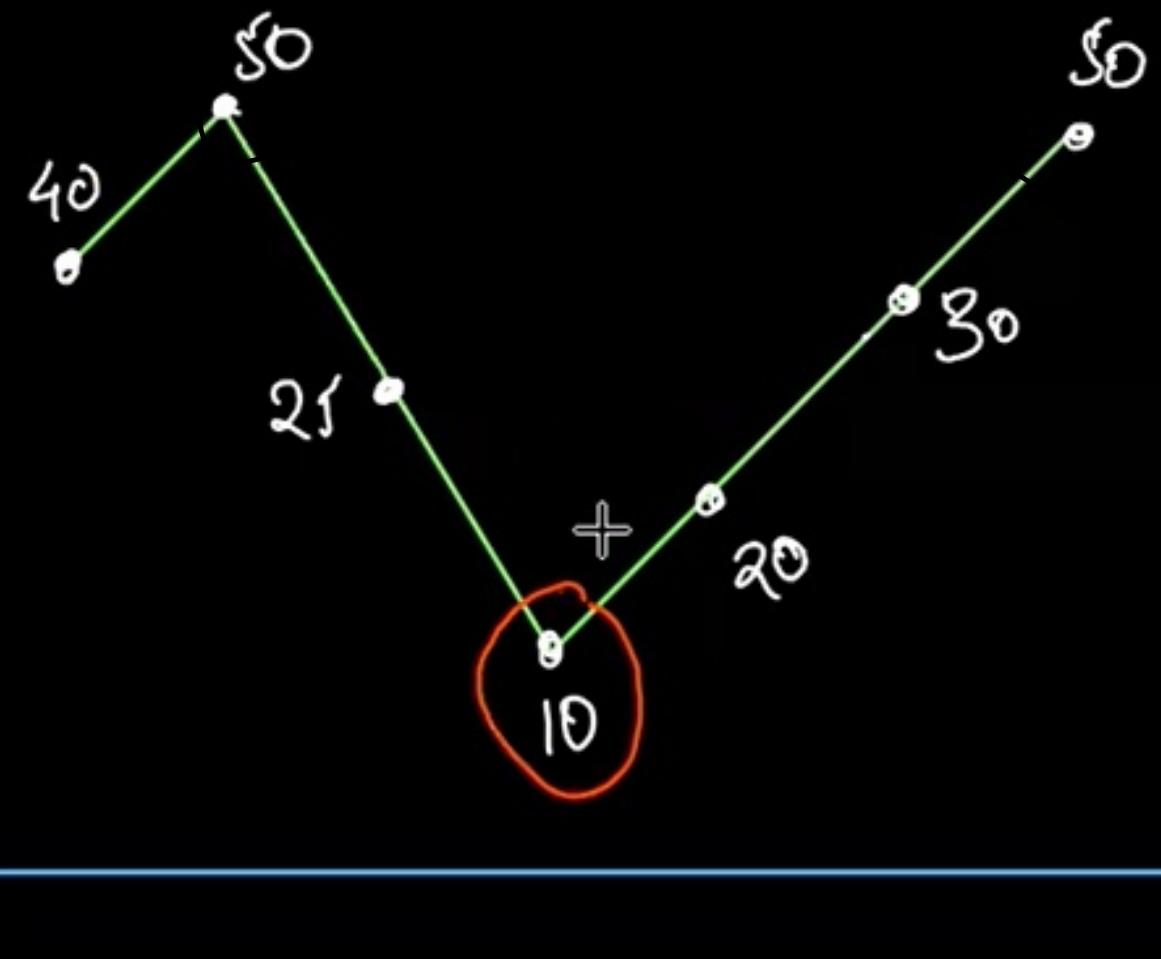
→ Modified Binary search can be applied.



*local
maxima*

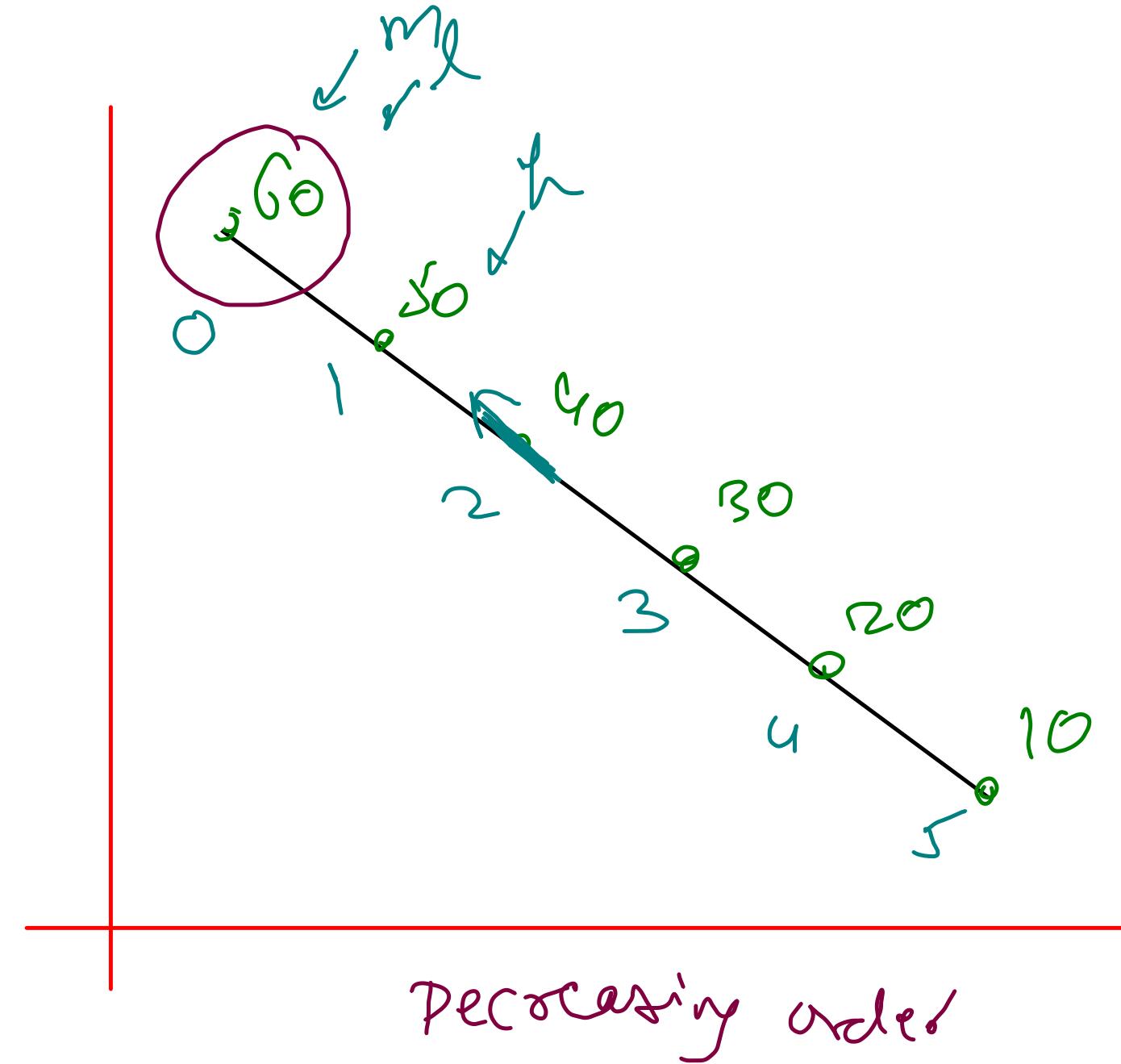
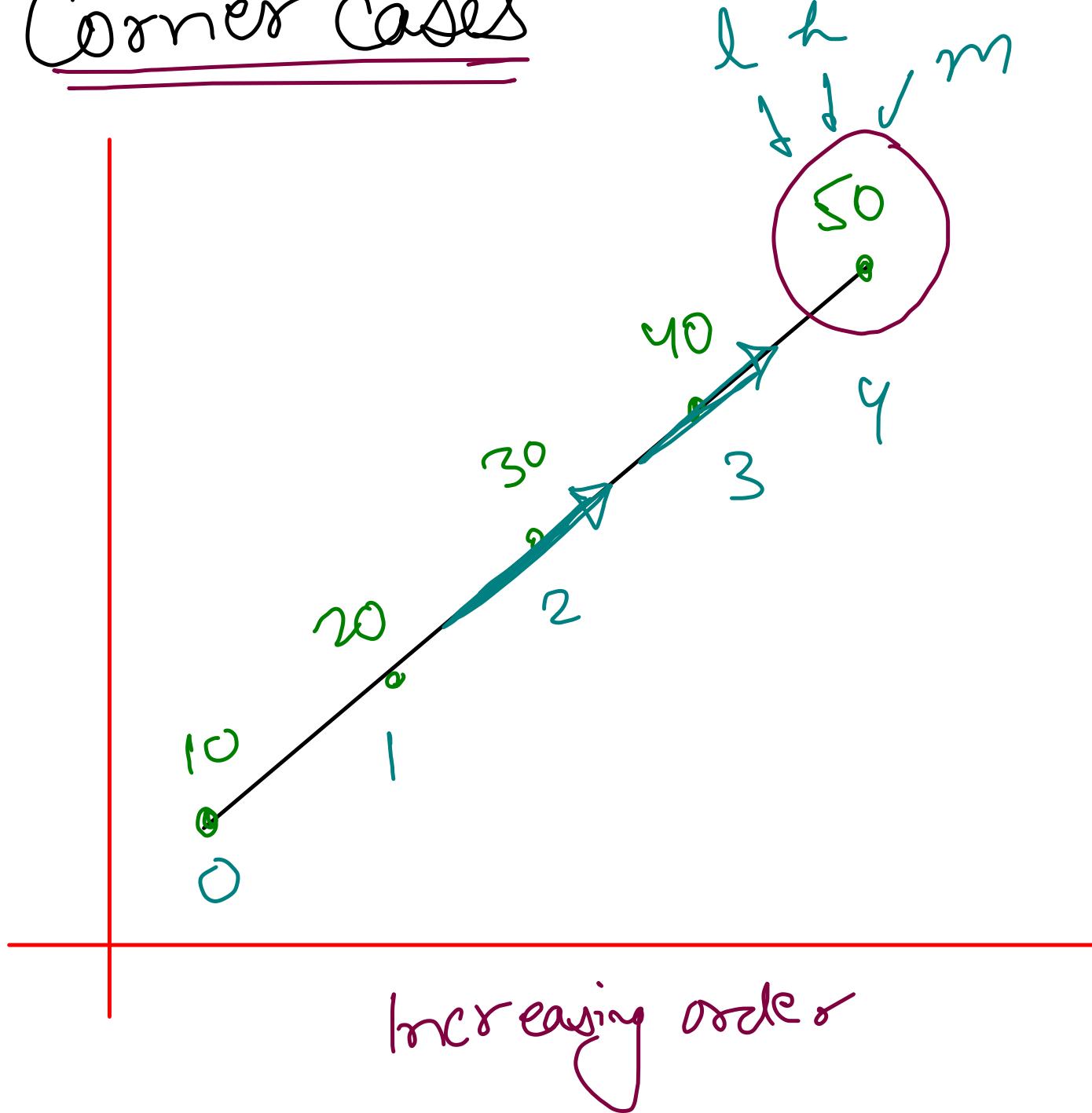


Case 5 :> Mid on valley



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

Corner Cases



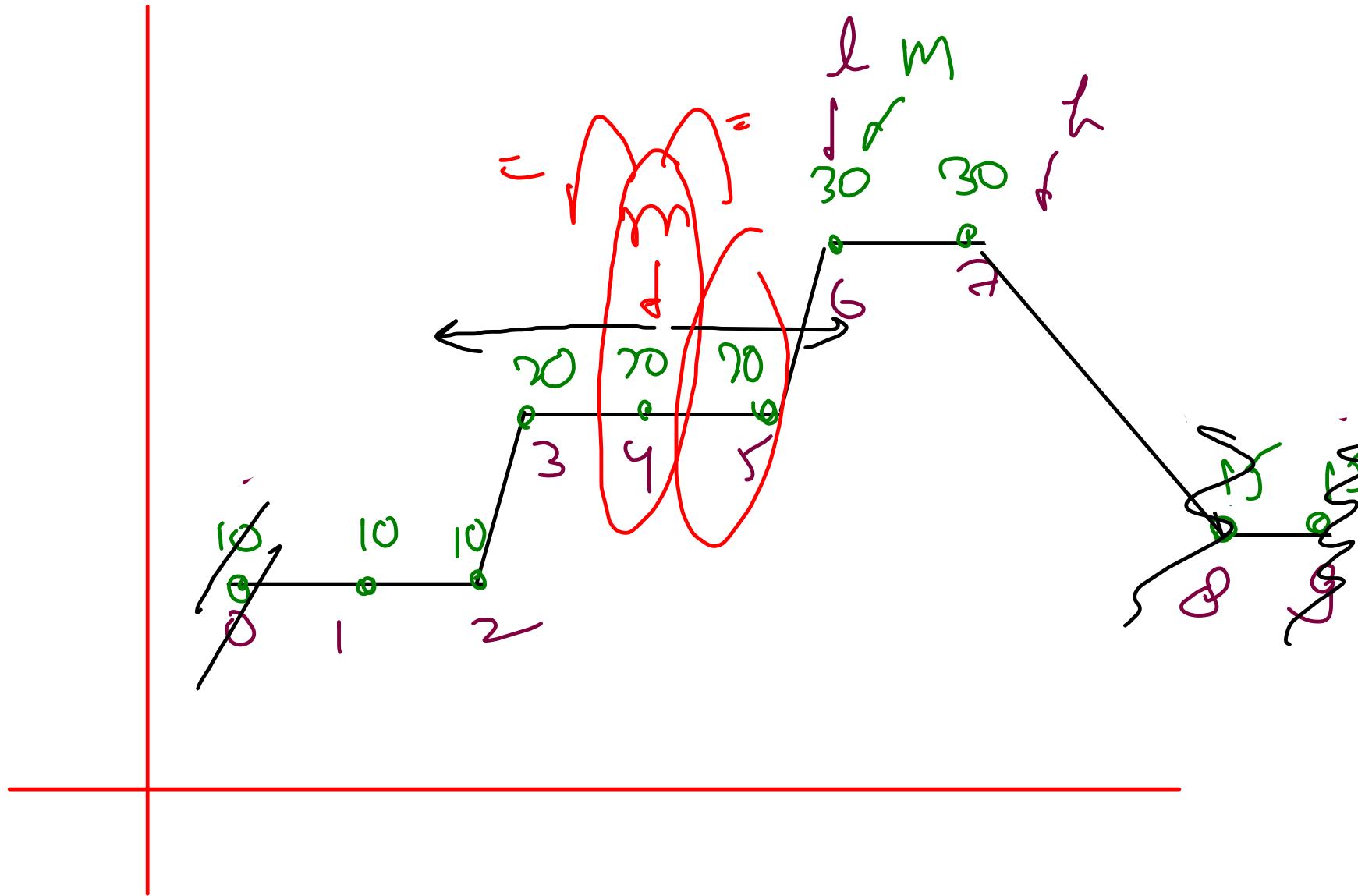
```
public int peakIndexInMountainArray(int[] arr) {
    int low = 0, high = arr.length - 1;
    while(low <= high){
        int mid = low + (high - low) / 2;

        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(lval > arr[mid]){
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }
    return -1;
}
```

```
{10,10,10, 20,20,20,30,30, 15,15 }
```

Code will fail in
duplicate Elements



```

public int peakIndexInMountainArray(int[] arr) {
    int low = 0, high = arr.length - 1;
    while(low <= high){
        int mid = low + (high - low) / 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(lval > arr[mid] && arr[mid] > rval){
            high = mid - 1;
        } else if(lval < arr[mid] && rval > arr[mid]){
            low = mid + 1;
        } else {
            // worst case -> O(N)
            if(arr[left] <= arr[right]) left++;
            else right--;
        }
    }
    return -1;
}

```

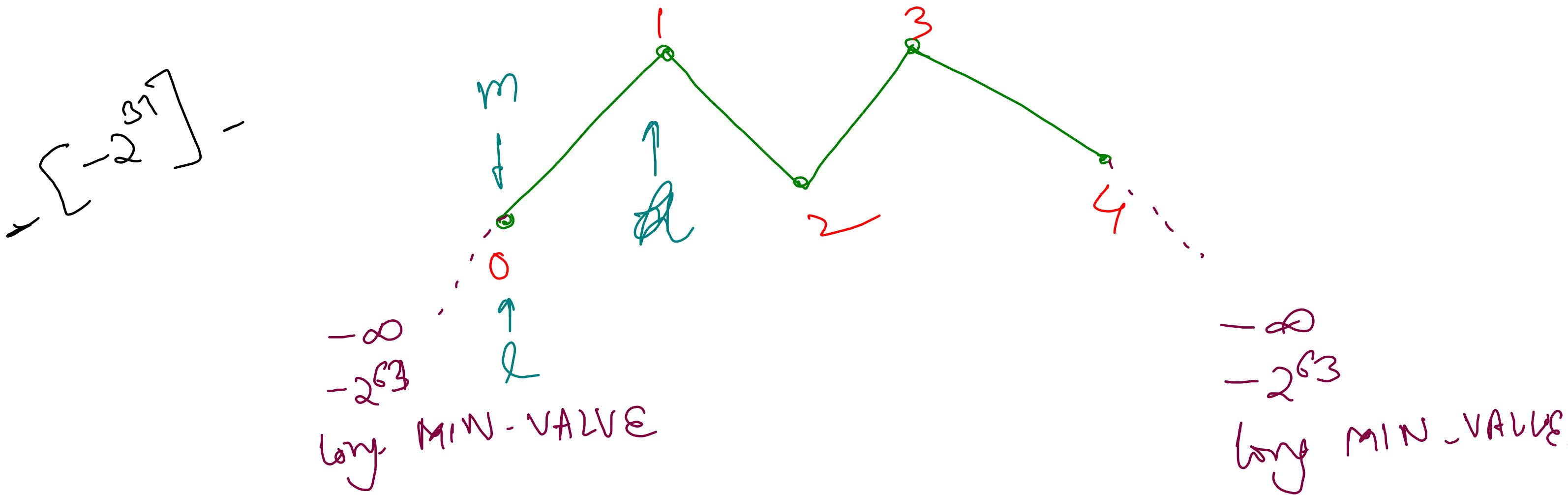
Handle Duplicates

↓

Worst case $O(N)$

Corner Case

$$\left[-2^{31}, 0, -2^{31}, 0, -2^{31} \right]$$



```
class Solution {
    public int findPeakElement(int[] arr) {
        int low = 0, high = arr.length - 1;

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

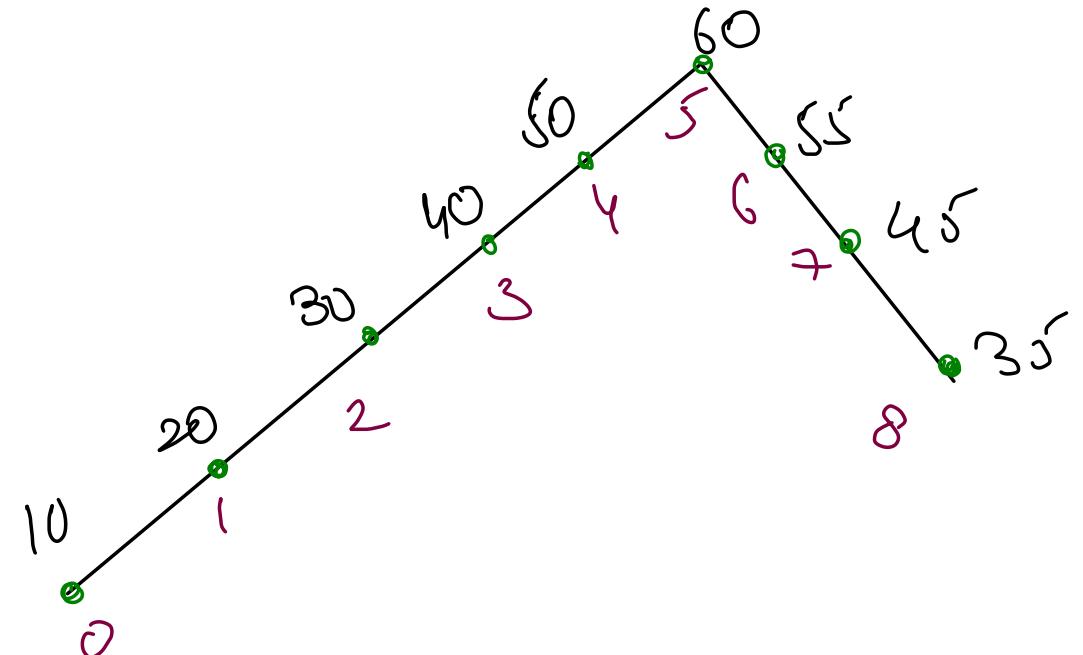
            long lval = (mid - 1 >= 0) ? arr[mid - 1] : Long.MIN_VALUE;
            long rval = (mid + 1 < arr.length) ? arr[mid + 1] : Long.MIN_VALUE;

            if((long)arr[mid] > lval && (long)arr[mid] > rval){
                return mid;
            }

            if(lval < (long)arr[mid]){
                // arr[mid] < rval
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return -1;
    }
}
```

Search in Bitonic Array



target = 20

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

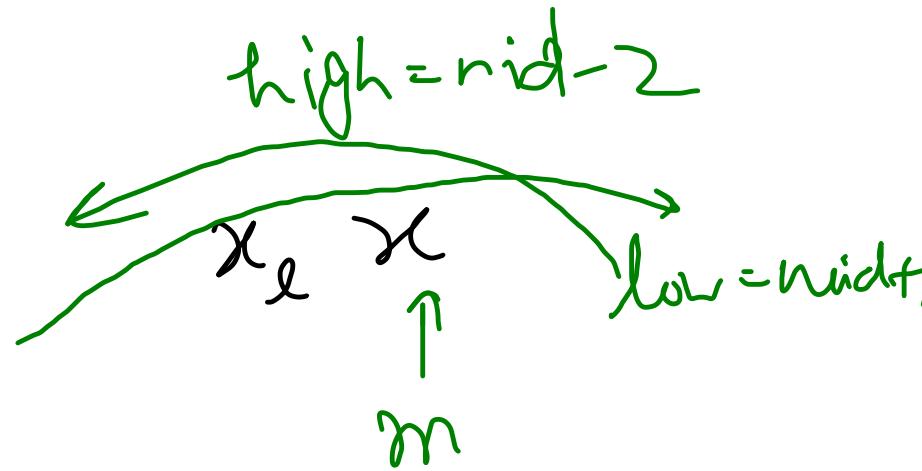
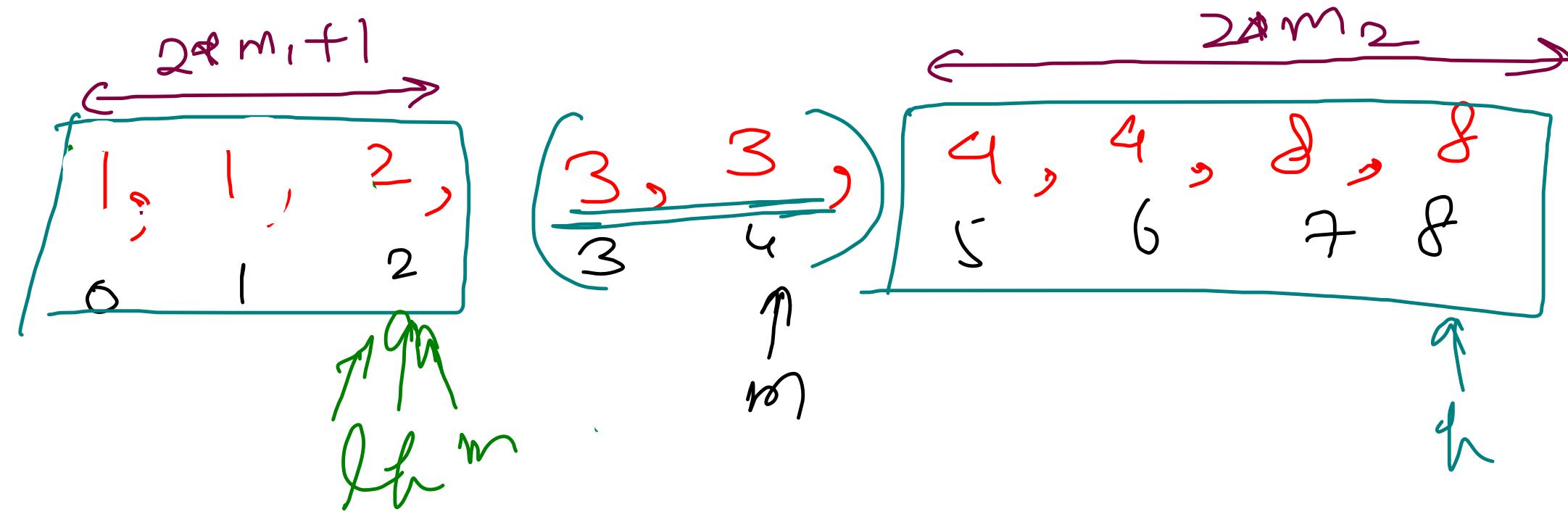
```
public int binarySearch(int[] A, int left, int right, int target, boolean isInc){
    if(left > right) return -1;
    int mid = left + (right - left) / 2;

    if(A[mid] == target) return mid;
    if((isInc == true && A[mid] < target) || (isInc == false && A[mid] > target)){
        return binarySearch(A, mid + 1, right, target, isInc);
    } else {
        return binarySearch(A, left, mid - 1, target, isInc);
    }
}
```

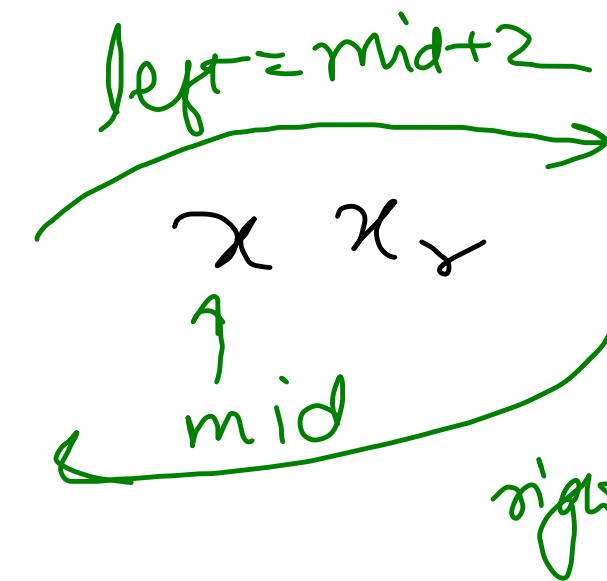
$O(3 \log_2 N)$

Single Element in Sorted Array

$O(\log_2 N)$



$$\begin{aligned} \text{left range} &= m-l-1 \\ \text{right range} &= h-m \end{aligned}$$



$$\begin{aligned} \text{left} &= \text{mid}+2 \\ x_1 & x_2 \\ \uparrow & \uparrow \\ \text{mid} & \end{aligned}$$

$$\begin{aligned} \text{left range} &= m-l \\ \text{right range} &= h-m-1 \end{aligned}$$

```

public int singleNonDuplicate(int[] nums) {
    int low = 0, high = nums.length - 1;
    while(low <= high){

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

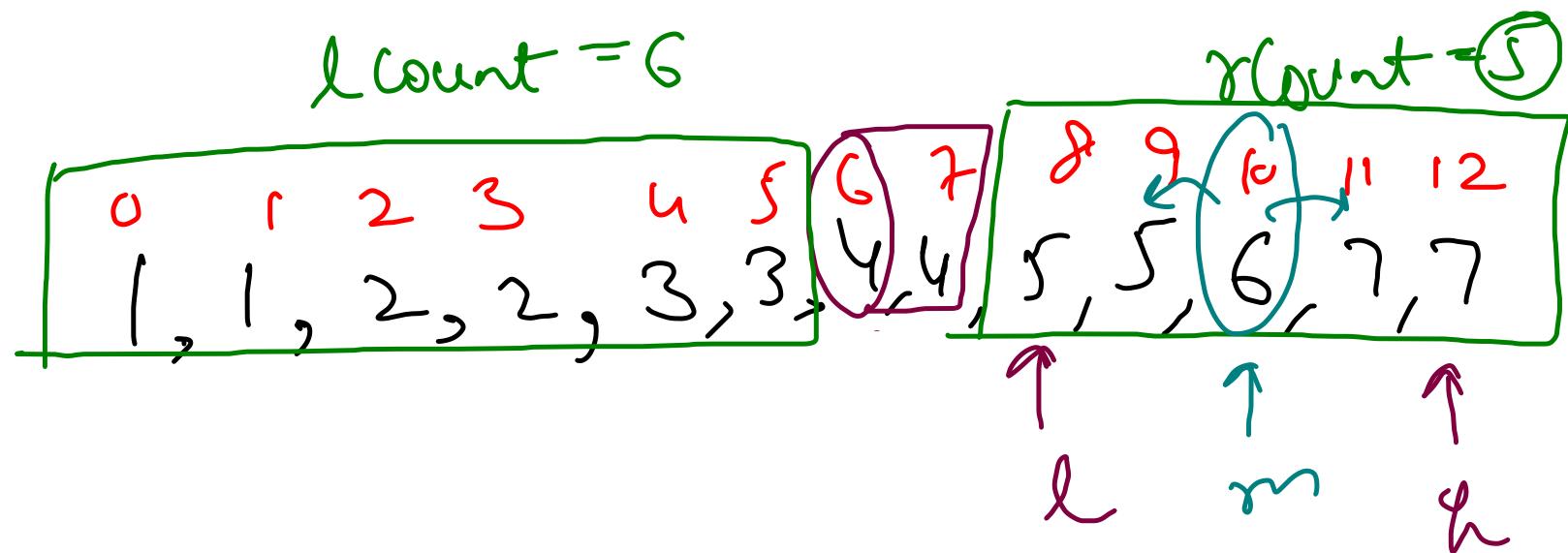
        int lval = (mid - 1 >= 0) ? nums[mid - 1] : -1;
        int rval = (mid + 1 < nums.length) ? nums[mid + 1] : -1;

        if(nums[mid] != lval && nums[mid] != rval){
            // mid is single element
            return nums[mid];
        }

        if(nums[mid] == lval){
            int lCount = mid - low - 1;
            if(lCount % 2 == 1){
                high = mid - 2;
            } else {
                low = mid + 1;
            }
        } else {
            int lCount = mid - low;
            if(lCount % 2 == 1){
                high = mid - 1;
            } else {
                low = mid + 2;
            }
        }
    }

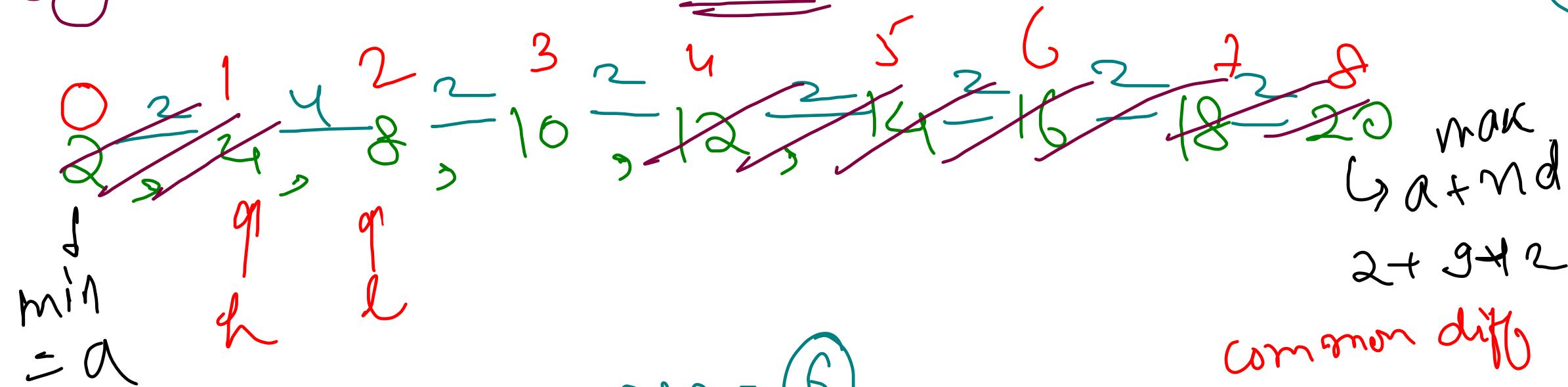
    return -1;
}

```



Missing Element in AP

Answer = 6



$$a + nd = 2 + 2 \times 2 = 6$$

$$\begin{aligned} & 0 & 1 & 2 & 3 & 4 & 5 & \dots & n-1 \\ & a & a+d & \cancel{a+2d} & a+3d & a+4d & a+5d & \dots & a+(n-1)d \\ & & & & & & & & = \frac{\text{max} - \text{min}}{\text{no of ele}} \\ & & & & & & & = \frac{a+nd - a}{n} \\ & & & & & & & = \frac{nd}{n} = d \end{aligned}$$

```
int low = 0, high = n - 1;
int commonDiff = (arr[n - 1] - arr[0]) / n;
int ans = 0;

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

    int actualVal = arr[mid];
    int expectedVal = arr[0] + commonDiff * mid;

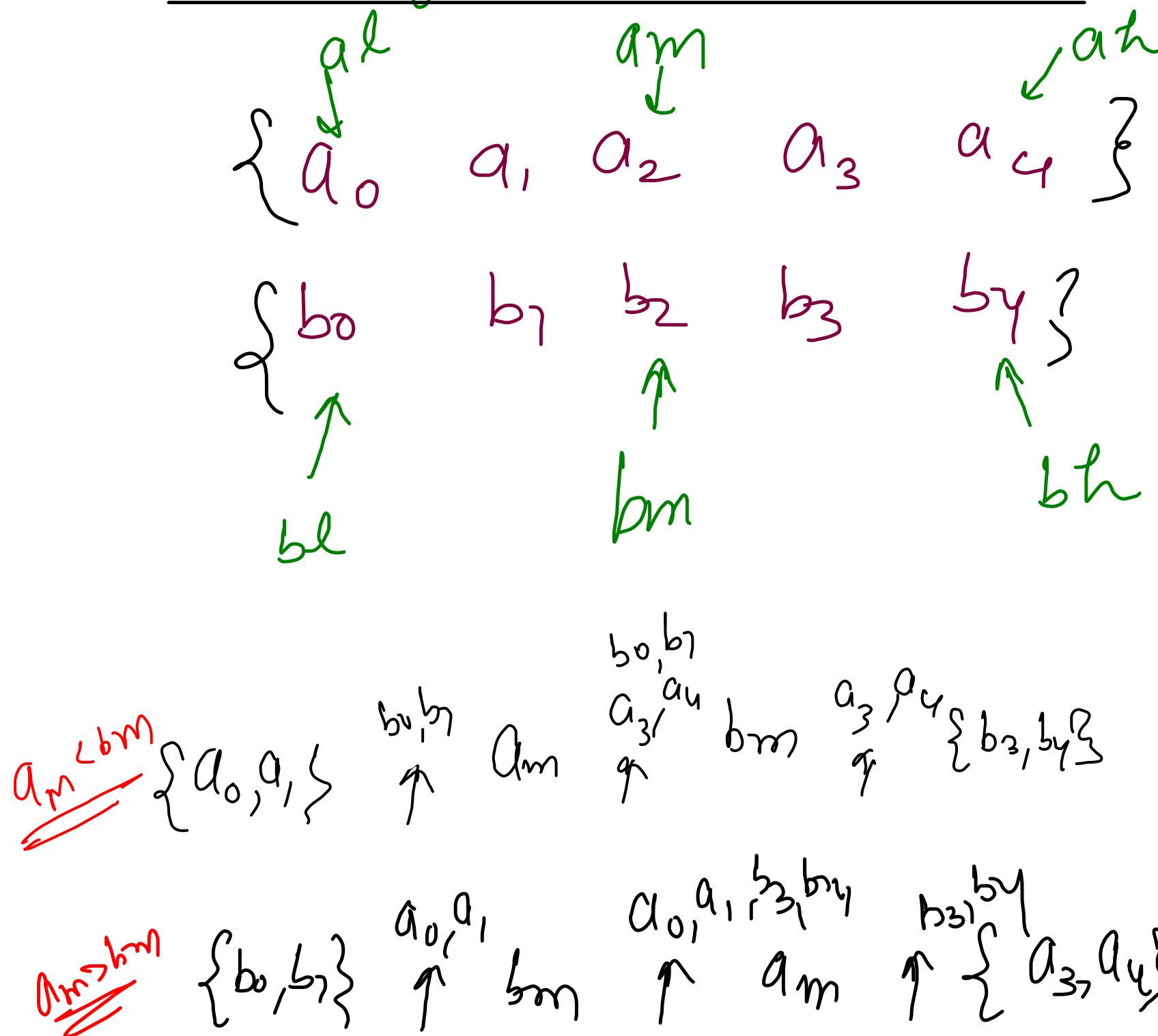
    if(actualVal == expectedVal){
        low = mid + 1;
    } else {
        ans = expectedVal;
        high = mid - 1;
    }
}
return ans;
```

Binary Search - lecture 7

~~HARD PROBLEMS~~

- ① Median of 2 Sorted Arrays
 - same size
 - Different size
- ② kth smallest Element → 2 sorted Arrays
- ③ Median → Row wise sorted Matrix

Median of 2 sorted Arrays (Same Size)



① Brute force: → Merge

2 6 7 9 10 (N)

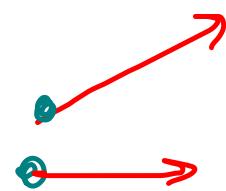
1 3 5 8 8 (N)

1 2 3 5 $\boxed{6 \ 7}$ 8 8 9 10 ($2N$)

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

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

Space: $O(2n)$



$$am = bm : \frac{amt\,bm}{2}$$

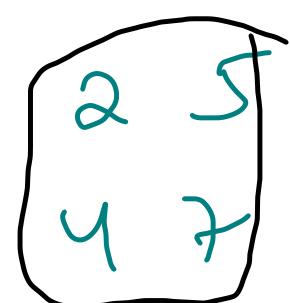
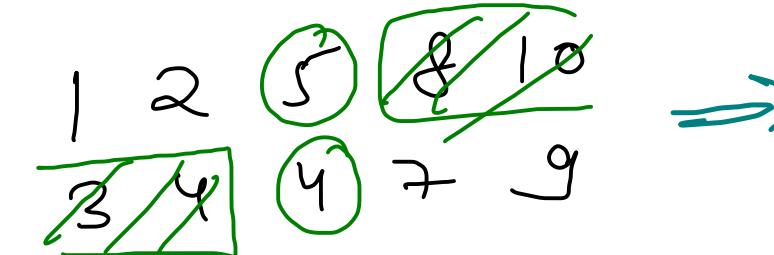
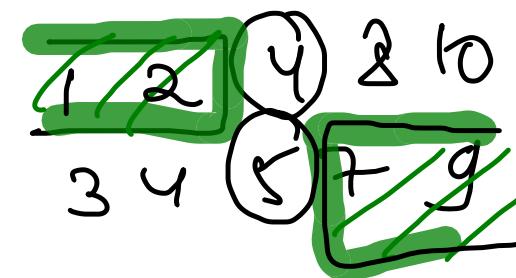
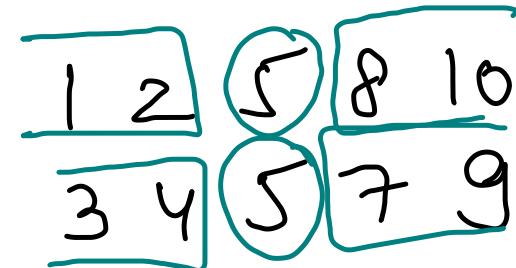
median]

$$am < bm : \begin{array}{l} \text{discard} \\ \text{left in a} \\ \text{discard} \\ \text{right ind} \end{array}$$

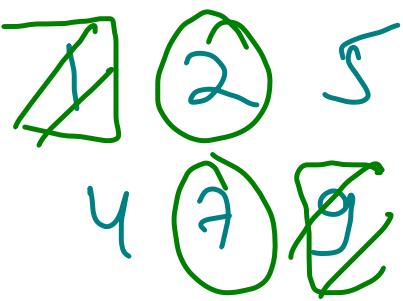
• discard right in a
discard left in 5

1 2 (3) 4 5 : 3

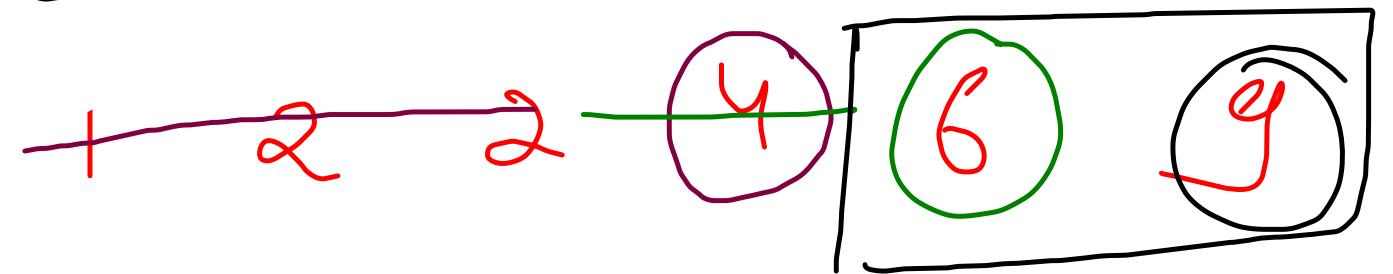
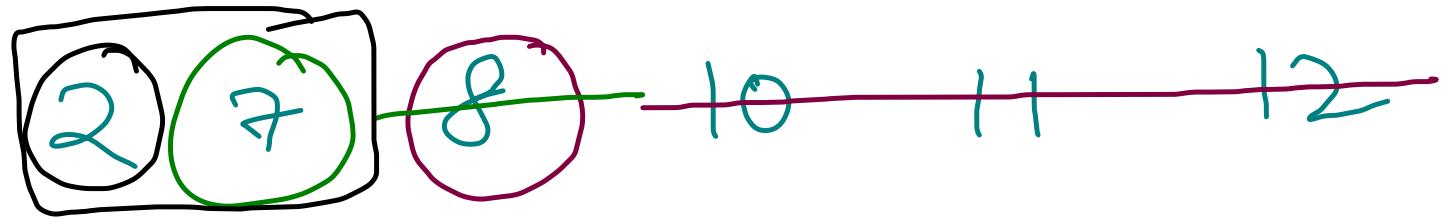
1 2 (3 4) 5 6 : 3 5



2, 4, 5, 7

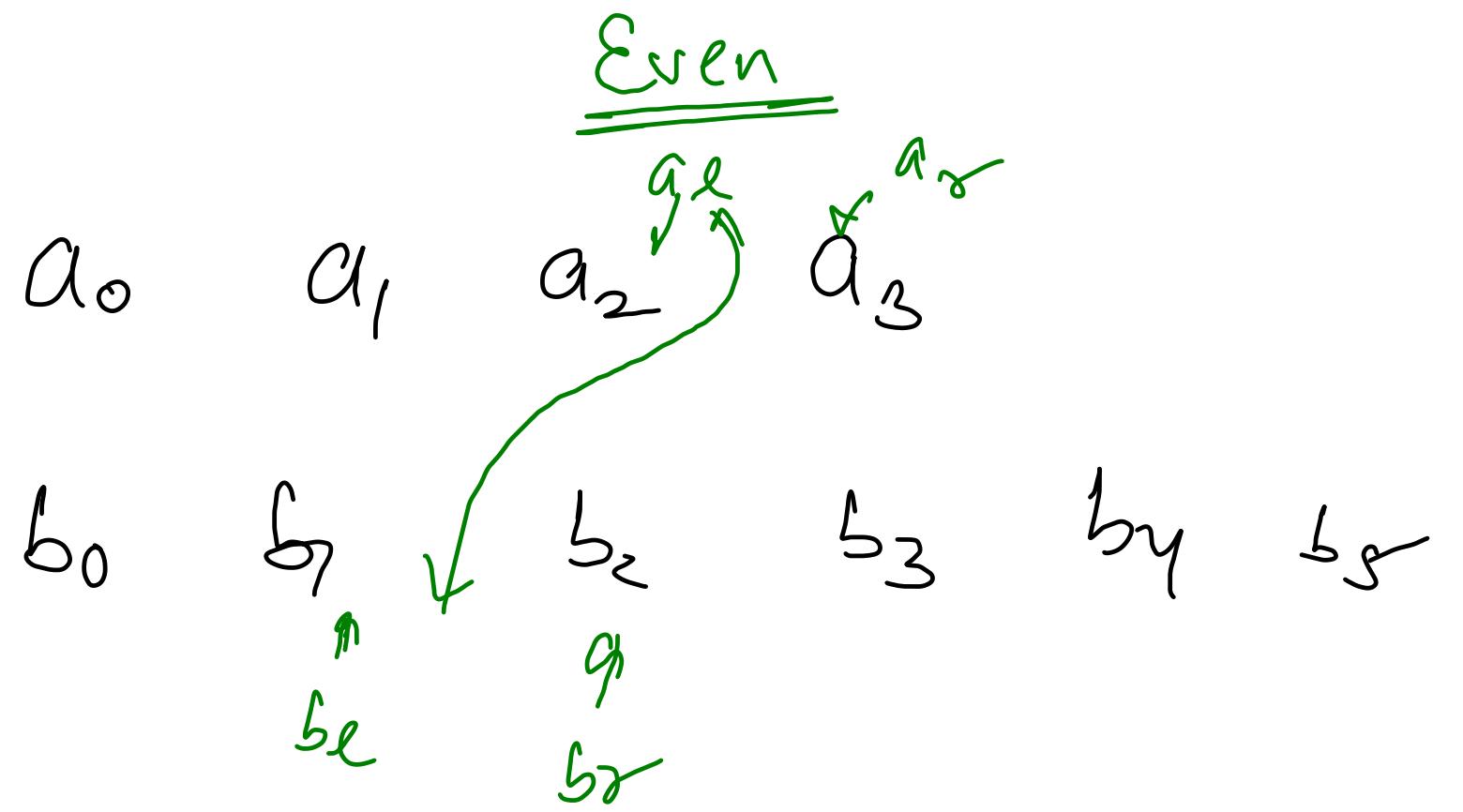


$O(\log_2 N)$



0 1 2 3 4 5

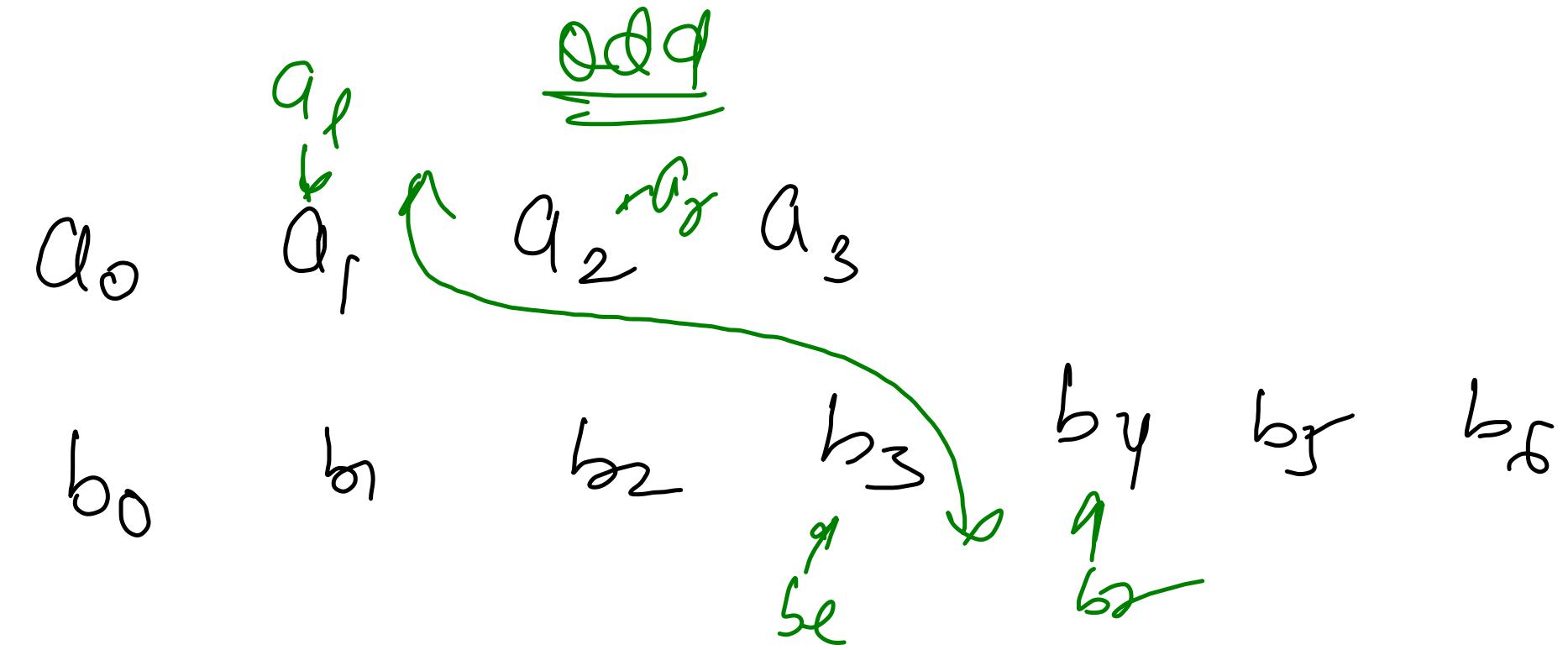
Merge 2 sorted Arrays (Different Size)



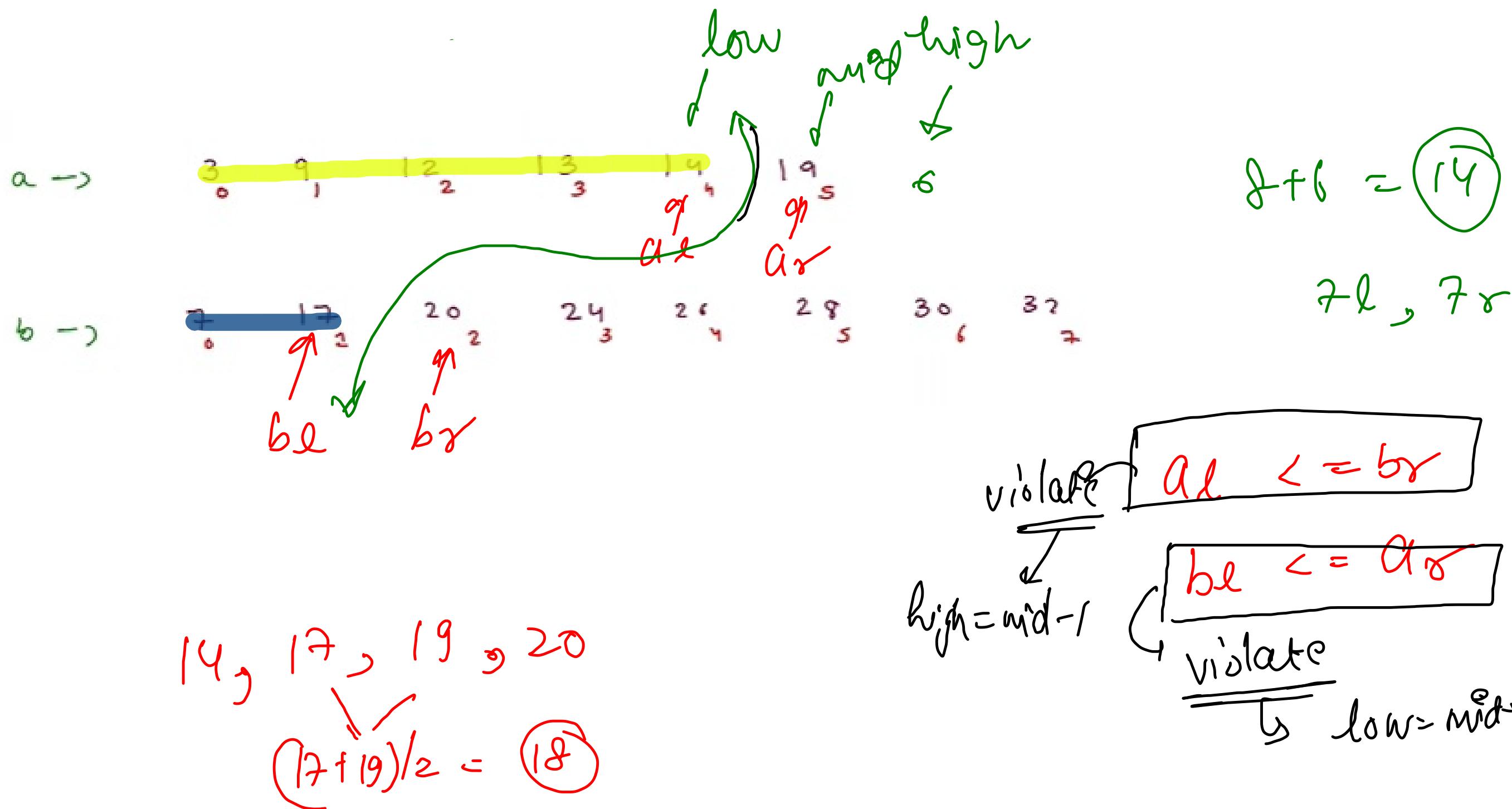
$$\frac{4+6}{2} = \frac{10}{2} = 5$$

Median = $\{a_1, b_1, a_2, b_2\}$

Median of above sorted array



$$\text{Median} = \max(a_l, b_l)$$



Corner case ①

0
5
10
12
14

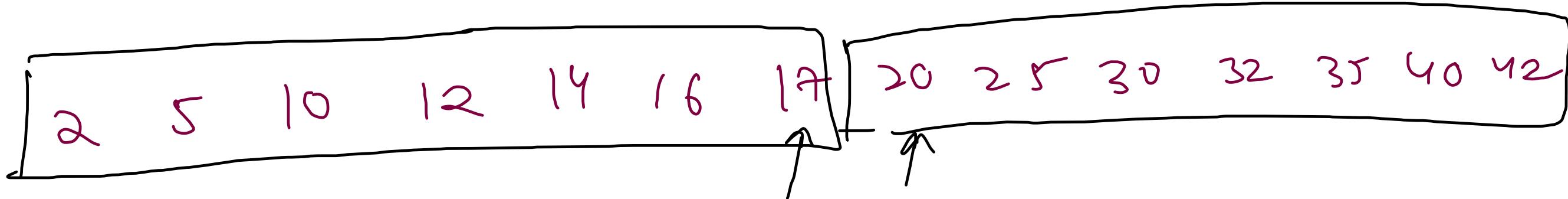
a_l
 a_r
low
high
mid

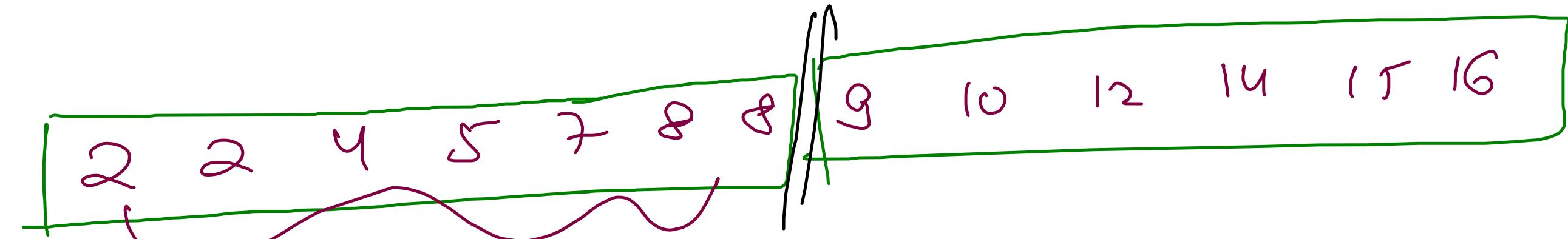
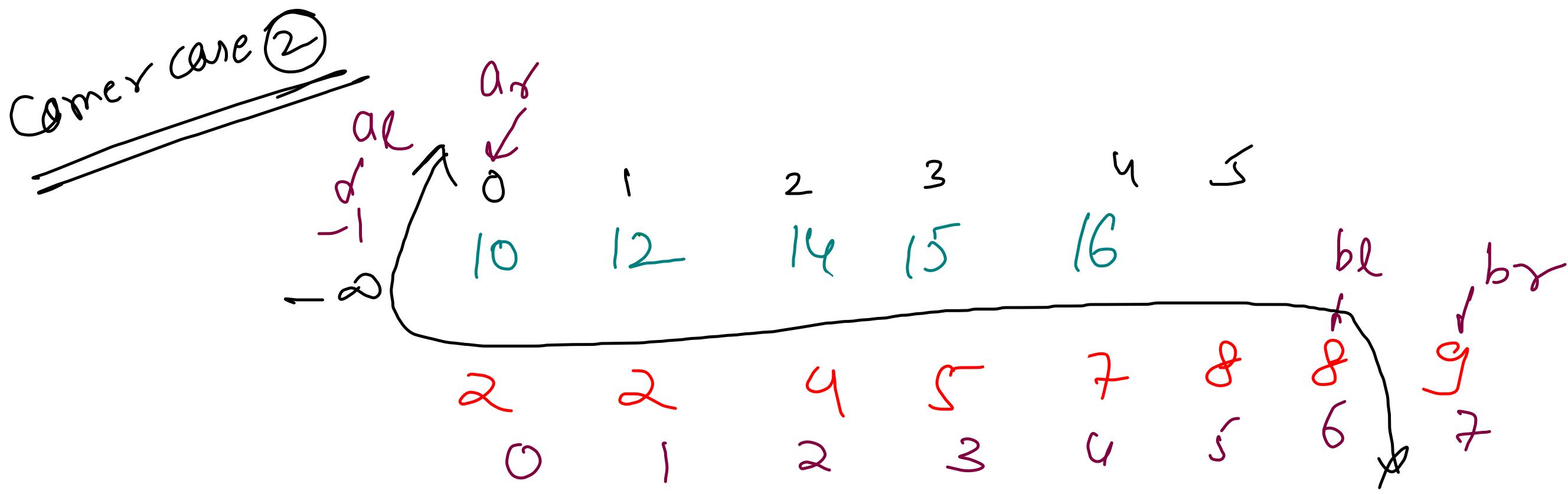
16

2
17
20
25
30
32
35
40
42
0
1
2
3
4
5
6
7
8

$\alpha_{log_2 N}$

$\alpha_x = \text{over length}$





$$7 - \varnothing = 7$$