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

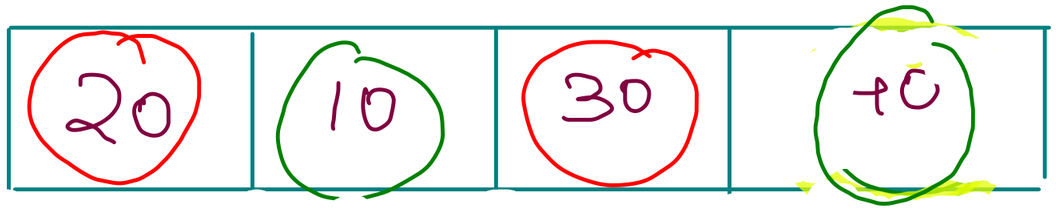
(3) Aggressive Cows
(4) Wood cutting - Eko
} SPOJ

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 stud = 2



Minimize the max^M no of pages

1. Each stud should take atleast 1 book
2. we should allocate each book to exactly 1 student
3. Continuous allocation
4. Unbreakable item

stud 2

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

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

60 (20, 10, 30) 40 (40) 60

stud 3

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

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

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

10 low 40 low 100 high

Binary Search on pages { ~~Books~~ ~~Array~~ }

books = stud

stud = 1

① low = 40 (Y) high = 100 [40, 100] 70

mid = 70 ans = ~~100~~ 70 ~~70 100~~

② low = 40 high = 69 [~~54~~ 40, 69]

mid = 54 (N) ans = 70

③ low = 55 high = 69 [55, 69] 62

mid = 62 (Y) ans = ~~70~~ 62

④ low = 55 high = 61

mid = 58 (N) ans = 62

⑤ low = 59 high = 61

mid = 60 (Y) ans = 60

low = max of array

high = sum of array

20	10	30	40

stud = V ≠ 3

pages = {(0+10)(30)(40)

stud = 2

⑥ low = 59 (N)

high = 59

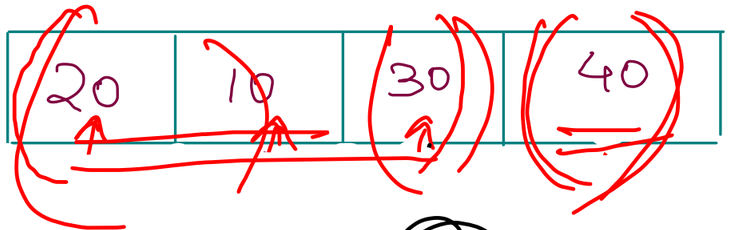
mid = 59 ans = 10

⑦ low = 60

high = 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;
```



Stud = 3

low = 40

high = 100 → 53

mid = 70 → 54 → ... → 40

ans = 100

isPossible

high - low

$O(2n + N * \log_2(sum - max))$

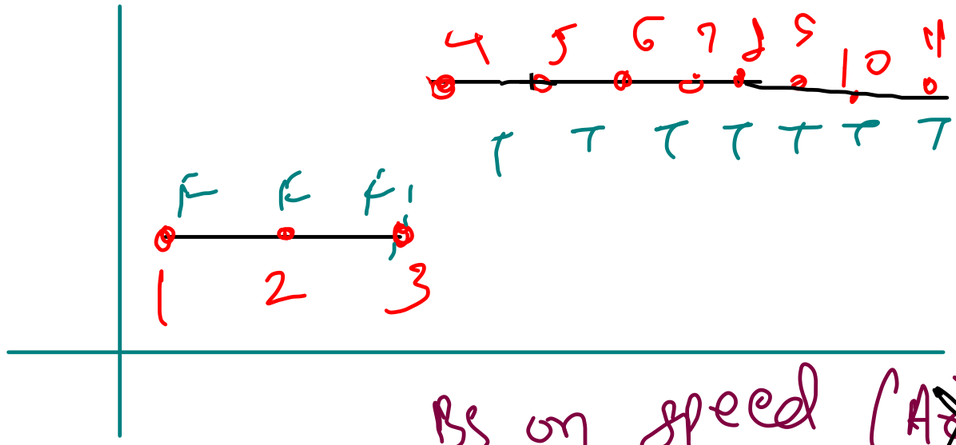
↳ low & high

```java
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;
}
```

```java
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



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.

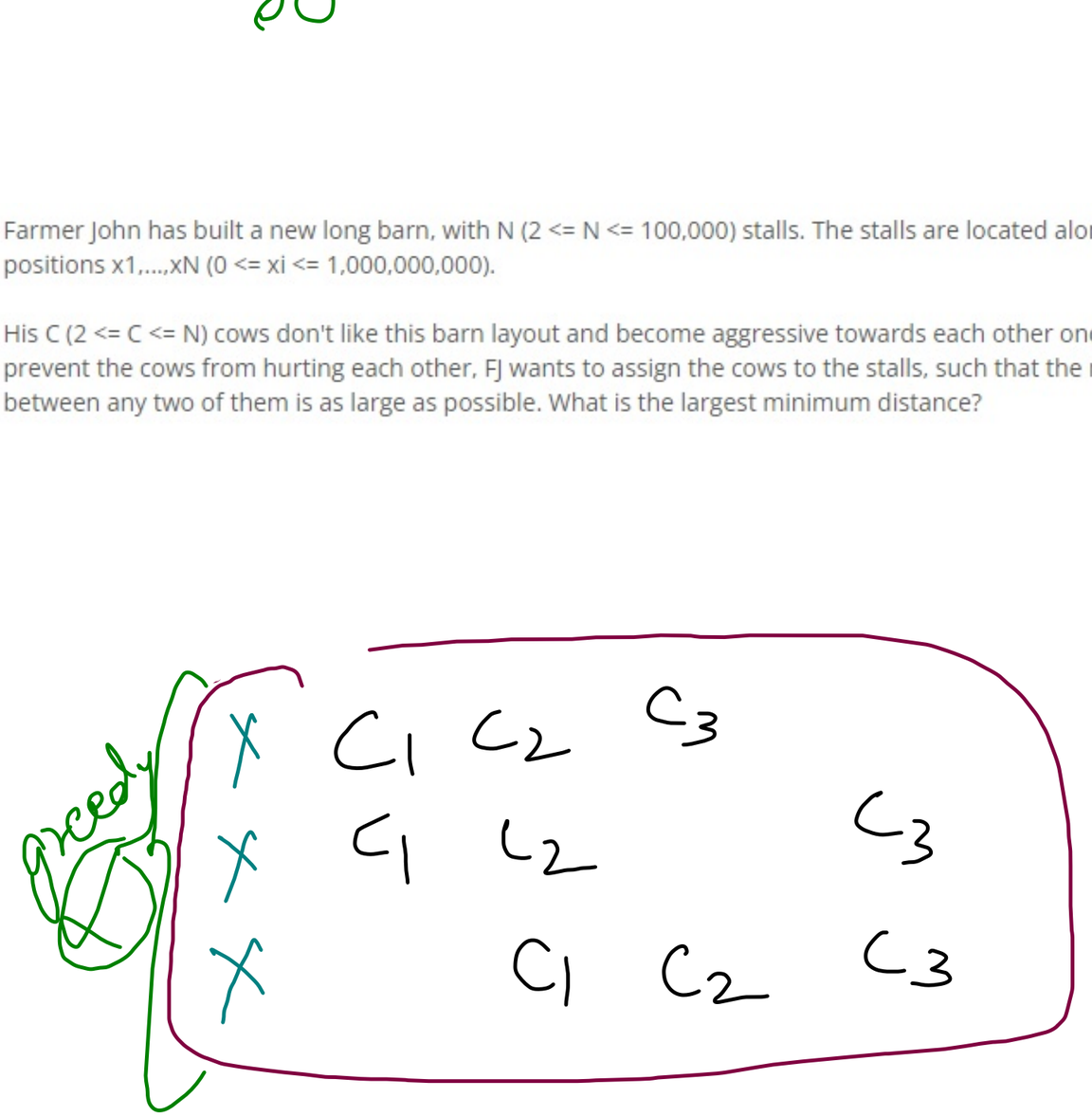
Koko likes to eat slowly but still wants to finish eating all the bananas before the guards return.

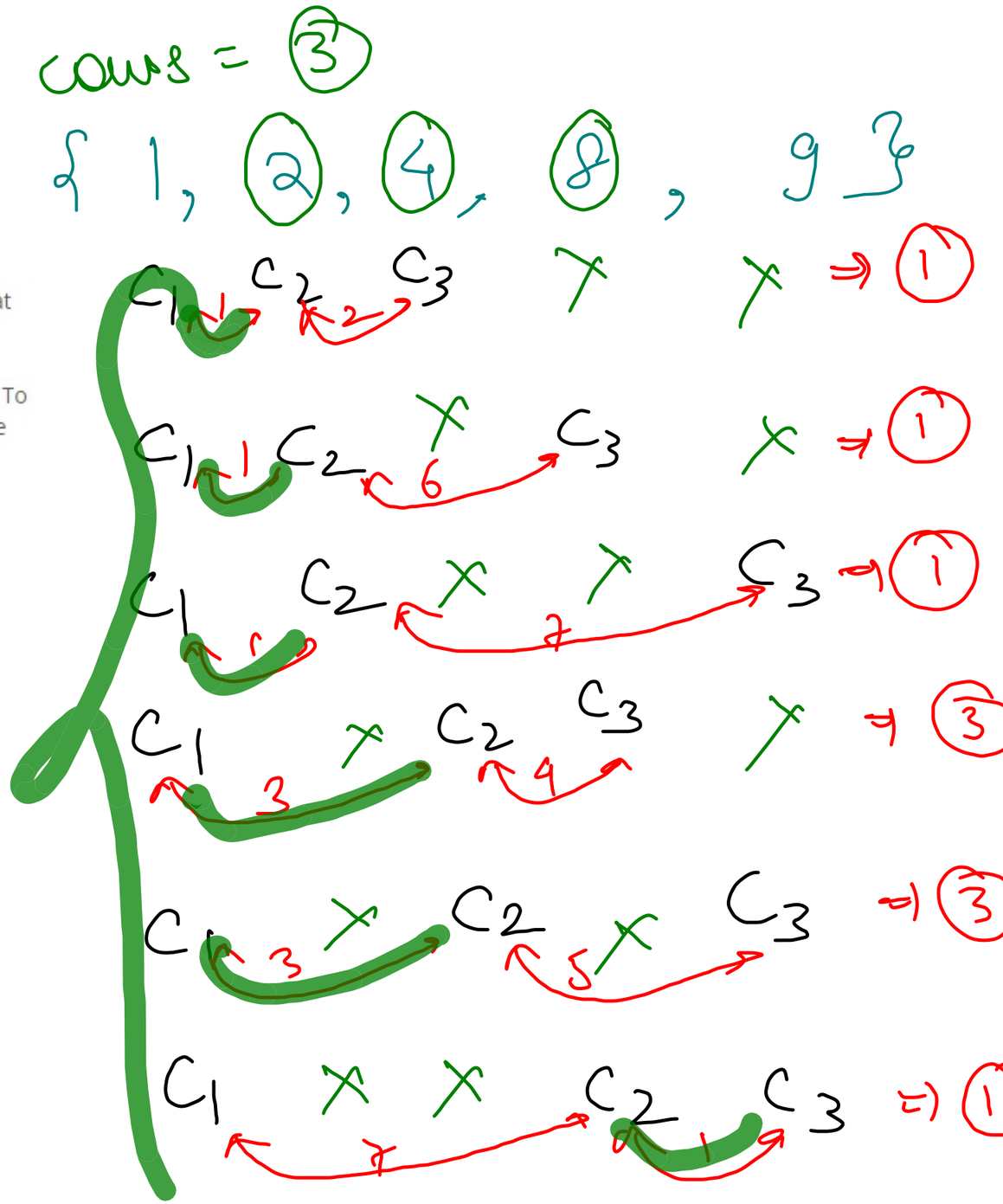
Return *the minimum integer* `k` *such that she can eat all the bananas within* `h` *hours.*

Aggressive Cows

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?

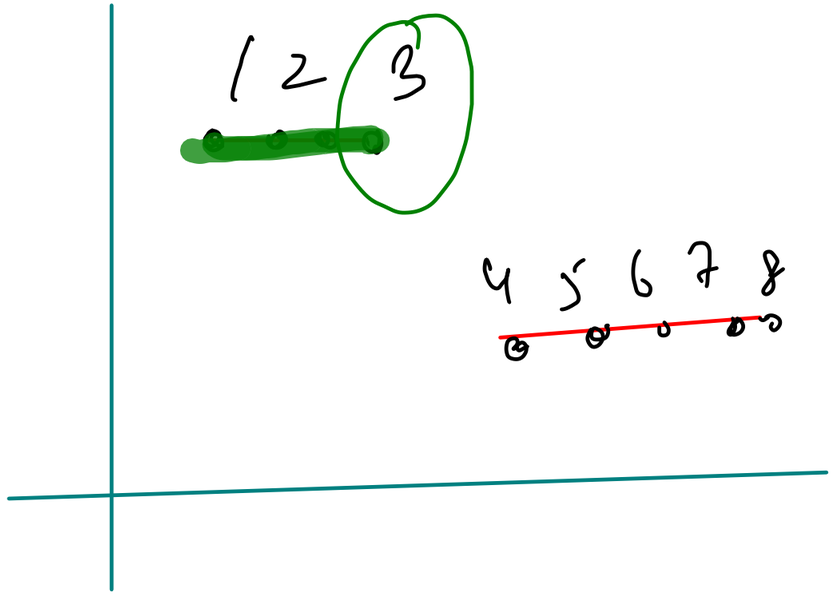




Binary Search on Adjacent distance

k = 3

1, 2, 4, 8, 9



Minimum adj. distance

low = 1

high = 8

1	Yes	$c_1 c_2 c_3 x x$
2	Yes	$c_1 x c_2 c_3 x$
3	Yes	$c_1 x c_2 c_3 x$
4	No	✗
5	No	✗
6	No	✗
7	No	✗
8	No	✗

if cows = 2

arr[n-1] - arr[0]

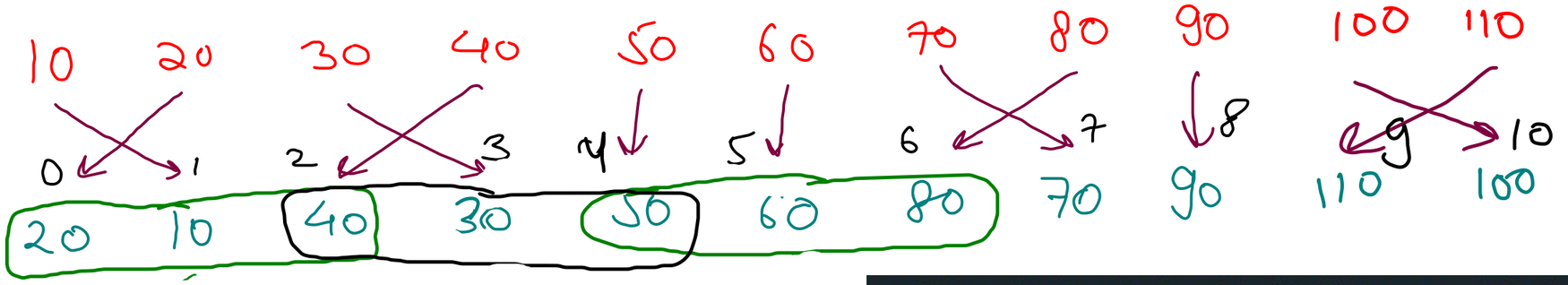
Binary Search - lecture (5)

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



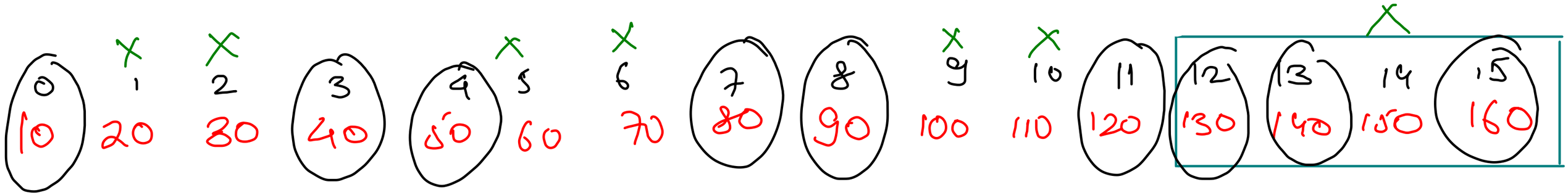
$O(\log_2 N * (2K+1))$

```java
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

LS ⇒ O(N)
BS ⇒ O(log₂N)

target = 140



Best case ⇒ First window → First ele O(1) (10)
→ last ele O(k) (60)

Total window = $\left\lceil \frac{N}{K} \right\rceil$

$= \sqrt{N}$

k = window
N = no of elements

worst case ⇒ $\left\lceil \frac{N}{K} \right\rceil + K$ → Linearch search in last window

skipping $\frac{N}{K}$ windows

$k = \sqrt{N}$ ⇒ $\left\lceil \frac{N}{\sqrt{N}} \right\rceil + \sqrt{N} = 2\sqrt{N}$

⇒ $O(\sqrt{N})$

Step Search

{ Index X → Property }

{ Element/value → Property }

target = 28

k = 3

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

2 3 1 3 2 2 2 3

k = 3

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

2 3 2 3 3 1 3 2 1

target = 14

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

24 21 20 18 16 17 14

target = 14

```java
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;
}
```

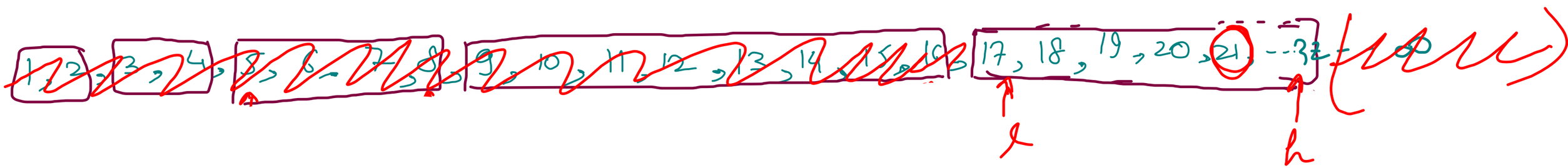
worst case ⇒ O(N)

avg case ⇒ O(√N)

best case ⇒ O(1)

(Infinite Sorted Array) Unbounded Binary Search

target = 21



[1, 2]

[3, 4]

[5, 8]

[9, 16]

[17, 32]

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

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

```java
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);
    }
}

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

240. Search a 2D Matrix II

Row → Sorted
Column → Sorted

top right.

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

$O(N + M)$ → worst case

① Brute force
Binary search on each row i.e. $O(n \log m)$

② Staircase search
↳ Divide & Conquer

target = 18

arr[mid] > target (col--)
discard column

arr[mid] < target
discard row++

unsuccessful search
→ out of matrix

```
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;
}
```

848 · Minimize Max Distance to Gas Station

k = 2

3 6 7 14 11 23

[3,6,12,19,33,44,67,72,89,95]

```cpp
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;
}
```

```cpp
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;
}
```

low = 3 high = 23

mid = 27/2 = 13 N

mid = 18 Y

mid → 14+17 → 15 Y

mid = 17 Y

Binary Search - lecture (6)

① Maximum Element in Bitonic Array

② Search Element in Bitonic Array

} Bitonic Array

③ Peak Element

④ Unique Element - Sorted Array

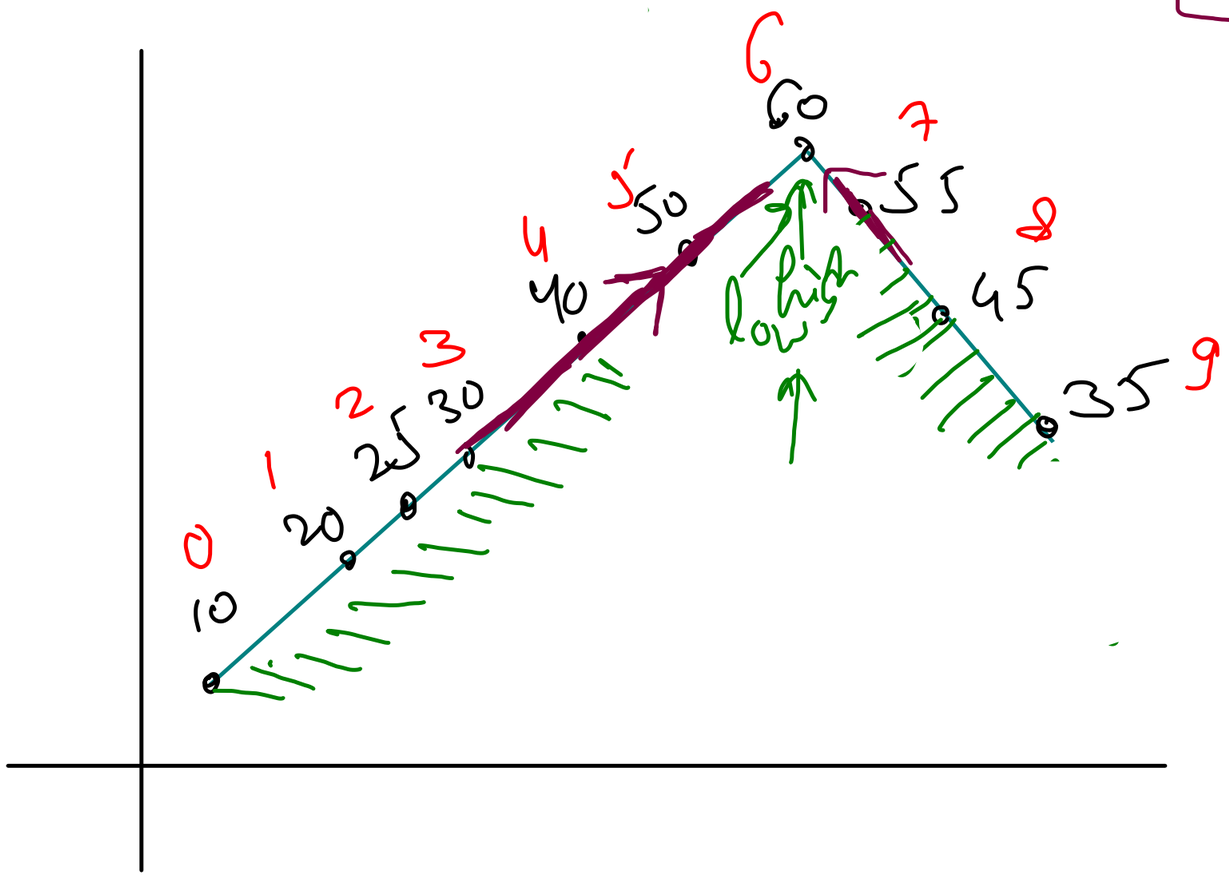
⑤ Missing Element of Arithmetic Progression

Bitonic Array

(Peak Element) → Maximum

↳ Brute force : Linear search
O(N)

↳ Binary search



Strictly increasing
then
Strictly decreasing

You are screen sharing Stop Share

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 nums[-1] = nums[n] = -∞.

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

→ nums[-1] = nums[n] = $-\infty$

→ No two adjacent elements are equal

→ There will be atleast one peak element.

→ Modified Binary Search can be applied.

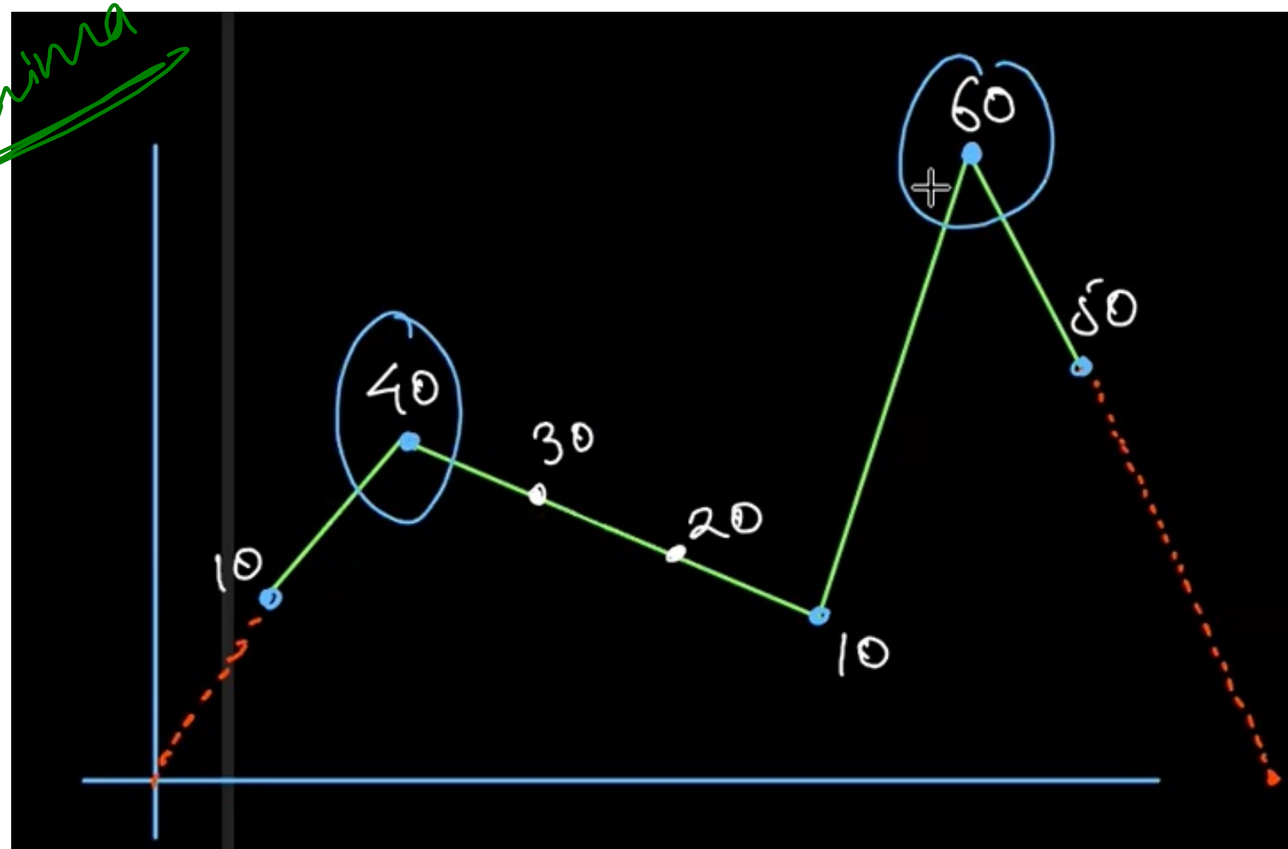
-2^{63}

log. MIN-VALUE

local maxima

0 1 2 3 4 5 6

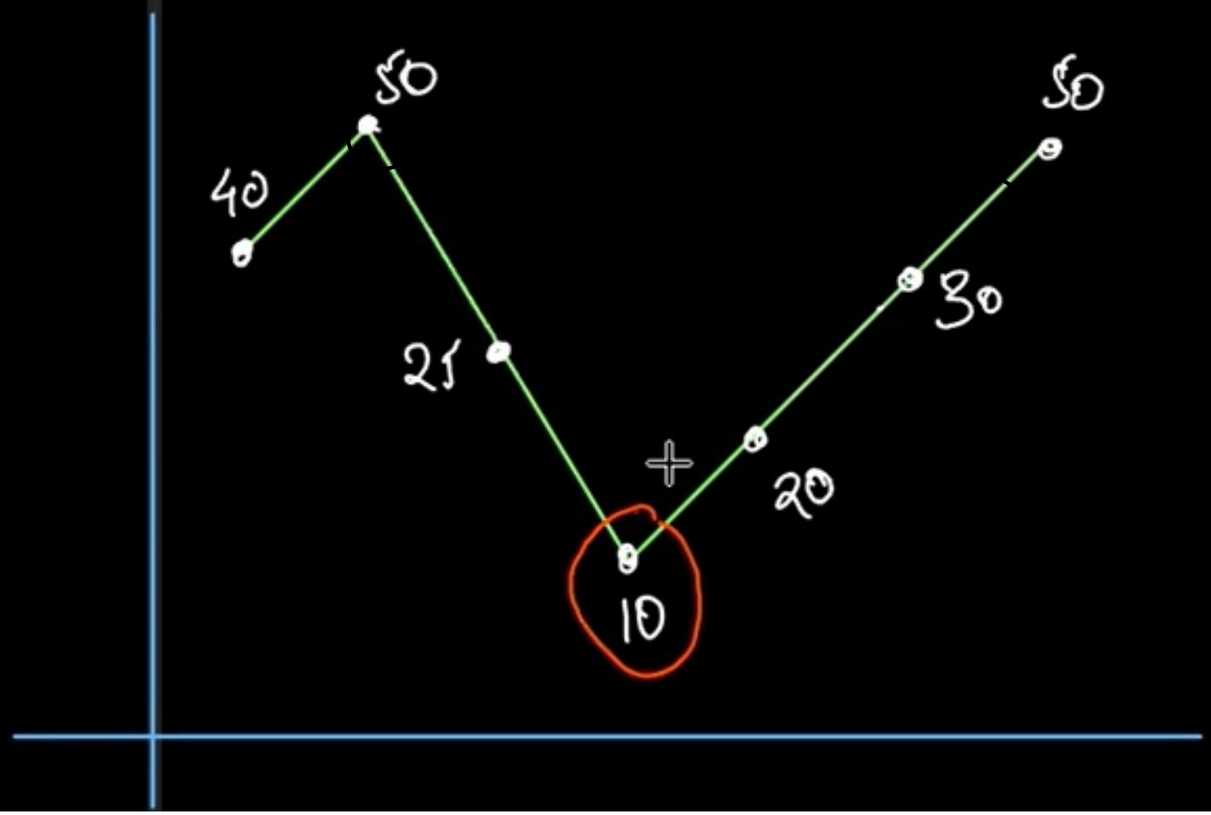
{ 10, 40, 30, 20, 10, 60, 50 }



Case (2) Unsorted Array

{More than one peak}

Case (5) :→ Mid on valley



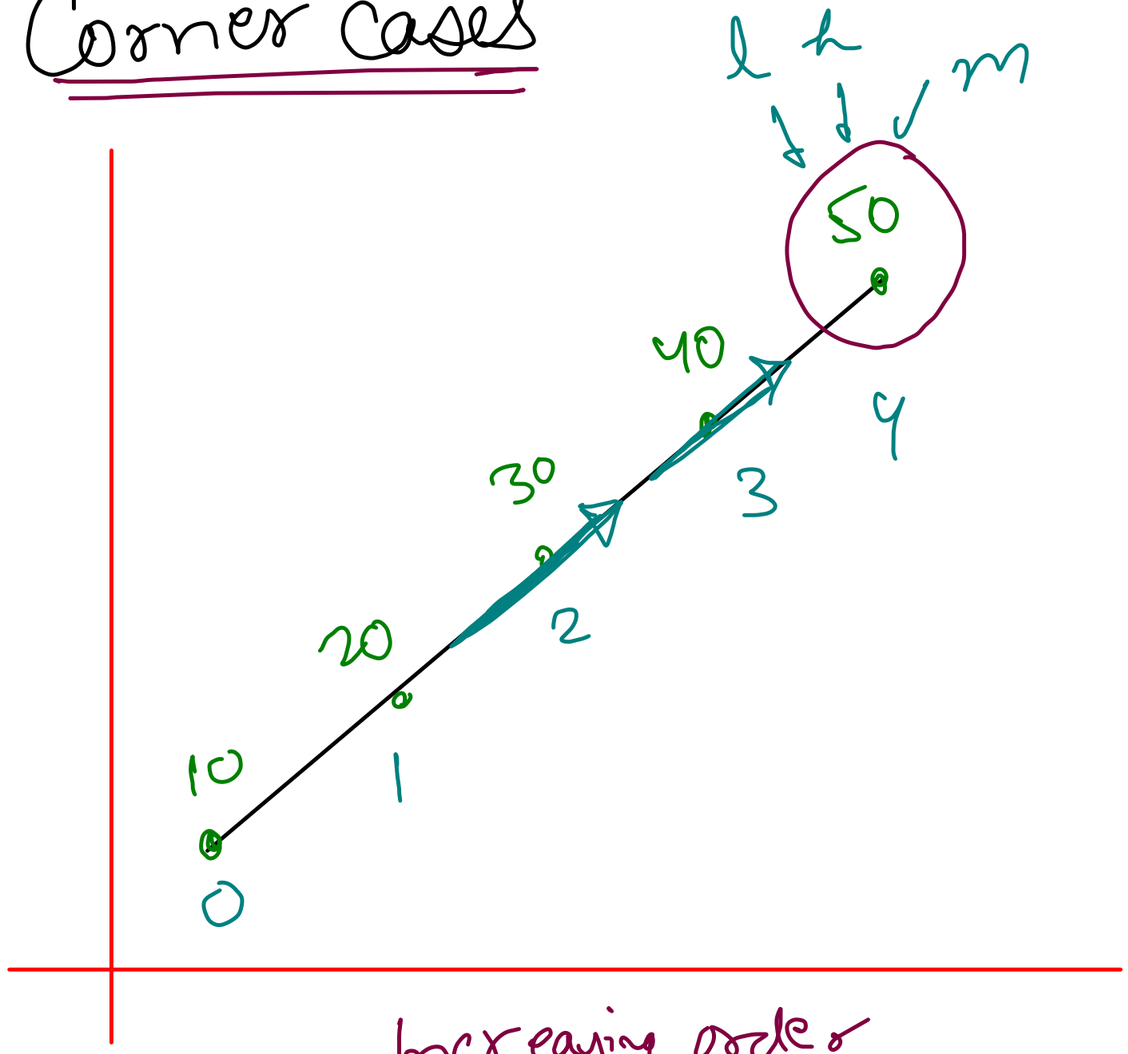
{ 40, 50, 25, 10, 20, 30, 50 }

l

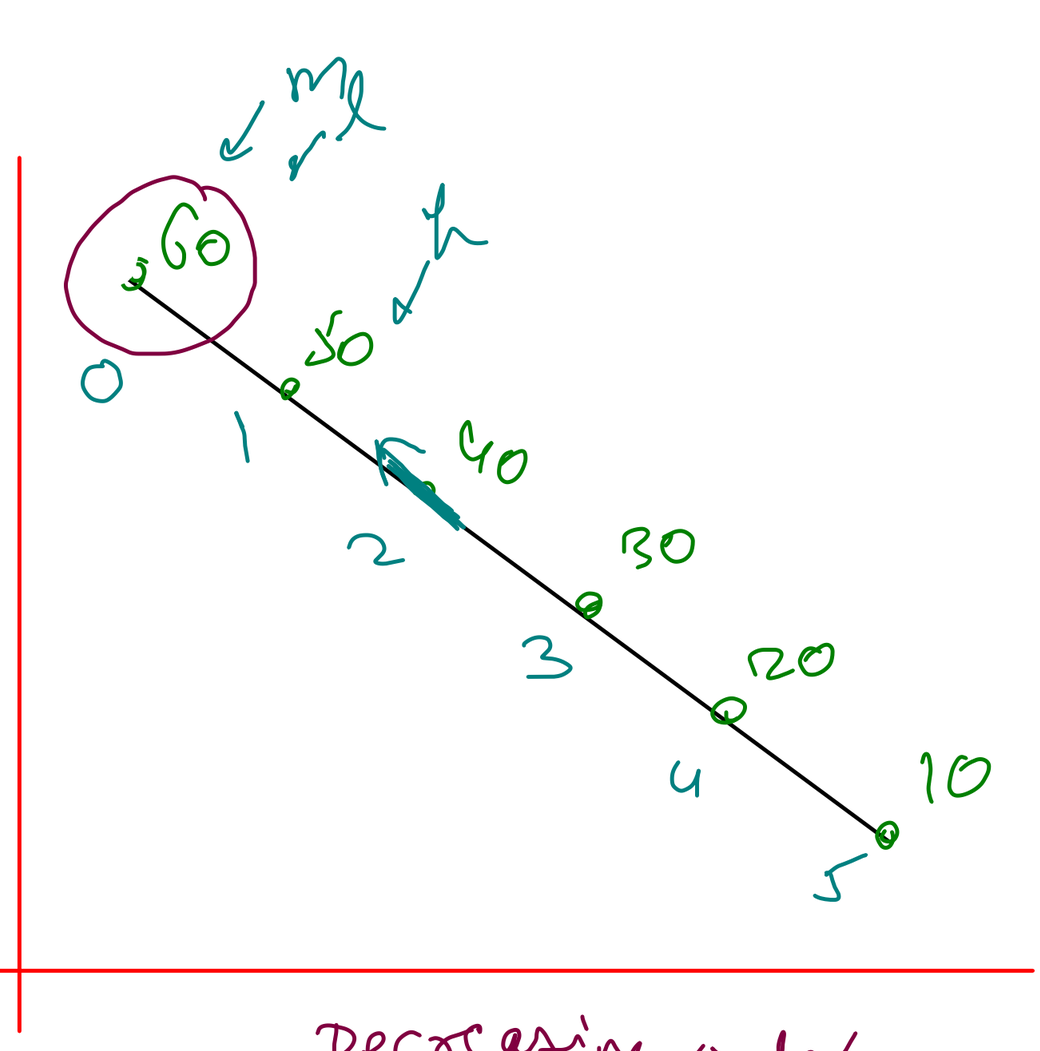
m

r

Corner cases



Increasing order

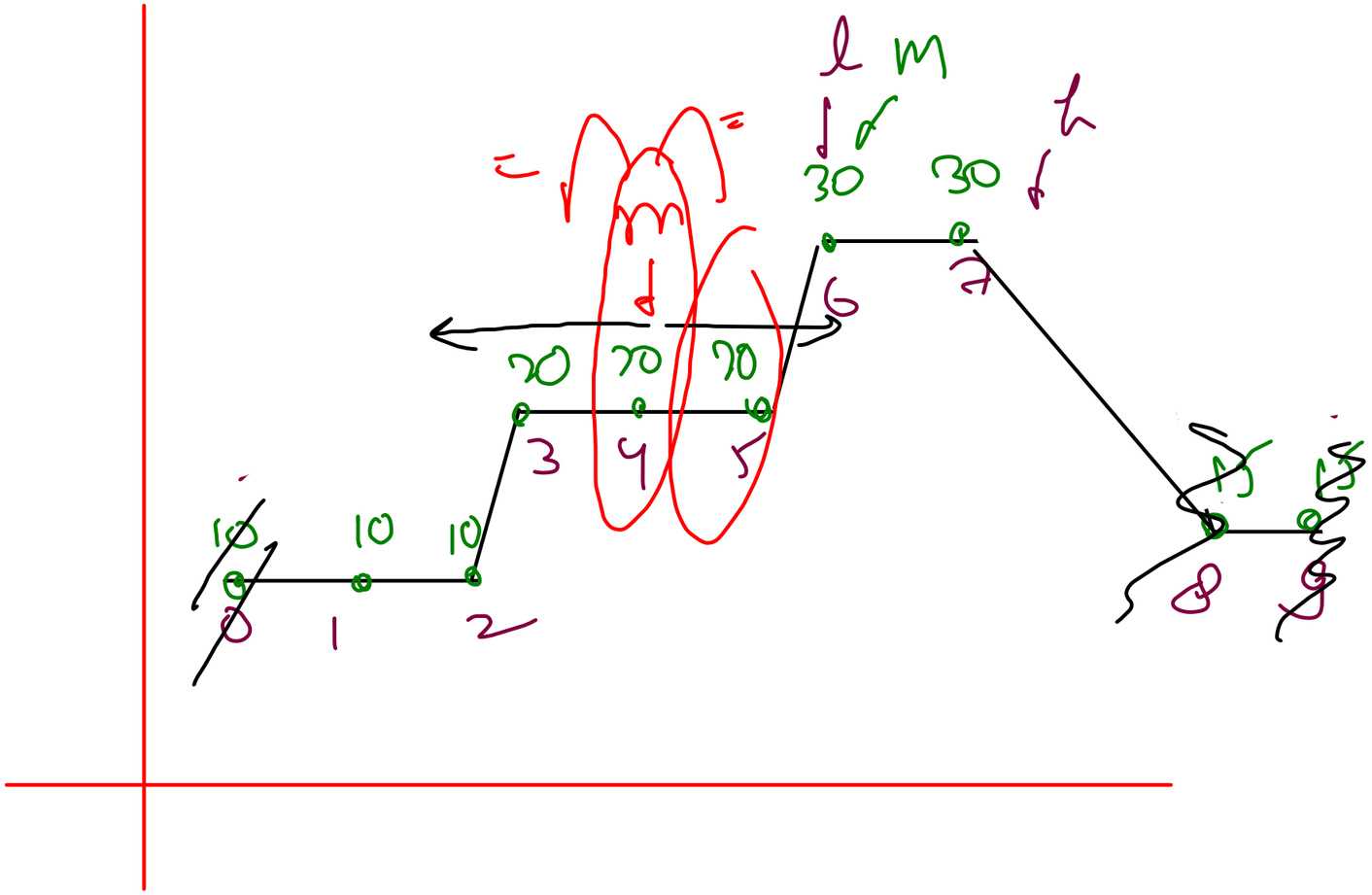


Decreasing order

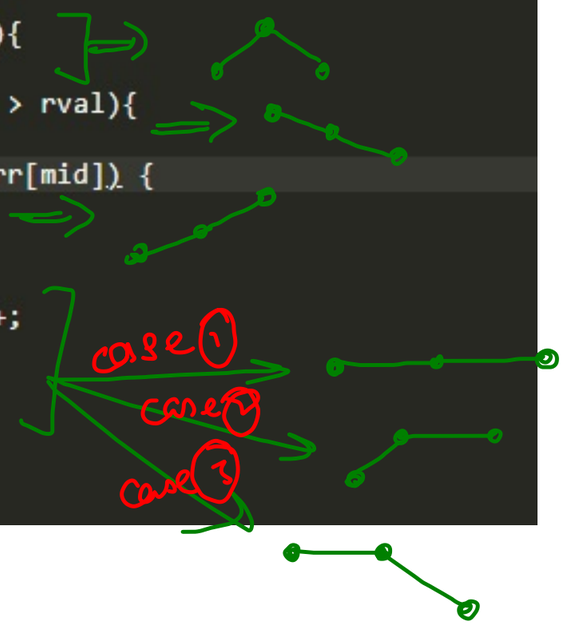
```java
public int peakIndexInMountainArray(int[] arr) {
    int low = 0, high = arr.length - 1;
    while(low <= high){
        int mid = low + (high - low) / 2;

        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



```java
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;
}
```



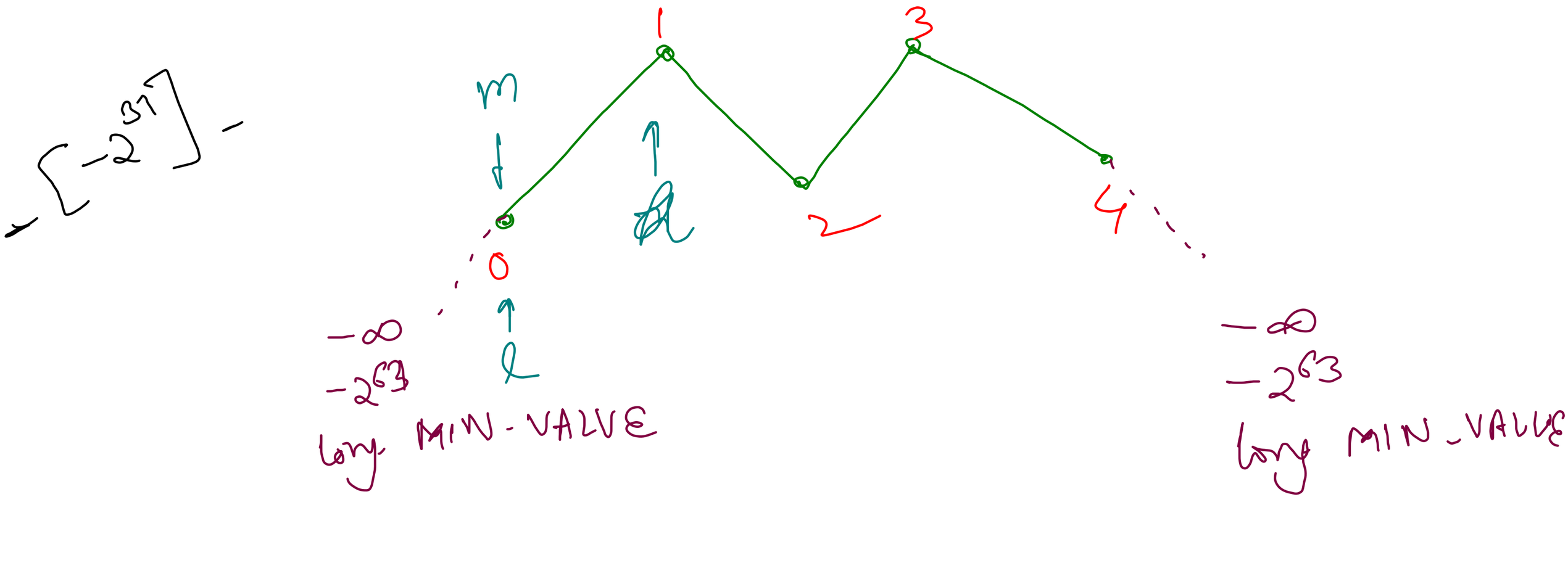
Handle Duplicates

↓

Worst case O(N)

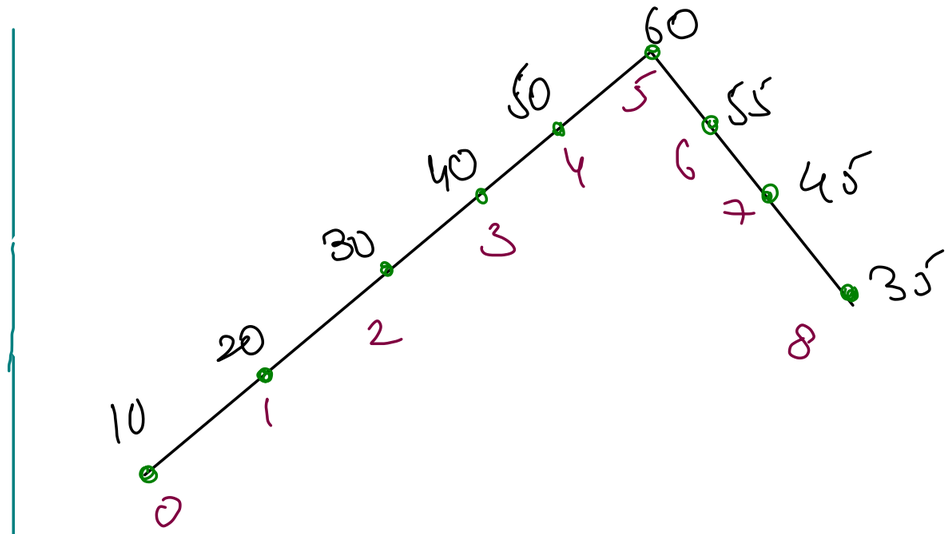
Corner case

$$[-2^{31}, 0, -2^{31}, 0, -2^{31}]$$



```java
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

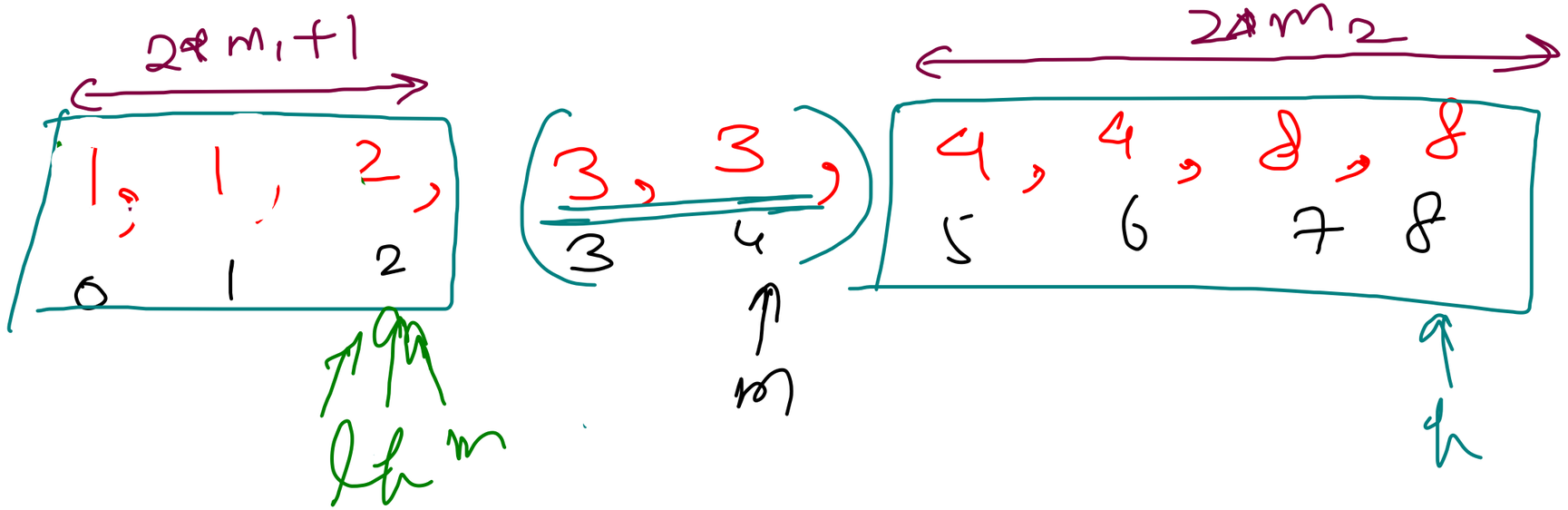
```
public int solve(int[] A, int B) {
    int peakIdx = peakIndexInMountainArray(A);
    if(A[peakIdx] == B) return peakIdx;
    if(A[peakIdx] < B) return -1;
    int left = binarySearch(A, 0, peakIdx - 1, B, true);
    if(left != -1) return left;
    return binarySearch(A, peakIdx + 1, A.length - 1,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

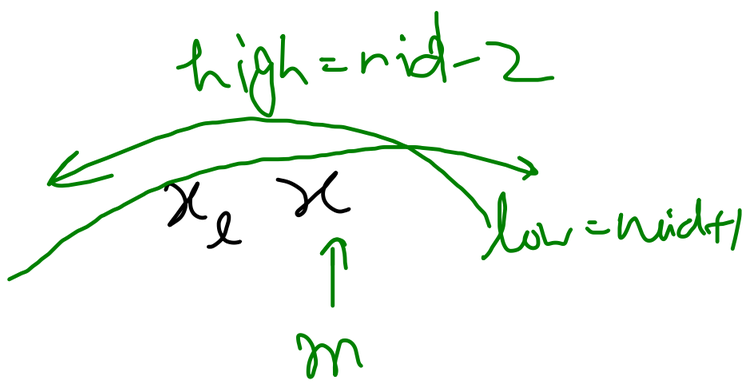
```
public int binarySearch(int[] A, int left, int right, int target, boolean isInc){
    if(left > right) return -1;
    int mid = left + (right - left) / 2;

    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

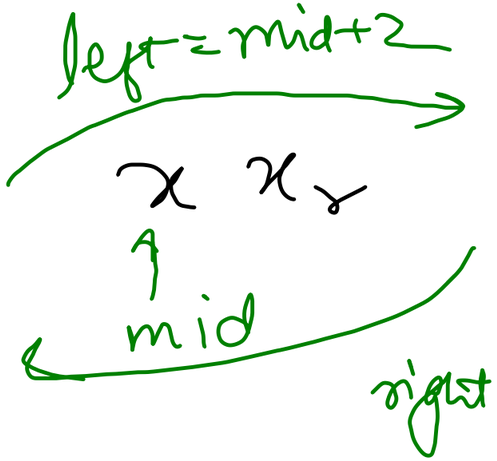
$O(\log_2 N)$





left range = m-l-1

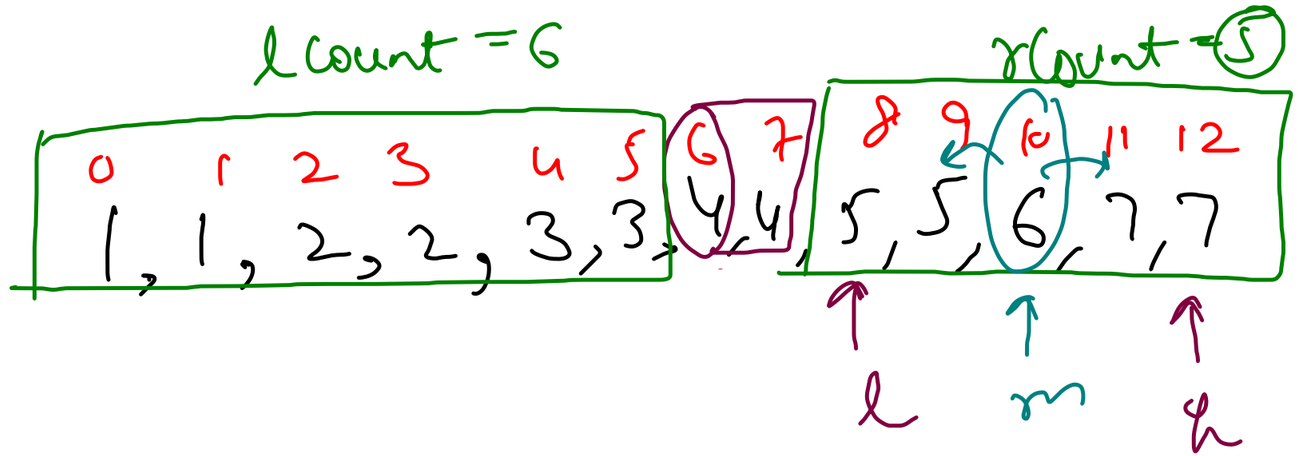
right range = h-m



left range = m-l

right range = h-m-1

```java
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

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

min = a

q q

h l

a+nd = 2 + 2*12 = 6

0	1	2	3	4	5	----	n-1
a	a+d	~~a+2d~~	a+3d	a+4d	a+5d		a+(n-1)d

max
↳ a+nd
2 + 9*2 = 2+18 = 20

common diff
= 20 - 2 = 18
g
= 2

= max - min / no of ele

= a+nd - a / n

= nd / n = d

```
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)

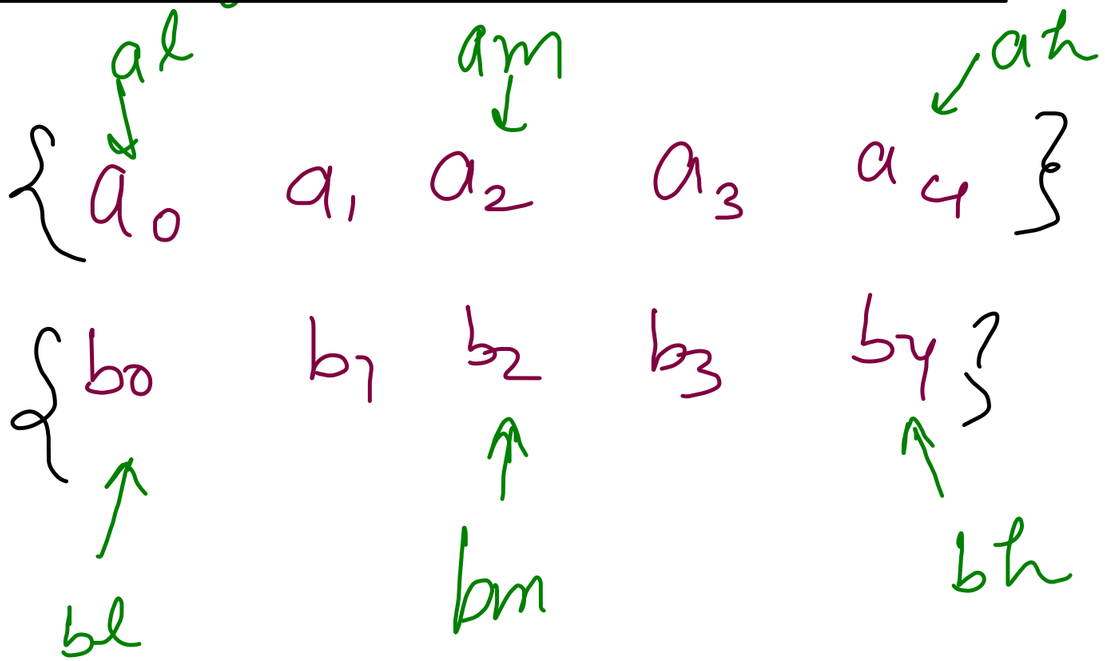
HARD PROBLEMS

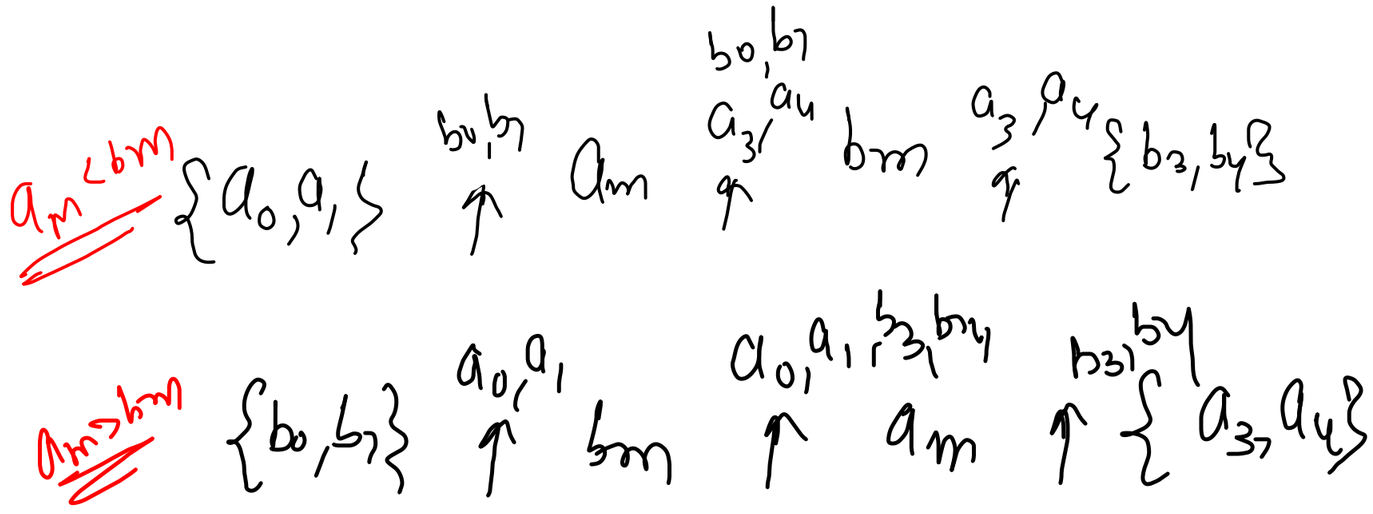
(1) Median of 2 Sorted Arrays
   → Same Size
   → Different Size

(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 6 7 8 8 9 10 (2N)

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

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

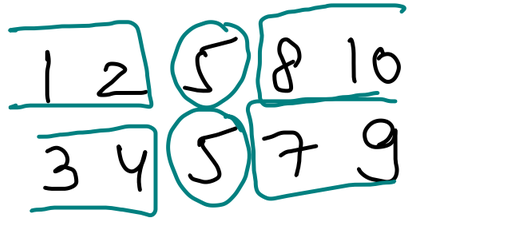
space: $O(2n)$

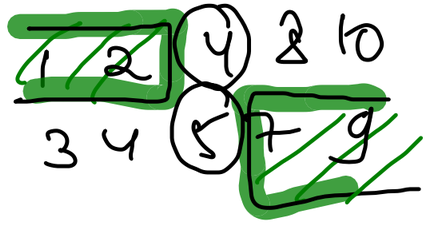
median

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

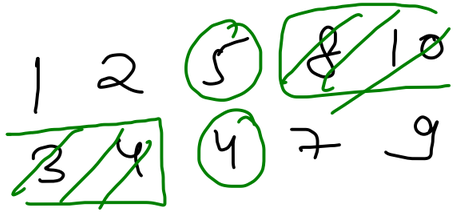
$am < bm$: discard left ind

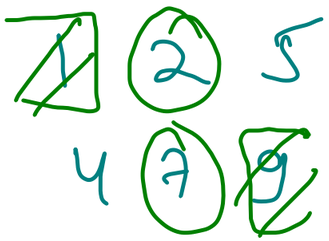
$am > bm$: discard right ind

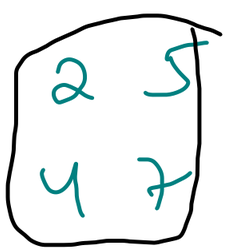




$O(\log_2 N)$



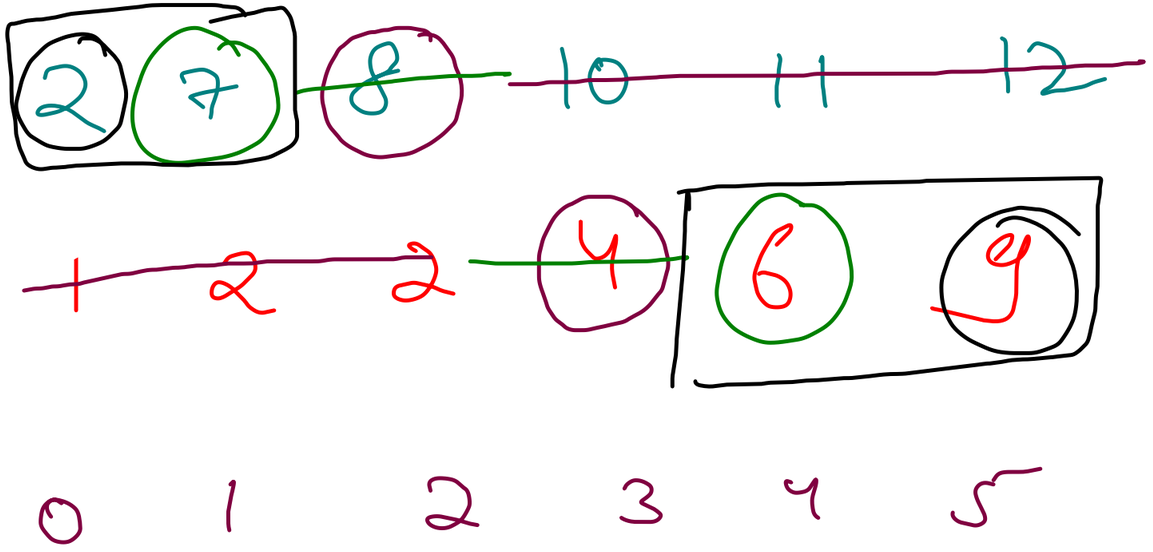




2, 4, 5, 7

1 2 3 4 5 : 3

1 2 3 4 5 6 : 3 5


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

Merge 2 Sorted Arrays (Different Size)

Even

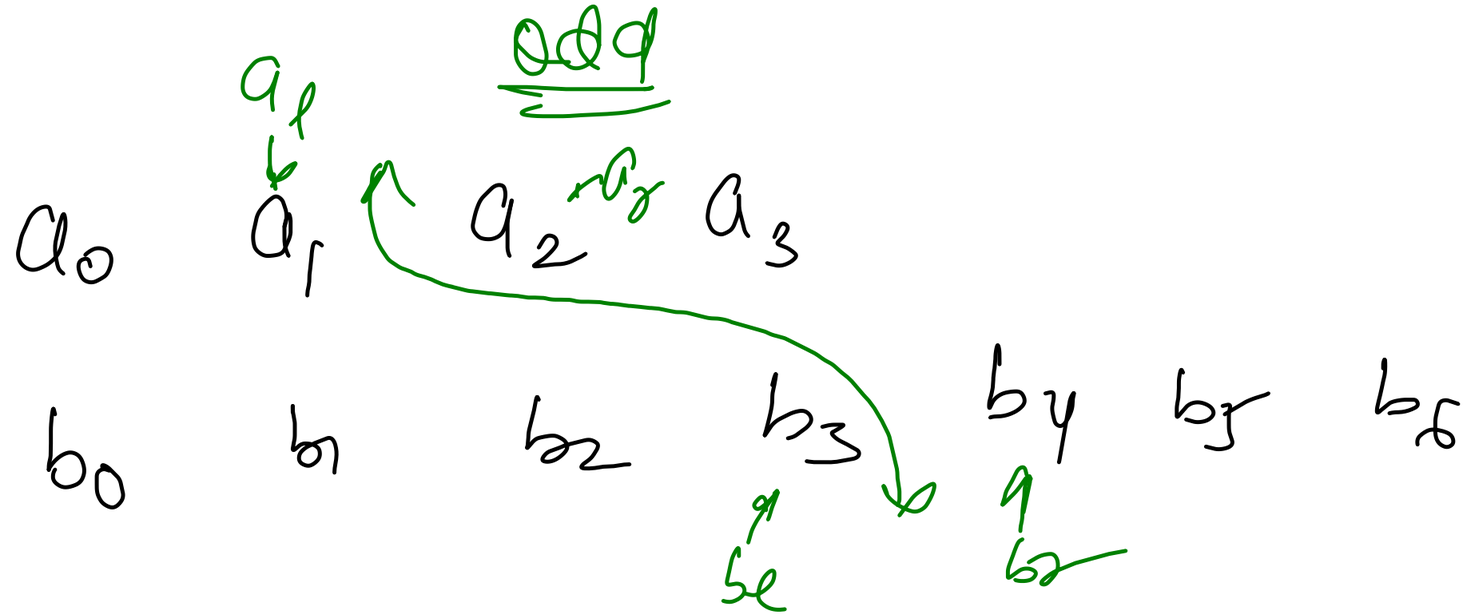
a_0 a_1 a_2 a_3

b_0 b_1 b_2 b_3 b_4 b_5

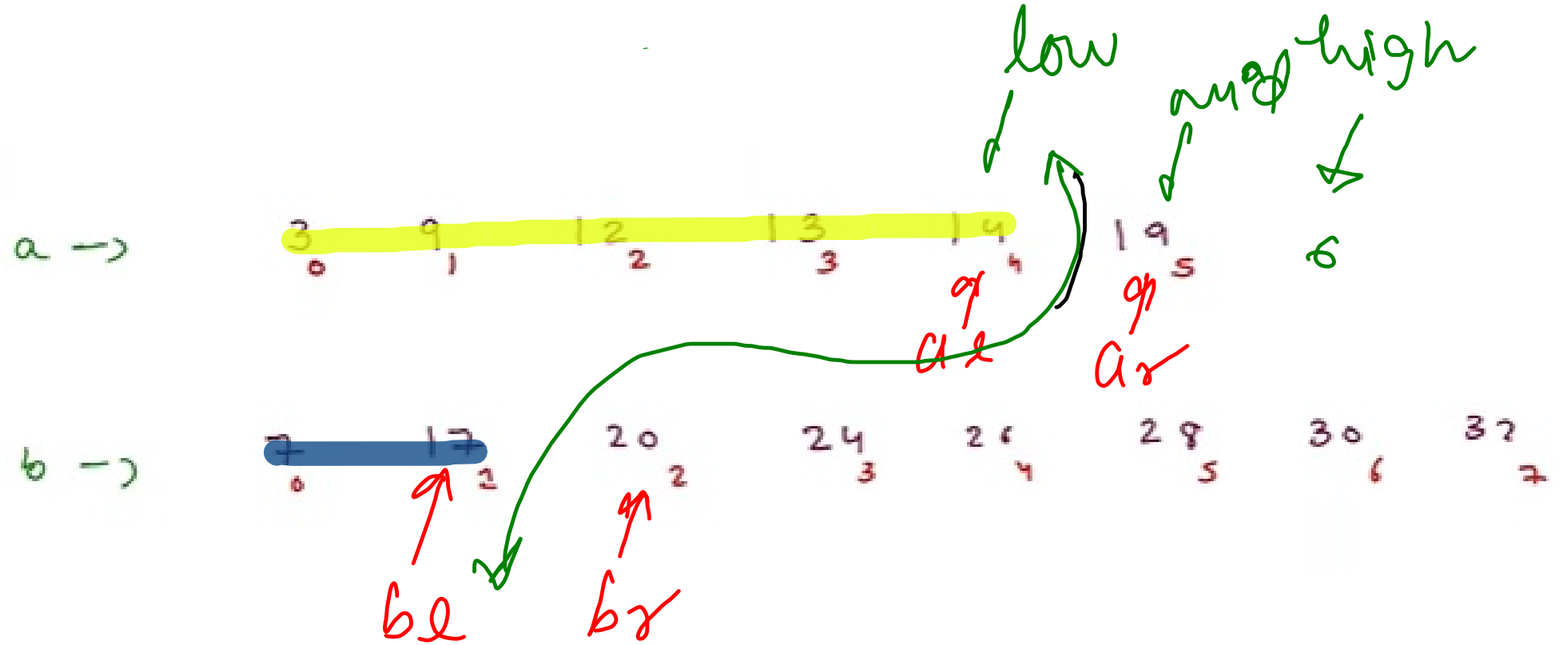
$\frac{4+6}{2} = \frac{10}{2}$ ✓

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

Median of above sorted array



Median = $\max(a_l, b_l)$



14, 17, 19, 20

(17+19)/2 = 18

target = 14

7l, 7r

violate → al <= br

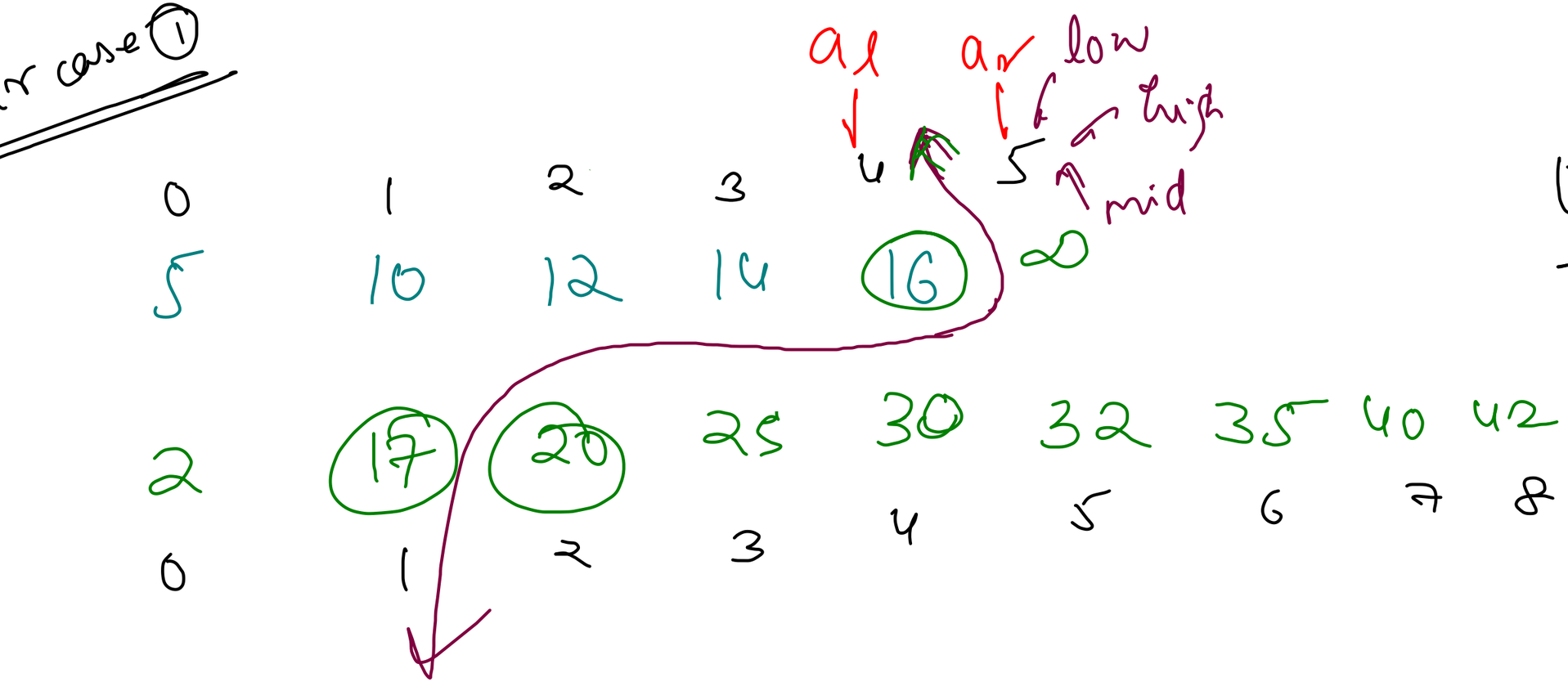
high = mid-1

bl <= ar

violate

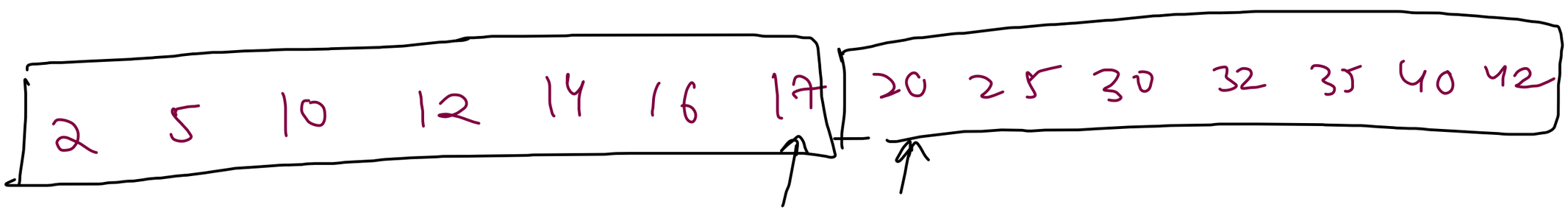
low = mid+1

Corner case ①



$O(\log_2 N_a)$

a_r = arr.length



Corner case ②