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

20	10	30	40

Minimize the maxm 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) (10, 30, 40) 80

(20, 10) (30, 40) 70

(20, 10, 30) (40) 60

stud ③

(20) (10) (30 40) 70

(20 10) (30) (40) 40

(20) (10 30) (40) 40

Binary Search on pages { Books Array }

low = max of array

high = sum of array

20	10	30	40

stud = 1 2

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

books = stud

stud = 1

① low = 40 high = 100 [40, 70, 100]

mid = 70 (Y) ans = ~~100~~ 70

② low = 40 high = 69 [40, 54, 69]

mid = 54 (N) ans = 70

③ low = 55 high = 69 [55, 62, 69]

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

stud = 2

⑥ low = 59 (N)

high = 59

mid = 59 ans = 60

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

mid = 70

ans = 100

70 → 55 → 46 → 42 → 40

100 → 55 → 45 → 41 → 39

70 → 54 → 46 → 42 → 40

isPossible

high - low

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

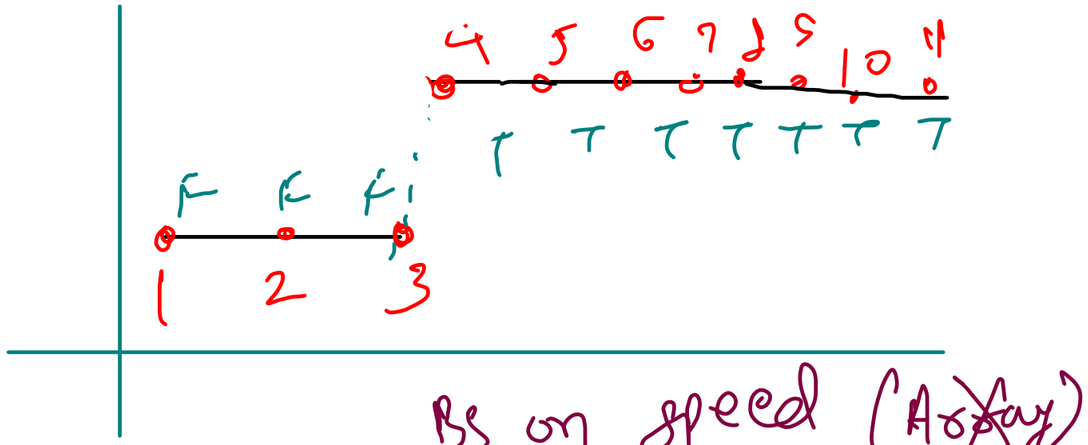
↳ low & high

```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.

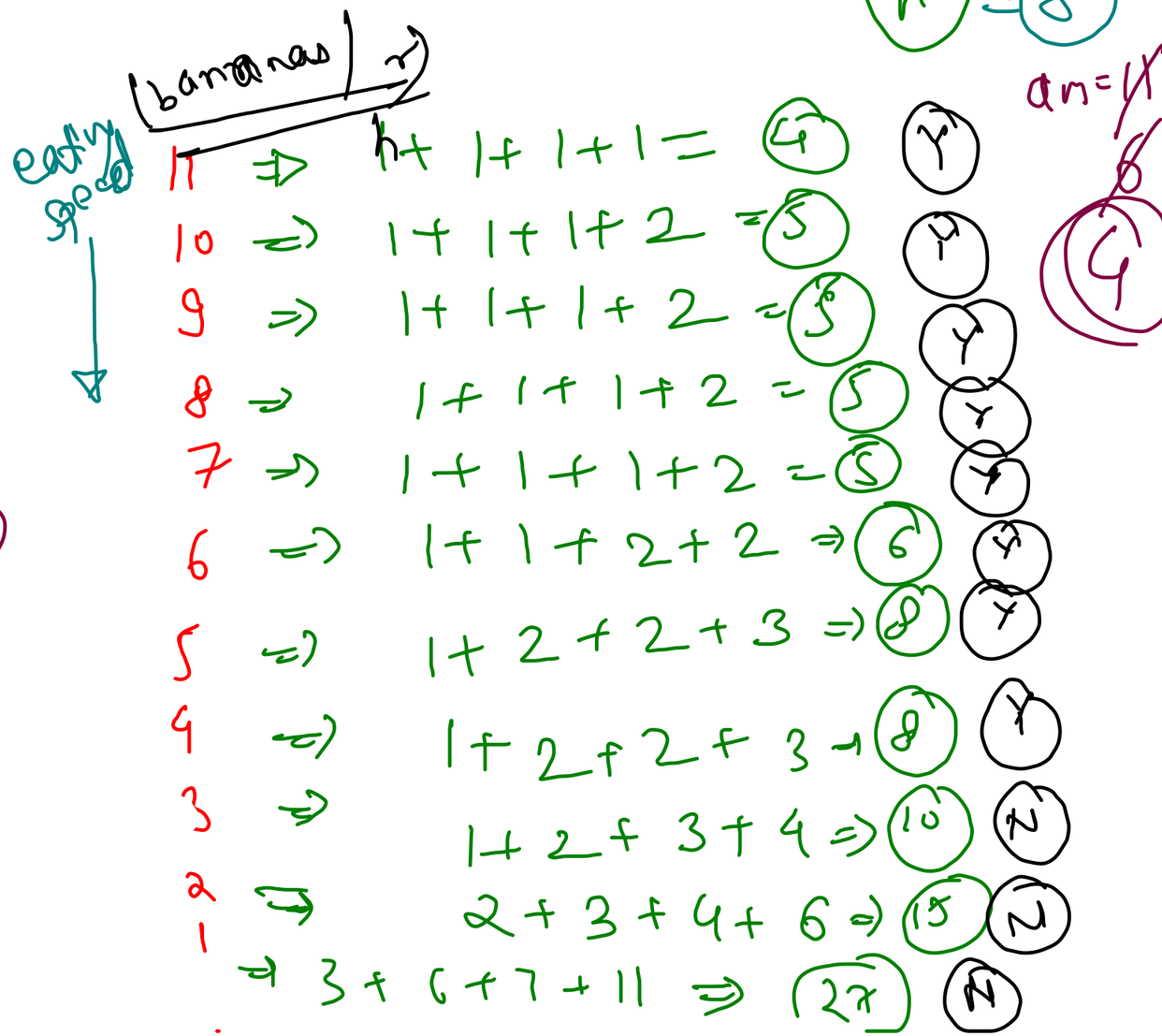
Return *the minimum integer* `k` *such that she can eat all the bananas within* `h` *hours.*

3	6	7	11

k

h = 8

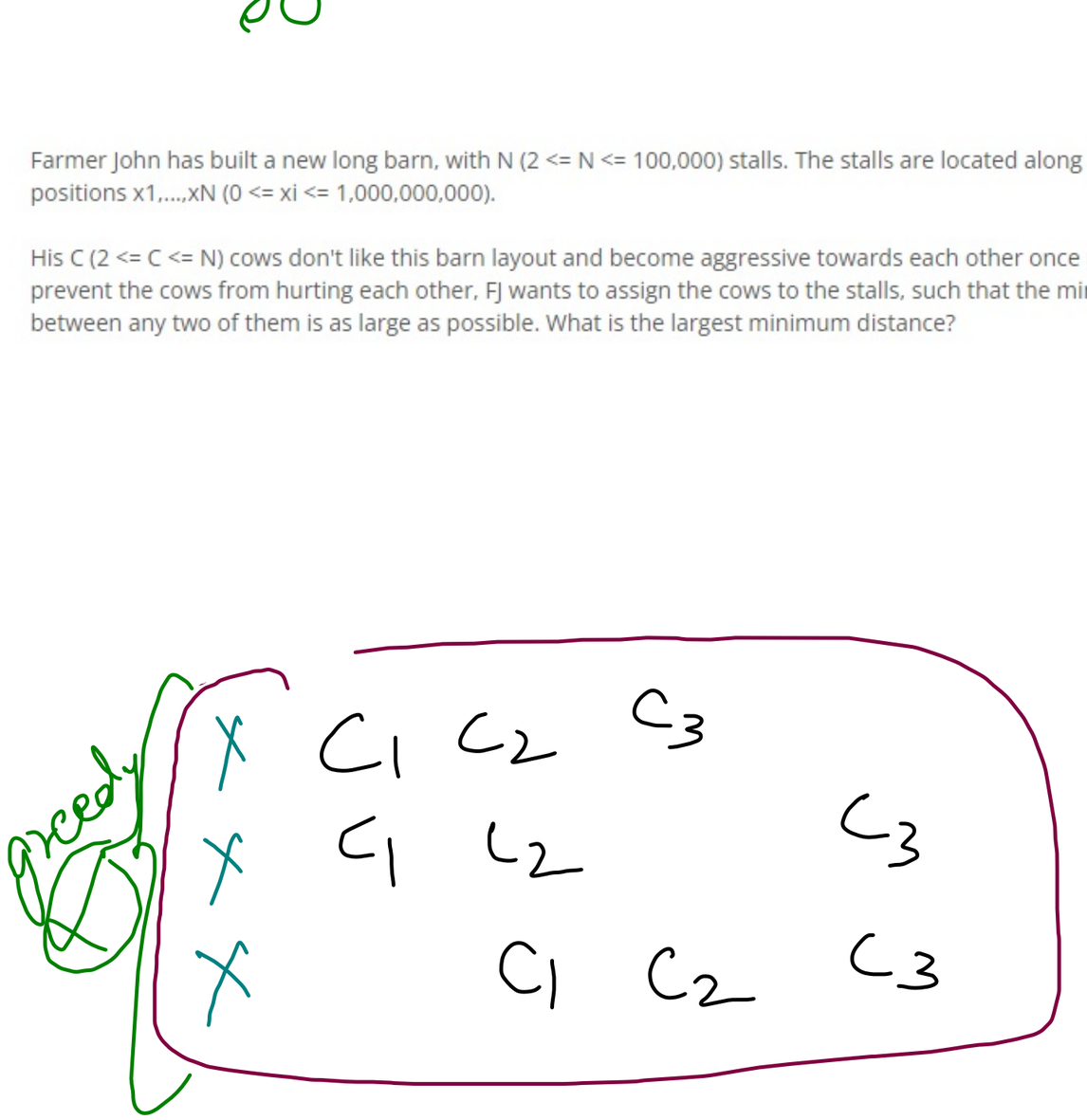
ans = 4

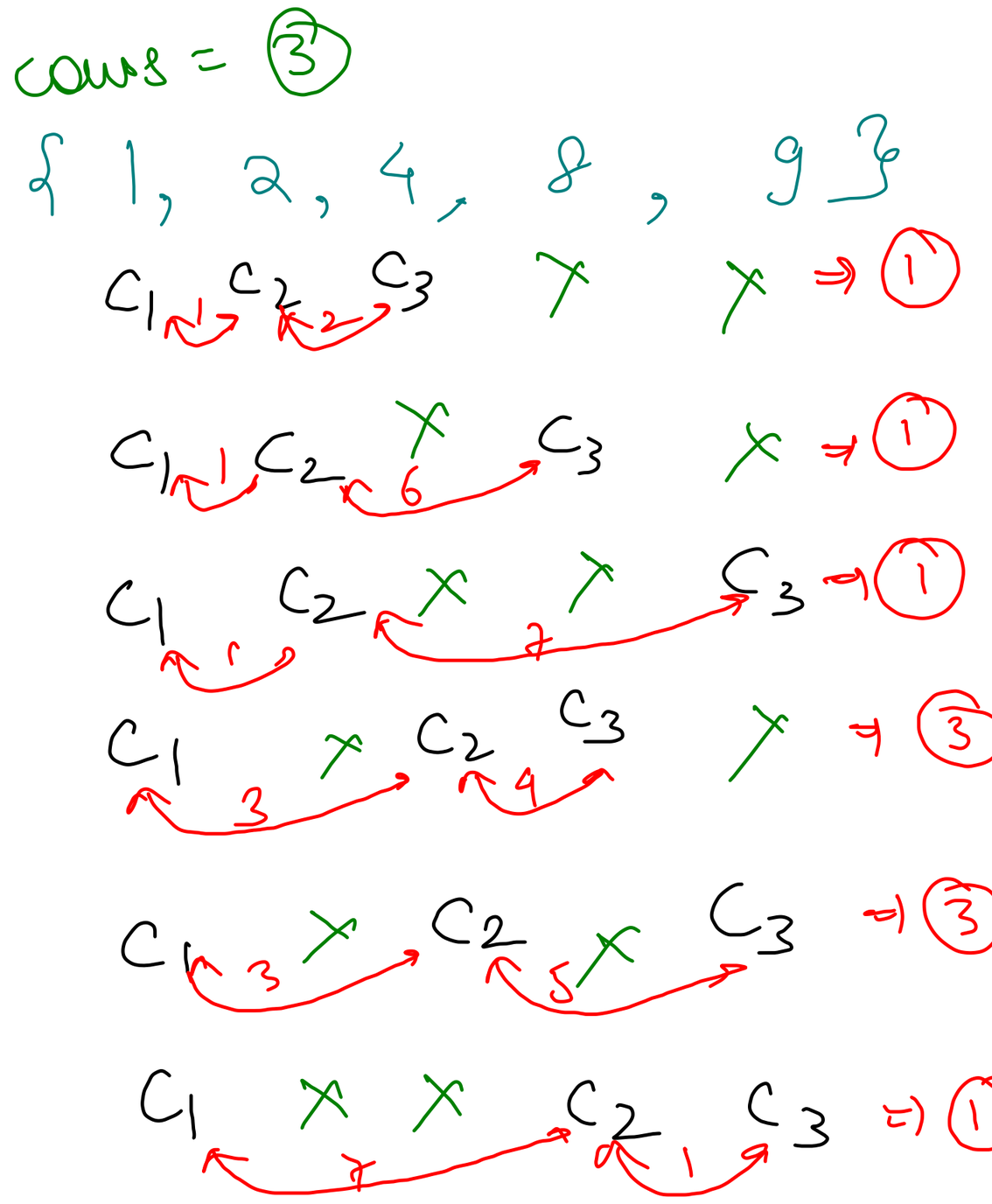


Aggressive Cows

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?

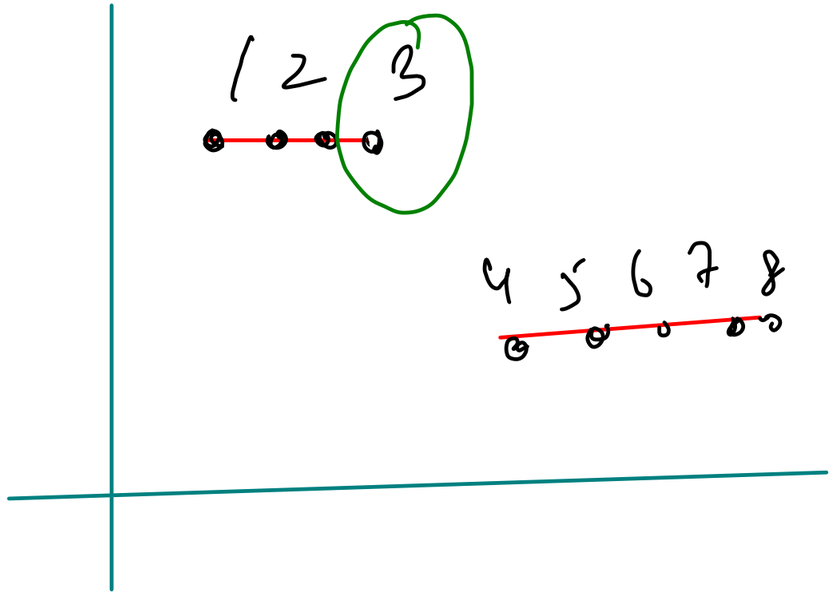




Binary Search on Adjacent distance

k = 3

1, 2, 4, 8, 9



Minimum adj. distance

low = 1

high = 8

1	Yes	$C_1 C_2 C_3 \times \times$
2	Yes	$C_1 \times C_2 C_3 \times$
3	Yes	$C_1 \times C_2 C_3 \times$
4	No	$\times$
5	No	$\times$
6	No	$\times$
7	No	$\times$
8	No	$\times$

if cows = 2

arr[n-1] - arr[0]

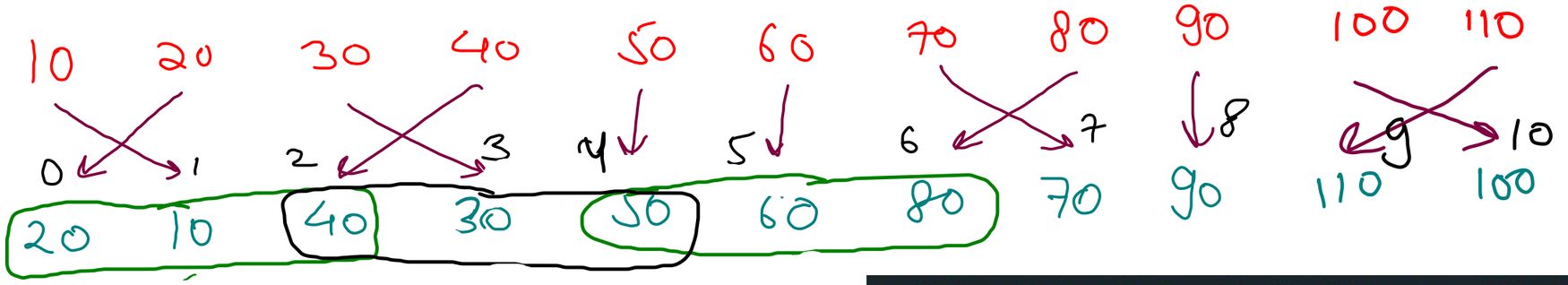
Binary Search - lecture (5)

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



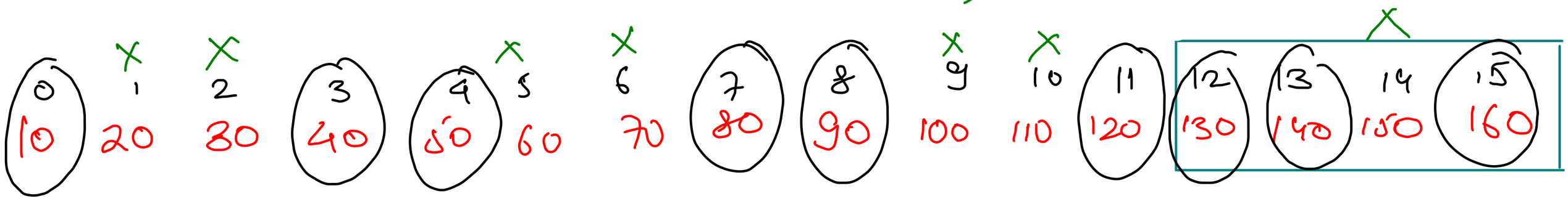
$O(\log_2 N * (2K+1))$

```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

```
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;
}
```

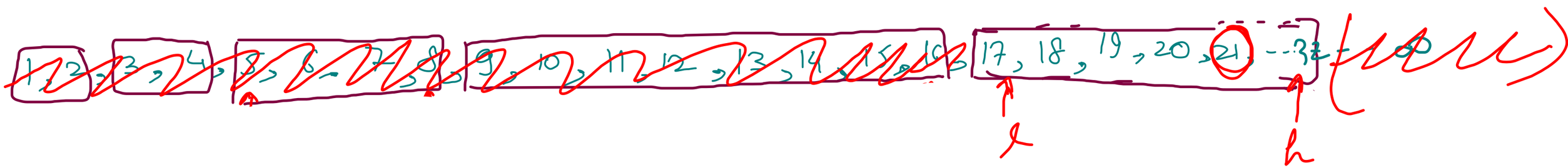
worst case ⇒ O(N)

My 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

O(N + M)
↳ worst case

```java
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 i.e. O(nlogm)

② Staircase search
↳ Divide & Conquer

target = 18

- arr[mid] > target (col--)
  discard column
- arr[mid] < target
  discard row (row++)
- unsuccessful search
  → out of matrix

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]

```
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;
}
```

low = 3 high = 23

mid = 27/2 = 13 N

mid = 18 Y

mid → 14+17/2 (B) (Y)

mid < 18

(Y)