Searching & Sorting

Starting at $\boxed{9:10}$

Lecture 1

Binary Search Basics

1
- Transition Point
- First Bad Version
- Guess Higher or Lower

2
- Floor & Ceil
- First & Last Occurence
- Count Occurences (HW)

3
- Upper & Lower Bound
- Search Insert Position

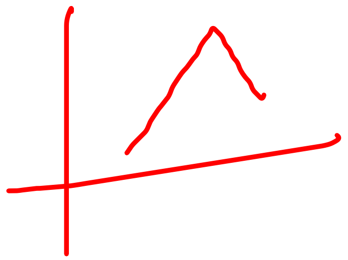
Binary Search

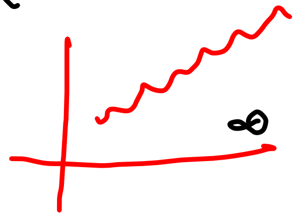
$T(n) = T(n/2) + k$

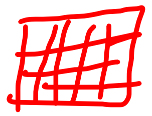
O(N) comparisons
linear search

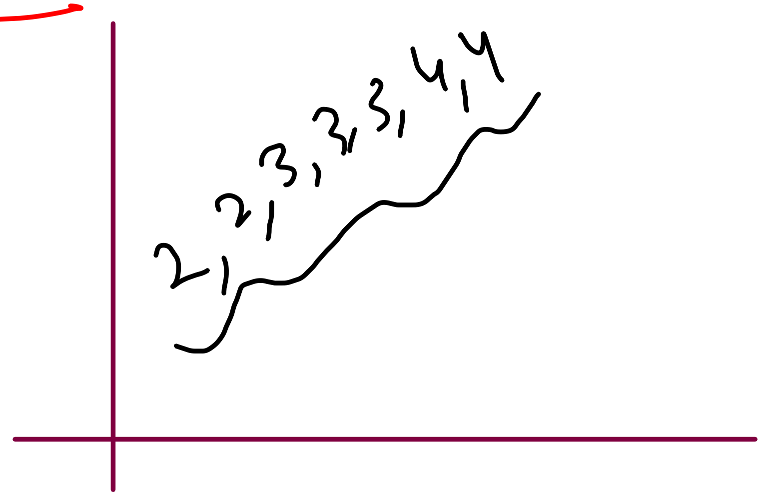
- Time Complexity :→ $O(\log_2 n)$
- Constraint :→ Monotonic → Binary
  - Monotonic
    - Increasing
      - Strictly
    - Decreasing
      - Strictly
  - Binary
    - 0s 1s
      - 00001111
    - 1s 0s → 1111110000
- Modified BS
  - Rotated Sorted Array
  - Bitonic Array
  - Infinite Array (unbounded)
  - Matrix

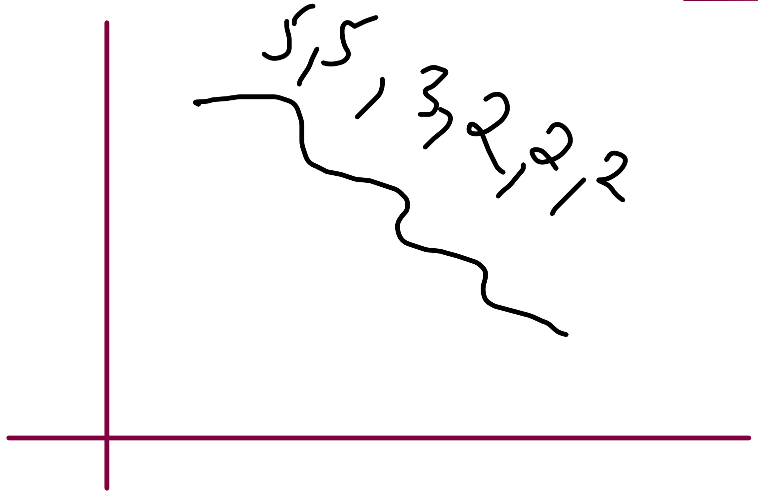
21354
Nearly sorted array



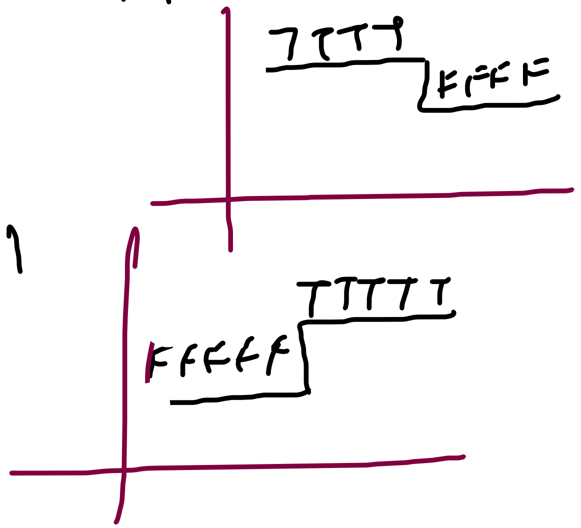








BS on Answer

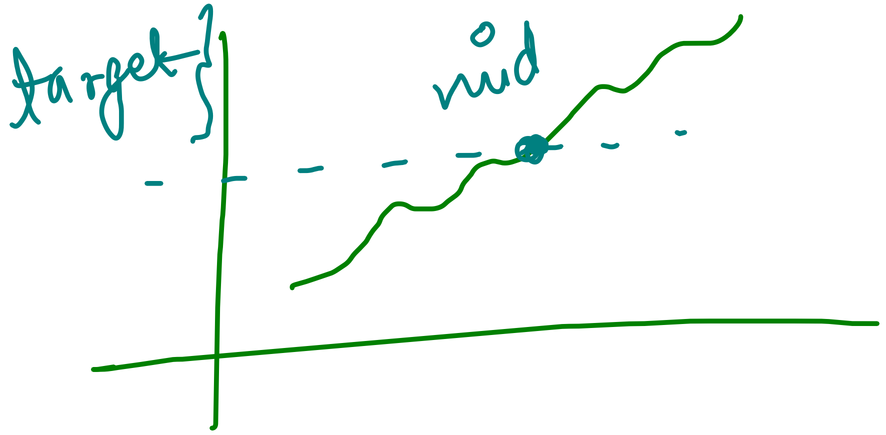


binary Search

high low

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

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




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

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

target => -30

$5n \quad 10n \quad 100n$

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

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

$$T(n/4) = T(n/8) + k$$

$\vdots$

$$T(1) = T(0) + k$$

$$T(n) = k + k + \dots x \text{ times}$$

$$= k * x$$

$$= k * \log_2 n$$

$$= \underline{\underline{O(\log_2 n)}}$$

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

$a = n$

$r = \frac{n/2}{n} = 1/2$

$ar^{x} = 1$

$n\left(\frac{1}{2}\right)^{x} = 1$

$2^x = n$

$x = \log_2 n$

Transition Point

First-Occurrence of (1)

$\{0, 0, 1, 1, 1, 1, 1, 1\}$

0 1 2 3 4 5 6 7

l r m

$ans = \cancel{7}\ \cancel{3}\ 2$

```
int transitionPoint(int arr[], int n) {
    int left = 0, right = n - 1;
    int ans = -1;

    while(left <= right){
        int mid = left + (right - left) / 2;
        if(arr[mid] == 0){
            left = mid + 1;
        } else {
            ans = mid;
            right = mid - 1;
        }
    }

    return ans;
}
```

0 1 2 3

$\{1, 1, 1, 1\}$

0 1 2 3 4

$\{0, 0, 0, 0, 0\}$

First Bad Version

bool isBadVersion(v)

API

V_1	V_2	V_3	V_4	V_5	V_6	V_7
F	F	F	F	T	T	T

high

low

ans = ~~6~~ 5

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.

Q374) Guess Number Or Lower

pick

1 2 3 4 5 6 7 8 9 10 (n)

low right

```
public int guessNumber(int n) {
    int left = 1, right = n;
    while(left <= right){
        int mid = left + (right - left) / 2;

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

guess(5) :- -1

guess(2) :- +1

guess(3) :- 0

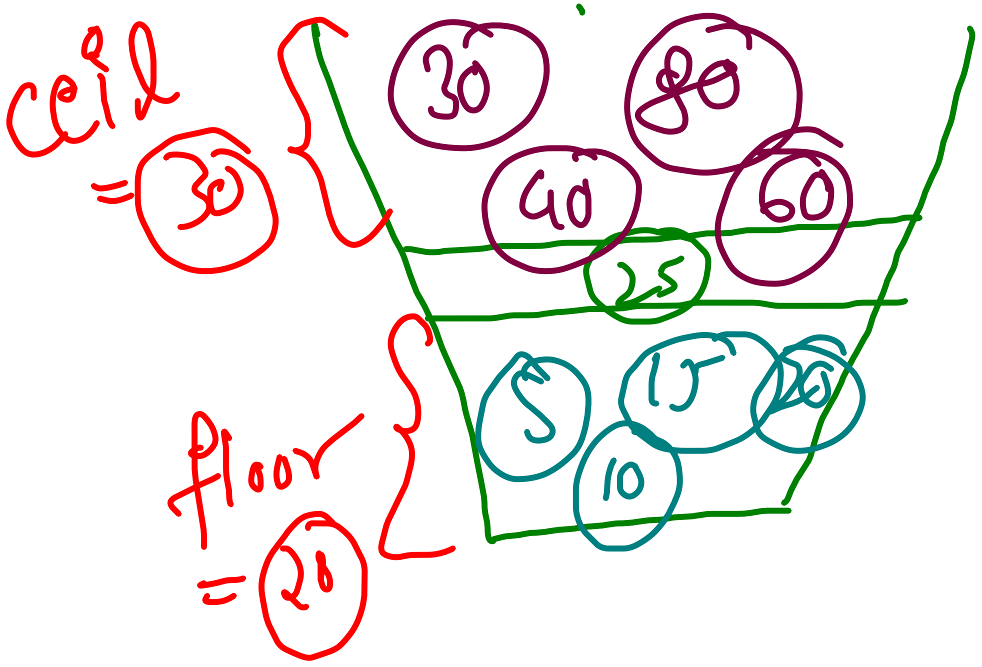
high low

Floor & Ceil {Broken Economy}

0 1 2 3 4 5 6 7

[5, 10, 15, 22, 33, 40, 42, 55]

target = 60

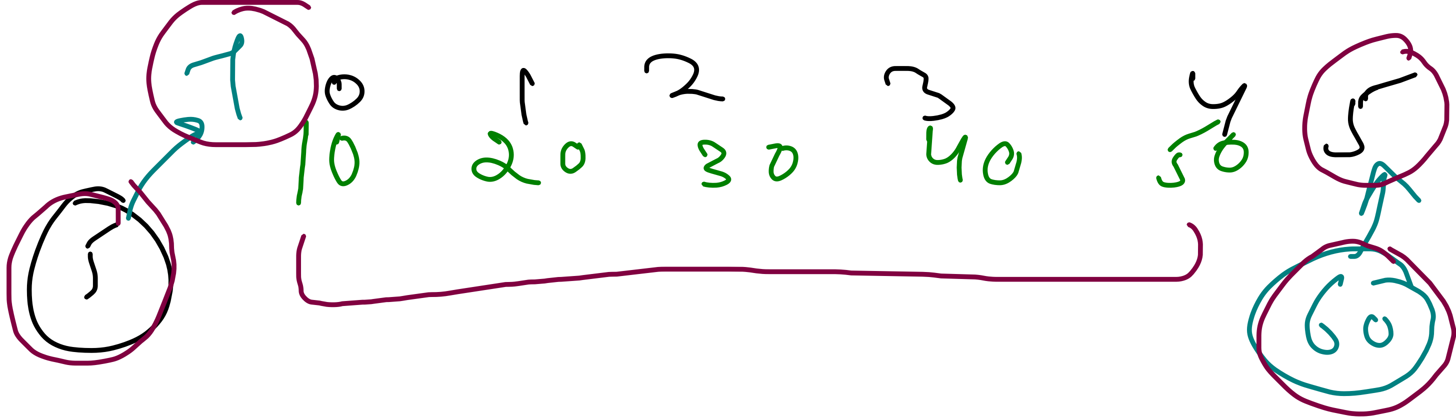


target = 41

40, 42

floor, ceil

lb(r, l)




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

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

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

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

if right => -1
floor does not exist

if left => n , ceil does not exist.

Q(34) First And Last Occurence

target = 40

0	1	2	3	4	5	6	7	8	9	10
~~10~~	~~10~~	~~20~~	~~20~~	~~20~~	~~30~~	~~30~~	~~40~~	~~40~~	40	~~50~~

m

r l

first = ~~8~~ 7

last = ~~8~~ 9

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

        if(nums[mid] == target){
            ans = mid;
            right = mid - 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            // nums[mid] > target
            right = mid - 1;
        }
    }
    return ans;
}
```

logn

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

        if(nums[mid] == target){
            ans = mid;
            left = mid + 1;
        } else if(nums[mid] < target){
            left = mid + 1;
        } else {
            // nums[mid] > target
            right = mid - 1;
        }
    }
    return ans;
}
```

logn

```java
public int[] searchRange(int[] nums, int target) {
    int[] ans = {-1, -1};
    if(nums.length == 0) return ans;

    ans[0] = firstOcc(nums, target);
    ans[1] = lastOcc(nums, target);
    return ans;
}
```

$1, 2, 3, 4, 5$

l $\quad$ r

(1) $[l, r] \Rightarrow 5 - 1 + 1 \Rightarrow r - l + 1$

(2) $(l, r]$, $[l, r) \Rightarrow 5 - 1 \Rightarrow r - l$

(3) $(l, r) \Rightarrow 5 - 1 - 1 \Rightarrow r - l - 1$

Search Insert Position (Q35)

target = ~~24~~, 15

lower bound

on unique elements

mid

0	1	2	3	4	5	6	7	8	9
~~1~~	~~5~~	~~8~~	~~10~~	~~13~~	~~17~~	~~20~~	~~25~~	~~28~~	~~30~~

r m l

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

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

lower & upper bound

C++

lower_bound (arr, arr+n)

upper_bound (arr, arr+n)

if element found

return first occurrence

arr[mid] == target

if element not found, then return just greater value (ceil)

arr[mid] > target

floor = lb - 1

just greater value

{ceil}

first occurence of ceil

{ 10, 10, 10, 20, 20, 40, 40, 40, 50, 50 }

0 1 2 3 4 5 6 7 8 9

```java
public static int lowerBound(int[] arr, int target){
    int left = 0, right = arr.length - 1;
    int ans = arr.length;

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

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

{0, 0}

target = 5

{ceil not found}

10 10

target = 100

{3, 5}

target = 20

40 40

{5, 5}

target = 30

Lecture-2

Starting at $\boxed{4:10}$.

1. Closest Element
   K-closest Elements

2. Heaters

3. Square Root
   Valid Perfect Square
   Nth Root (HW)

4. Pivot in Rotated Sorted Array
   → Duplicates not allowed
   → Duplicates allowed
   Count Rotations (HW)

5. Search in Rotated Sorted Array
   → Duplicates not allowed
   → Duplicates allowed (HW)

Closest Element

floor ← | → ceil

{ 1, 1, 10, 20, 25, 48, 48, 80, 60, 60 }

0 1 2 3 4 5 6 7 8 9

25, 20, 48, 48, 10

target = 30

25 − 30 = 5

48 − 30 = 18

target = 10

10

Binary Search on Element

→ if search successful

⇒ return ele (diff = 0)

→ if search unsuccessful

↳ return floor or ceil whichever is closer

$O(2\log_2 n)$

+

$O(k)$

5

∞

−∞ 10

60

10 80 ∞ ∞

using binary search

```java
int second = lowerBound(arr, x);
int first = second - 1;
```

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

while(first >= 0 && second < arr.length && k-- > 0){
    if(Math.abs(arr[first] - x) <= Math.abs(x - arr[second])){
        res.add(arr[first]);
        first--;
    } else {
        res.add(arr[second]);
        second++;
    }
}

while(first >= 0 && k-- > 0){
    res.add(arr[first]);
    first--;
}

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

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

lowerBound

two pointer

sorting

$O(\log_2 n) + O(k) + O(k \log k)$

$= O(\log_2 n + k + k \log k)$

Heaters (Q475)

answer will be maximum of closest heater from every house

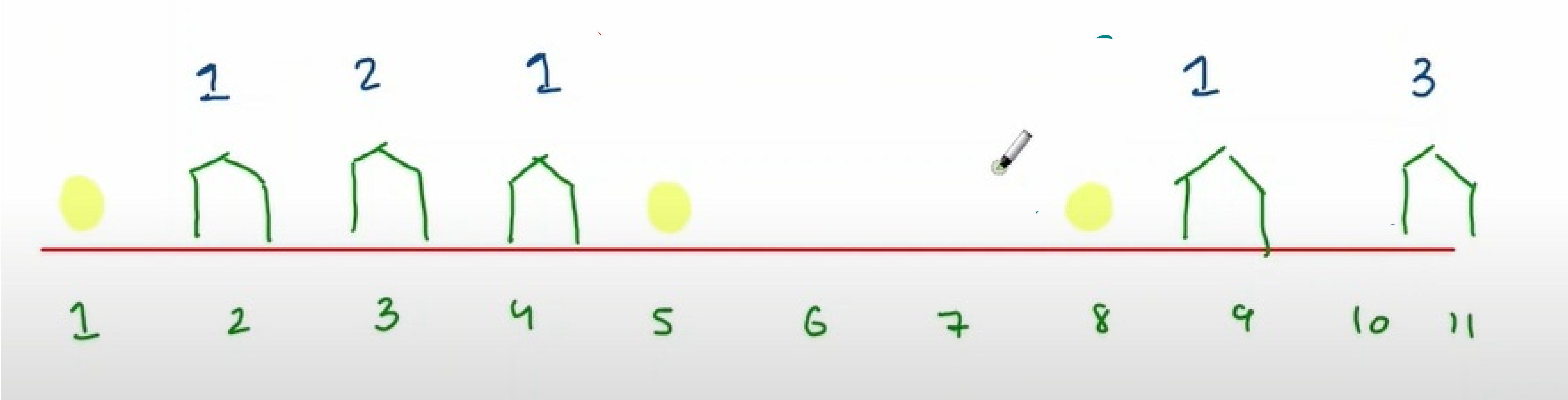
houses = h_1 [3, 9, 2, 4, 11]

heaters = h_2 [1, ~~8~~ 5, ~~5~~ 8]

heater ⇒ sorted

house ⇒ queries (unsorted)

an = 3



bs(heaters, 3) (1, 5, 8) 2 = 1

bs(heaters, 9) 1 = 8

bs(heaters, 2) (1, 5, 8) 1 = 1

bs(heaters, 4) 1 = 5

bs(heaters, 11) 3 = 8

$h_2 \log h_2 + h_1 * \log h_2$

$= (h_1 + h_2) \log h_2$

```java
public static int lowerBound(int[] arr, int target){
    int left = 0, right = arr.length - 1;
    int ans = arr.length;

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

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

```java
public static int closest(int[] arr, int target){
    int lb = lowerBound(arr, target);

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

    else if(Math.abs(target - arr[lb]) < Math.abs(target - arr[lb - 1]))
        return arr[lb];

    else return arr[lb - 1];
}

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

    int max = Integer.MIN_VALUE;
    for(int i=0; i<houses.length; i++){

        int closestHeater = closest(heaters, houses[i]);
        max = Math.max(max, Math.abs(closestHeater - houses[i]));

    }
    return max;
}
```

Square Root → Integral (Q 69)

floor-sqrt = ~~5~~ ~~6~~ 7

5*5 = 25   6*6 = 36   7*7 = 49   8*8 = 64

1 2 3 4 5 6 7 8 9 10

l   r

$O(\log_2 n) \approx O(1)$

floor sqrt = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ 7

$O(\sqrt{N})$

perfect square → 49 ⇒ 7

not a perfect square → 60 ⇒ 7

```java
public int mySqrt(int x) {
    if(x == 0) return 0;

    long left = 1L, right = x;
    long floorSqrt = 1L;
    while(left <= right){
        long mid = left + (right - left) / 2L;
        long square = mid * mid;

        if(square == x){
            return (int)mid;
        } else if(square < x){
            floorSqrt = mid;
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
    return (int)floorSqrt;
}
```

Square root → Fractional No

precision = 3 p

target = 60

1 floor square root → 7

accuracy 0.1

2.1 7.0 / 7.1 / 7.2 / 7.3 / 7.4 - - - - 7.9

7+0.1*1 7+0.2 = 7+2*0.1 → 7+0.1*3 7+0.1*4

floor + accuracy ↑ (0→9)

2 0.01

2.2 7.x0 / 7.x1 / 7.x2 - - - / 7.x9

7.x + 0.01*1 7.x + 0.01*2 7.x + 0.01*9

0.001

2.3 7.xy0 / 7.xy1 / 7.xy2 / - - -

floor precision

$O(\log_2 N + p * 10)$

```
double sqroot(int n, int precision)
{
        double ans = floor_sqroot(n);
        double j = 0.1;
        while (precision--)
        {
                while (ans * ans <= n)  ans += j;
                ans = ans - j;
                j = j / 10;
        }
        return ans;
}
```

Starting at 9:10

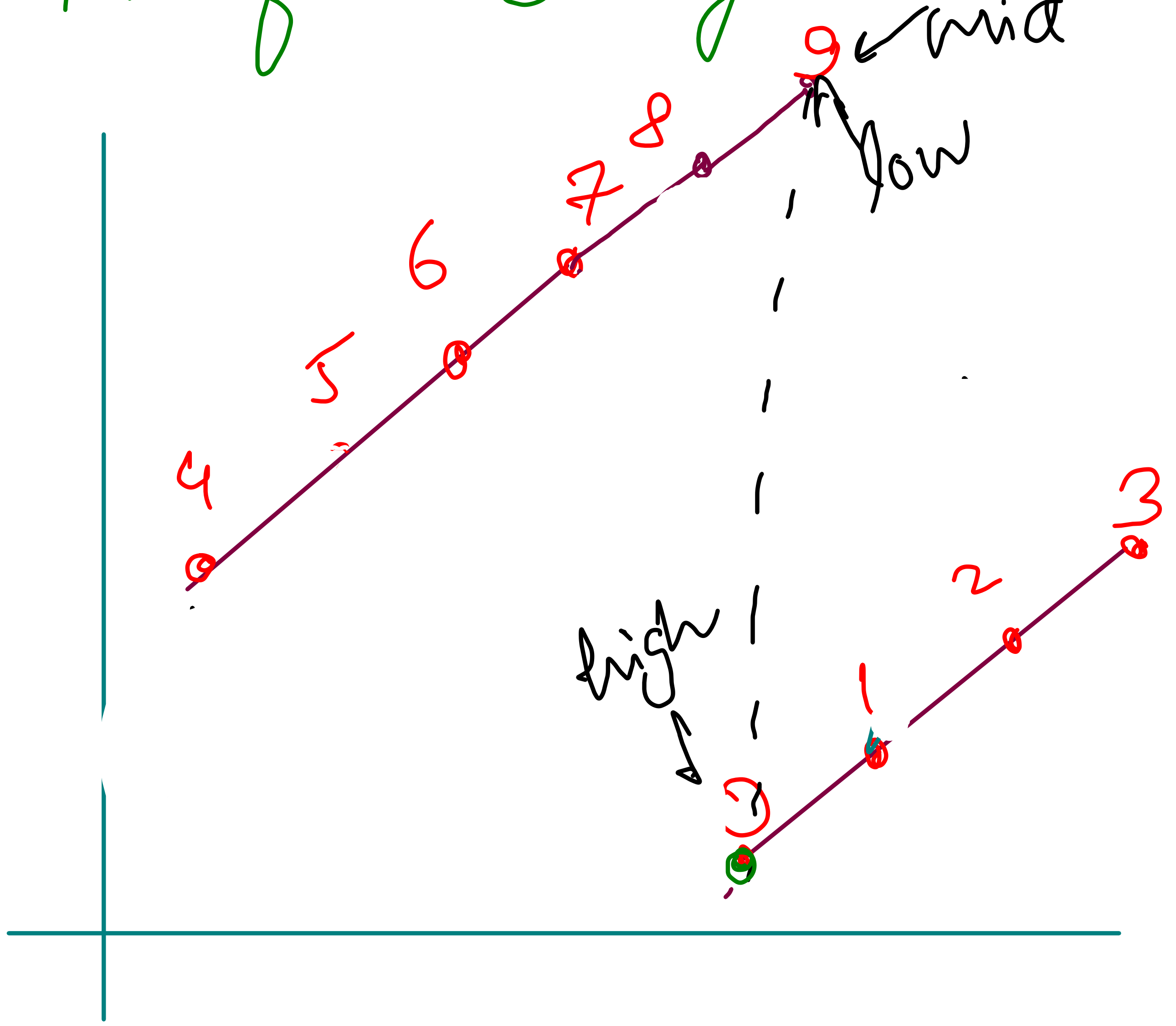
Binary Search Lecture - 3

Rotated Sorted Array, Bitonic Array

Rotated Sorted Array
1 Pivot or Minimum - I (IB)
Pivot or Minimum - I
Pivot or Minimum - II
2 Find Rotation Count
Find Rotation Count
3 Searching - I (IB)
Searching - I
Searching - II

Bitonic Array
4 Maximum Element (IB)
Maximum Element
5 Search Element
Search Element
6 Peak Element
Peak Element

Modified Binary Search



mid

0	1	2	3	4	5	6	7	8	9
~~4~~	~~5~~	~~6~~	~~7~~	~~8~~	9	0	~~1~~	~~2~~	~~3~~

l h

mid > left
mid > right

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

if (mid == min) return mid;

mid < left, < right

$O(\log_2 n)$

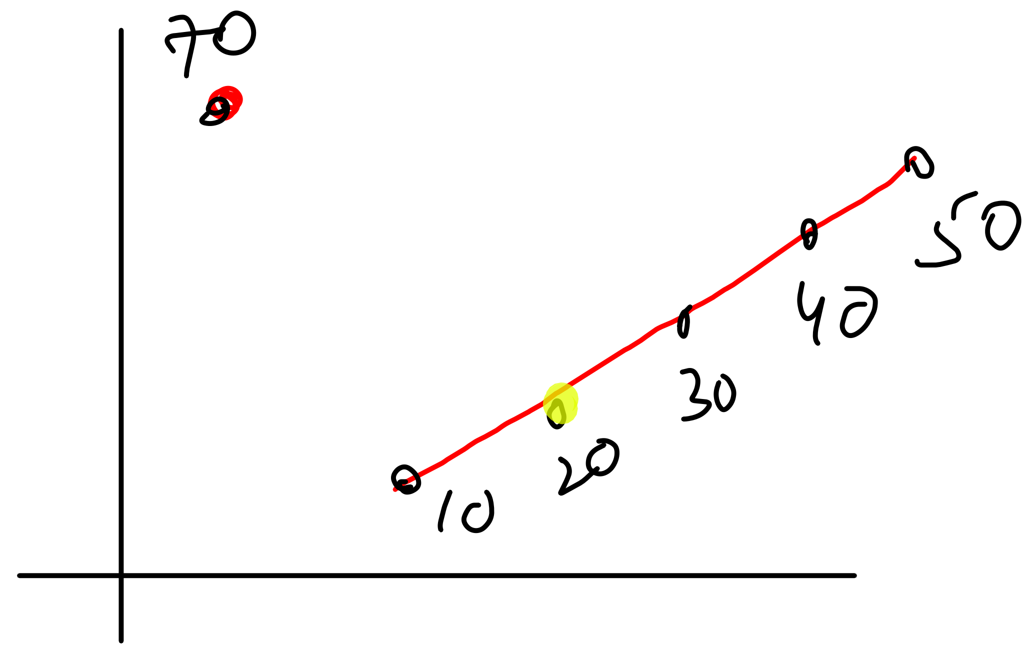
nums[low] < nums[mid]

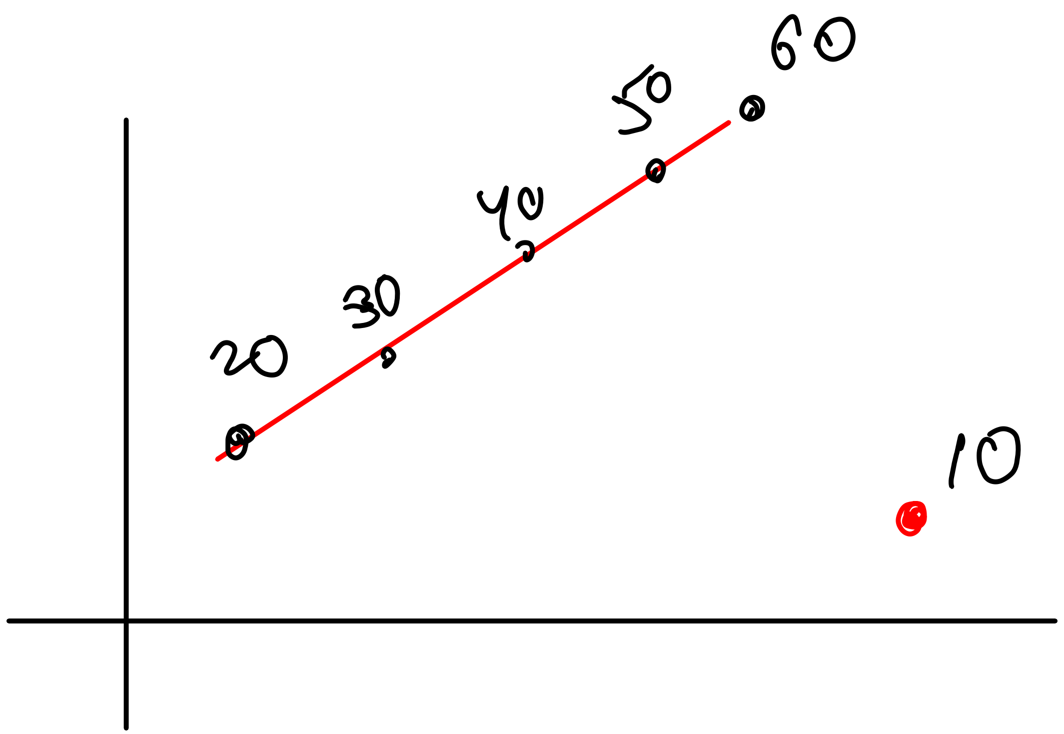
⇒ left range is sorted

⇒ l = m+1

else ⇒ left range is unsorted

⇒ r = m-1





0	1	2	3	4	5
70	10	20	30	40	50

m l h

Corner case → entire sorted array

nums[0] < nums[n-1] : return 0;

l m h

20	30	40	50	60	10
0	1	2	3	4	5

Searching in Rotated Sorted Array

0	1	2	3	4	5	6
4	5	6	7	0	1	2

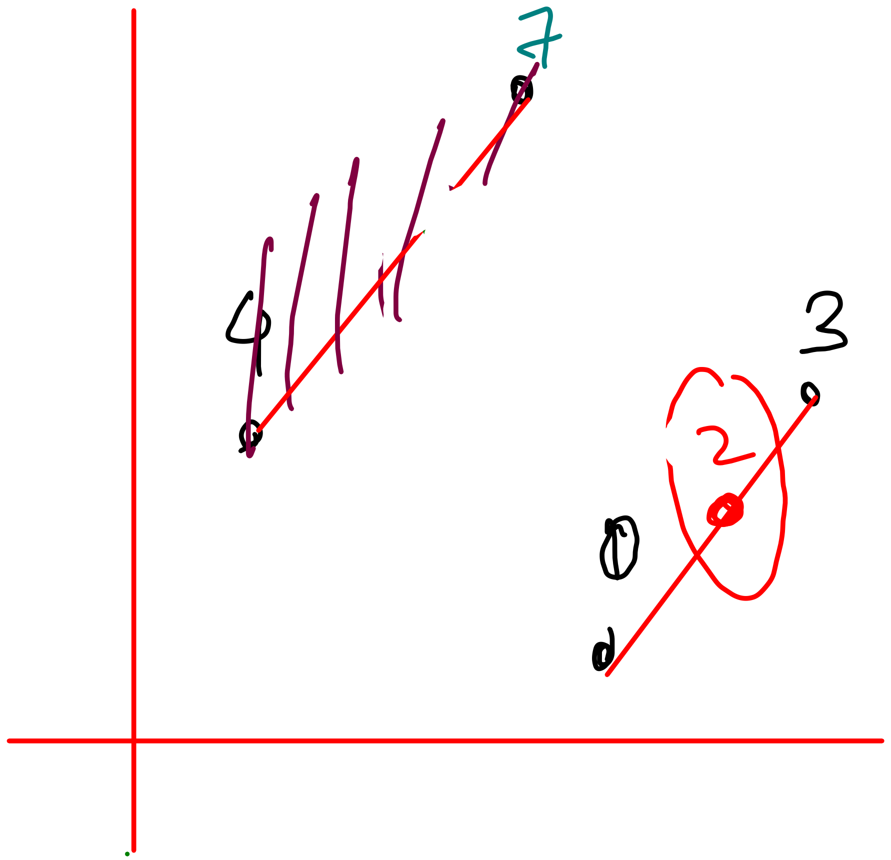
target = 3

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

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

    if(arr[left] <= arr[mid]){
        // left range is sorted
        if(arr[left] <= target && target < arr[mid]){
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    else if(arr[mid] < arr[right]){
        // right range is sorted
        if(arr[mid] < target && target <= arr[right]){
            left = mid + 1;
        } else {
            right = mid - 1;
        }
    }
}
```



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

l h

fp = 6

fp = 4

fp = 3

fp = 1

$n = 6$

$m = 7$

no of ones $= m - fp$

ans = $\cancel{0}\ \cancel{1}\ \cancel{3}\ \cancel{4}\ 6$

best case ⇒ $\log m$

worst case ⇒ $n \log m$

```
int transitionPoint(int arr[], int left, int right) {
    int ans = -1;

    while(left <= right){
        int mid = left + (right - left) / 2;
        if(arr[mid] == 0){
            left = mid + 1;
        } else {
            ans = mid;
            right = mid - 1;
        }
    }

    return ans;
}
```

```
int rowWithMax1s(int arr[][], int n, int m) {
    int right = m - 1;
    int ans = -1, noOfOnes = 0;
    for(int i=0; i<n; i++){
        int transitionPt = transitionPoint(arr[i], 0, right);
        if(transitionPt != -1 && m - transitionPt > noOfOnes){
            ans = i;
            noOfOnes = m - transitionPt;
            right = transitionPt - 1;
        }
    }
    return ans;
}
```

target = 10

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

(0,2) (0,3) (1,0) (1,1)

n*m-1

$$\begin{array}{cccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ \{1, & 3, & 5, & 7, & 10, & 11, & 16, & 20, & 23, & 30, & 34, & 60\} \end{array}$$

l r

$O(\log_2(n*m))$

(c) cell → row (c/m), col (c%m) → arr[r][c]

```
public boolean searchMatrix1(int[][] matrix, int target){
    int n = matrix.length, m = matrix[0].length;
    int left = 0, right = n * m - 1;

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

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

        if(matrix[rowIdx][colIdx] == target){
            return true;
        }

        if(matrix[rowIdx][colIdx] < target){
            left = midCell + 1;
        }

        else right = midCell - 1;
    }
    return false;
}
```