Searching & Sorting

Starting at 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

O(N) comparisons linear search

Time Complexity :→ $O(\log_2 n)$

Constraint :→

21354

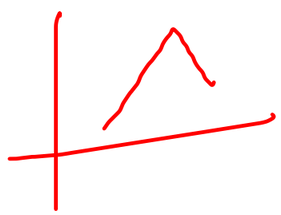
Nearly sorted array

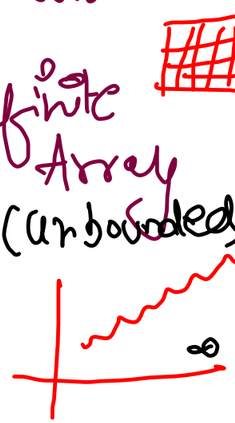
Modified BS → Matrix

Infinite Array (unbounded)

Rotated Sorted Array

Bitonic Array



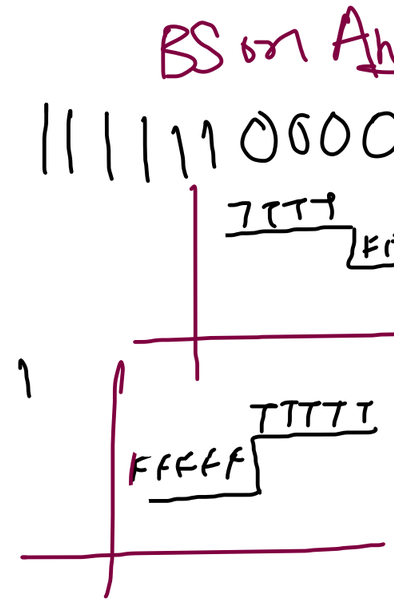


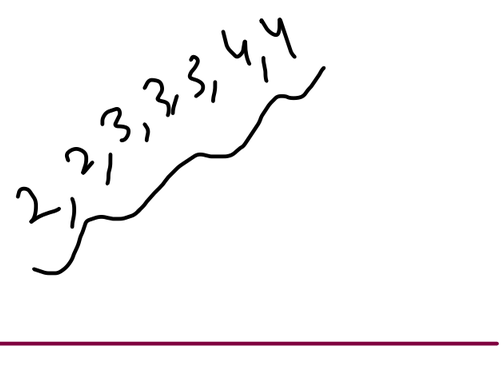
Monotonic

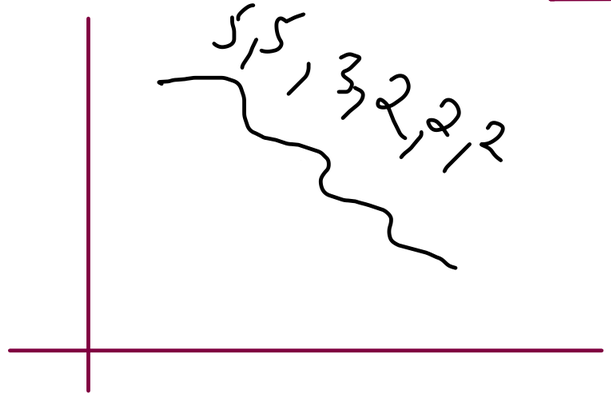
- Increasing
  - Strictly
- Decreasing
  - Strictly
- Binary
  - 0s 1s
  - 1s 0s

0000111 11

BS on Answer



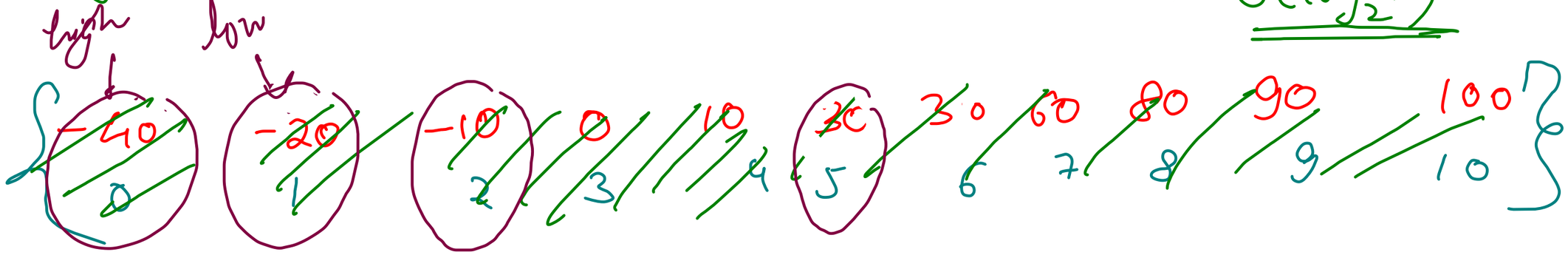


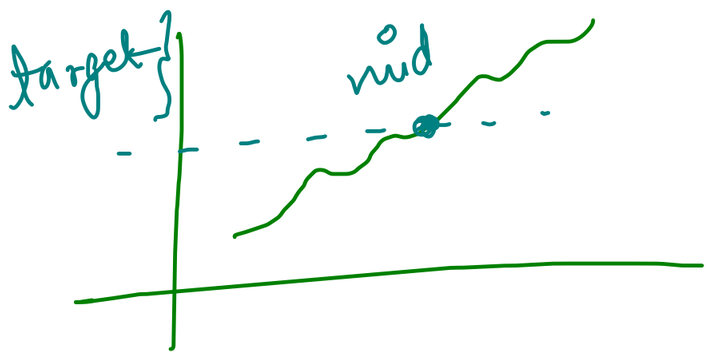


binary Search

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

$\Rightarrow O(\log_2 n)$






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

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

target => -30

Transition Point

First-Occurence of (1)

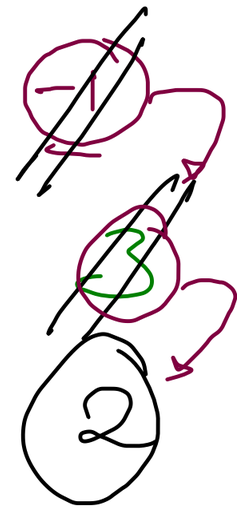
{ 0, 0, 1, 1, 1, 1, 1, 1 }

0 1 2 3 4 5 6 7

high

low

ans =




```
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;
}
```

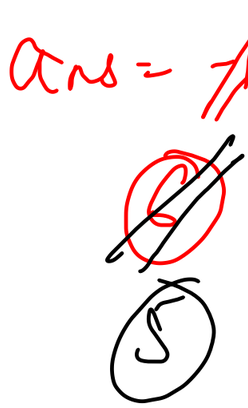
First Bad Version

bool isBadVersion(v)

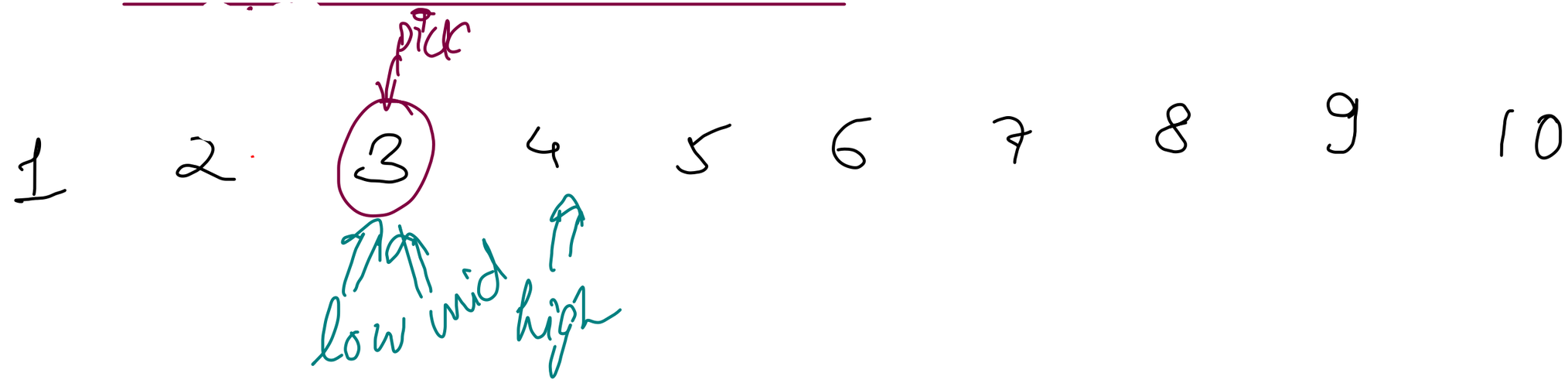
V_1 V_2 V_3 V_4 V_5 V_6 V_7

F F F F T T T

```java
public int firstBadVersion(int n) {
    int left = 1, right = n, ans = -1;

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

        if(isBadVersion(mid) == false){
            left = mid + 1;
        } else {
            ans = mid;
            right = mid - 1;
        }
    }
    return ans;
}
```



(0374) Guess Number Or lower




```java
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

Floor & Ceil & Broken Economy 3

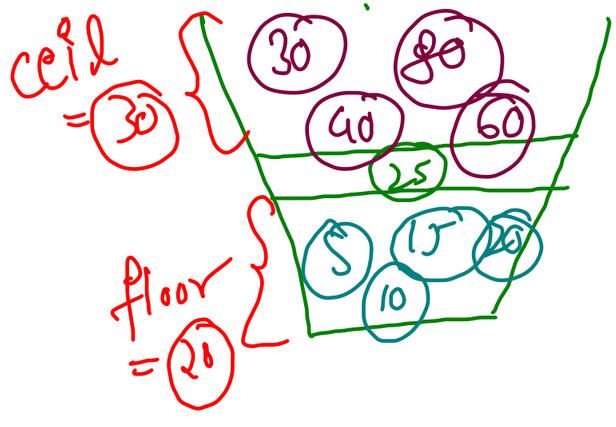
target = 60

0 1 2 3 4 5 6 7

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

target = 41

40, 42



floor, ceil
Ub (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];
}
```

if right => -1
floor does not exist

```java
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 left => n, ceil does not exist.

(134) First And Last Occurence

target = 40

Count occurences
if not found return 0
else return l-f+1.

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

f

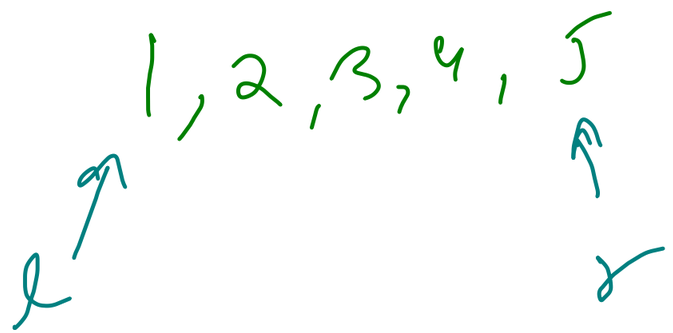
```java
public int firstOcc(int[] nums, int target){
    int left = 0, right = nums.length - 1;
    int ans = -1;
    while(left <= right){
        int mid = left + (right - left) / 2;

        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;
}
```

l

```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;
}
```

first = -1 ⇒ 7
last = -1 ⇒ 9

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

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

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

Search Insert Position (35)

target = 19

lower bound

on unique elements

{ 1, 5, 8, 10, 13, 17, 20, 21, 28, 30 }

0 1 2 3 4 5 6 7 8 9

mid

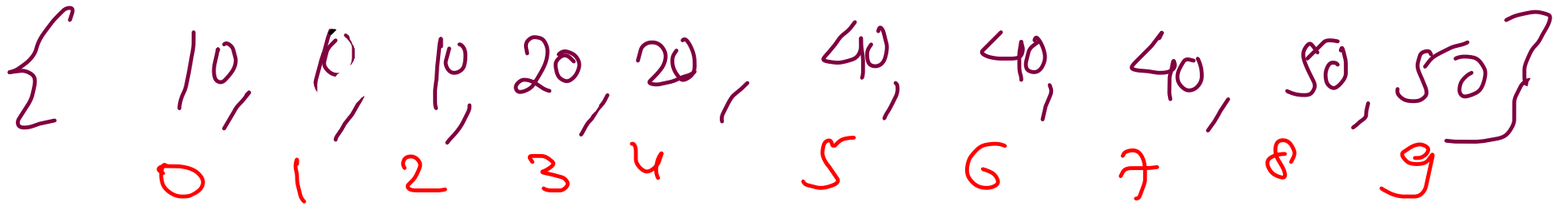
```java
public int searchInsert(int[] nums, int target) {
    int left = 0, right = nums.length - 1;
    while(left <= right){
        int mid = left + (right - left) / 2;

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

lower & upper bound

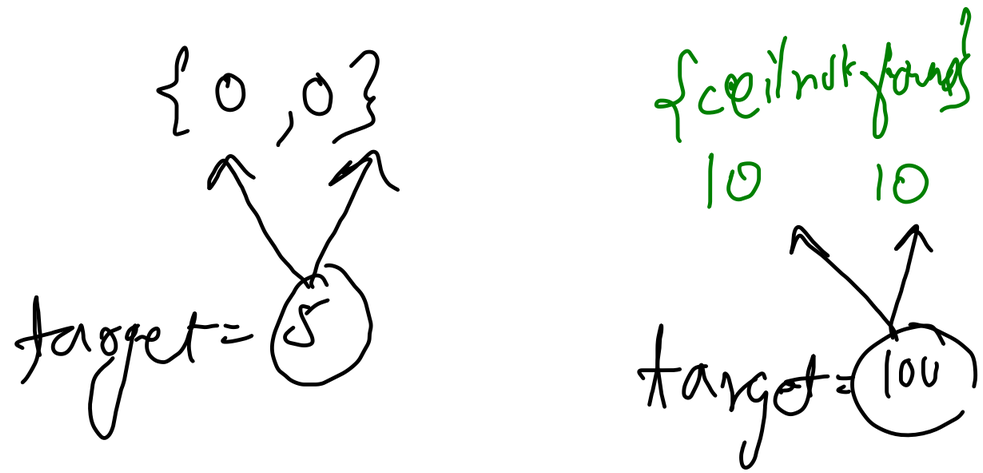
if element found
↓
return first occurrence
arr[mid] == target

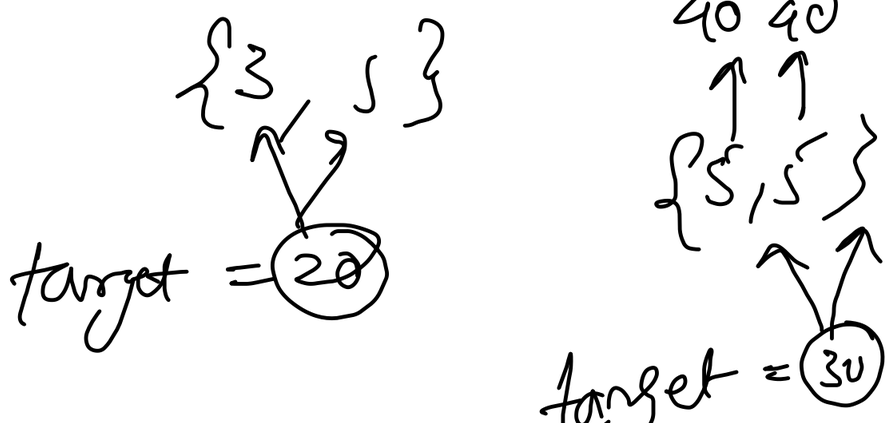
if element not found, then return just greater value (ceil)
arr[mid] > target
↓
low = mid + 1

just greater value
{ceil}
↓
first occurence of ceil




```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;
}
```





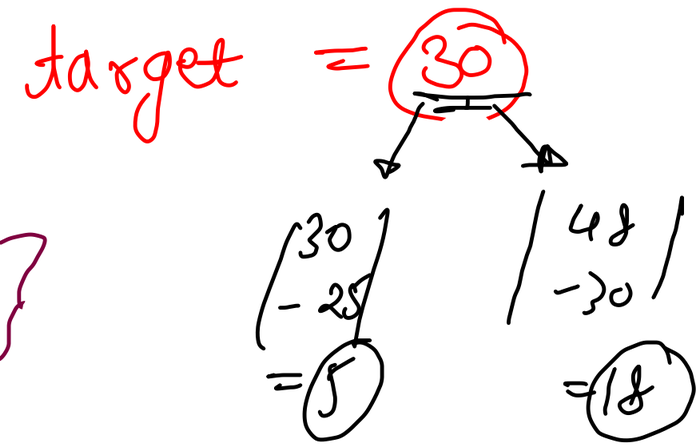
Lecture-(2)

Starting at 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

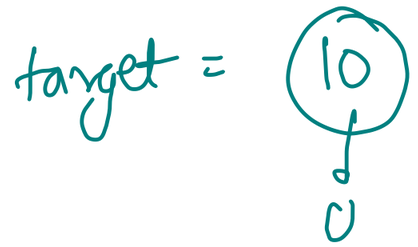
{ 10, 10, 10, 20, 25, 48, 48, 48, 50, 50 }
0 1 2 3 4 5 6 7 8 9

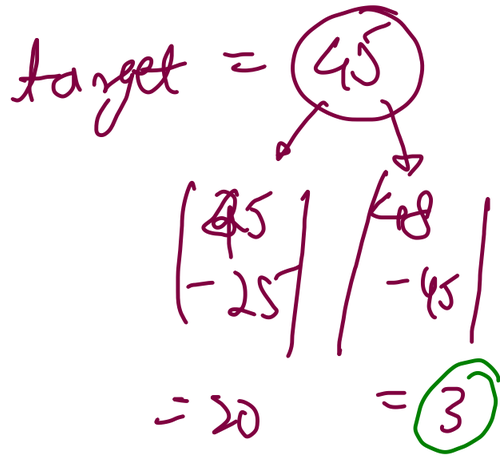


Binary Search on Element

→ if search successful
⇒ return ele (diff = 0)

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





using binary search

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

```
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

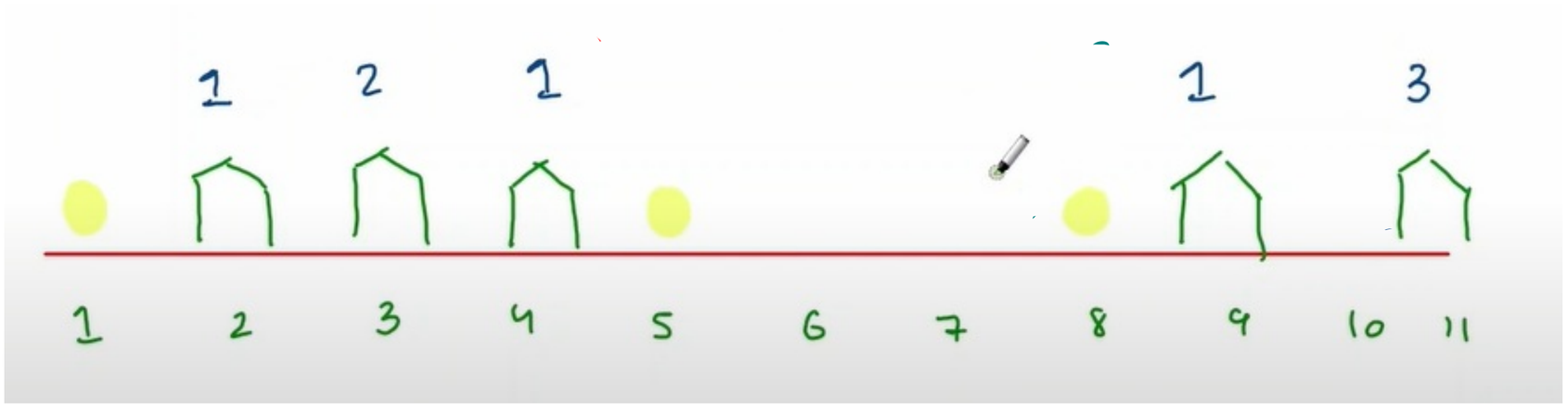
$O(\log_2 n) + O(k) + O(k \log k)$

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

Heaters (0475)

houses = h1 [3, 9, 2, 4, 11]

heaters = h2 [1, 8, 5]



answer will be maximum of closest heater from every house

heater ⇒ sorted

house ⇒ queries (unsorted)

ans = 3

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

bs(heaters, 9) = 8

bs(heaters, 2) = 1

bs(heaters, 4) = 5

bs(heaters, 11) = 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;
}
```

floor-sqrt

=

Square Root → Integral (0 69)

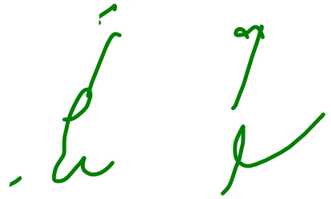
1 2 3 4 5 6 7 8 9 10

5*5 = 25

6*6 = 36

7*7 = 49

8*8 = 64



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

floor sqrt =

$O(\sqrt{N})$

perfect square

49

11

7

not a perfect square

60

11

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

(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 + accuracy ↑ (0→9)

```
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;
}
```

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

floor, precision