1. Closest Elements
   - K- closest elements
2. Heaters
3. Square root
   - Valid perfect square
   - N^{th} root (HW)
4. Pivot in Sorted rotated array
   - Duplicates not allowed
   - Duplicates allowed
   - Count Rotations (HW)
5. Search in rotated sorted array
   - Duplicates not allowed
   - Duplicates allowed (HW)

Previous class work

① Upper bound: Similar to finding ceil value

② Lower bound:

```
int start=0, end = n.length-1, ans = arr.length;
while (start <= end){
    int mid = start + (end - start)/2;
    if(arr[mid] >= target) {
        ans = mid;
        end = mid - 1;
    } else {
        start = mid +1;
    }
}
return ans;
```

① Closest elements

{10, 10, 10, 20, 25, 48, 48, 48, 50, 50}

Case ①, target = 30 → if 30 found → 30

→ abs(30-25) = 5, abs(30-48) = 18

if target does not exist, then floor or ceil value of target

Approach: Binary Search of Element

if search successful, return 0

if Search fails,

look for floor and ceil check for lowest absolute difference b/w them & target and return it

return min(abs(floor - target), abs(ceil - target))

Use Lower Bound

② k- Closest elements. TC: $O(\log_2 n + k + k \log_2 k)$

after finding closest element, then use 2 pointer approach

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

for target = 30

found

ceil →

← floor

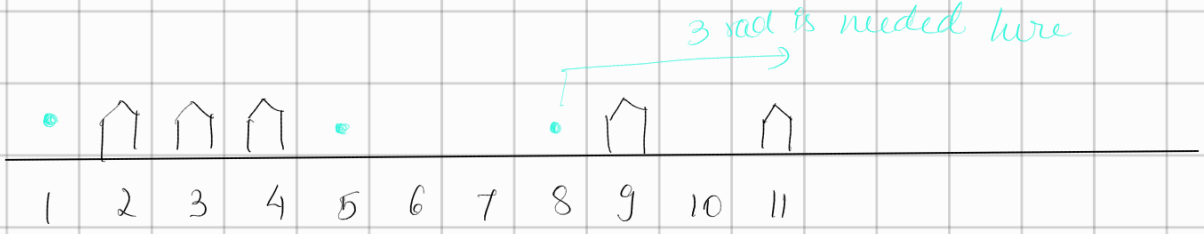
```
while (k--) {
    add min(floor, ceil)
    in answer arr/vector
    & move added correspondingly
    floor ++ / ceil--
```

Corner Case: floor < 0 or ceil > n-1

③ Heaters

houses: [3 9 2 4 11]

heaters: [1 8 5]



Approach 1: make radius = 1 2 - - upto all houses covered

✗ bad TC.

Approach 2: find max. distance of nearest heater for all heaters for all houses

→ Binary Search on heaters.

for house at 2 → 1
3 → 2
4 → 1
9 → 1
11 → 3 ↓ max = 3,

for each house find closest, and return max. of these closest arrays as answer (min required heater radius)

TC: $(h1 + h2) \log h2$

④ SQRT(x)

25 → 5
31 → 5
37 → 6
41 → 6
48 → 6
49 → 7

apply find floor in 1 to x range

check linearly upto $\sqrt{n}$ → $O(\sqrt{n})$

⑤ SQRT: Fractional (60)

① floor → sqrt → ④ ✓

② Increase steps one by one : 7.(0→9) 0.1
: 7.a (0→9) 0.01
: 7.ab (0→9)
: 7.abc (0→9)
⋮
7.abcdef 0.000001

```
ans = floorVal(n), i=0; double j = 0.1;
while (precision--) {
    while (ans * ans <= n) ans += j;
    ans -= j;
    j = j/10;
```

⑥ Valid Perfect Square

① If no. of factors are odd → Perfect Square
② Brute force ($O(\sqrt{N})$)
③ Binary Search $O(\log n)$

```
if((floorSQRT(n))^2 == n) return true
```

(7) Nth Root Binary Search