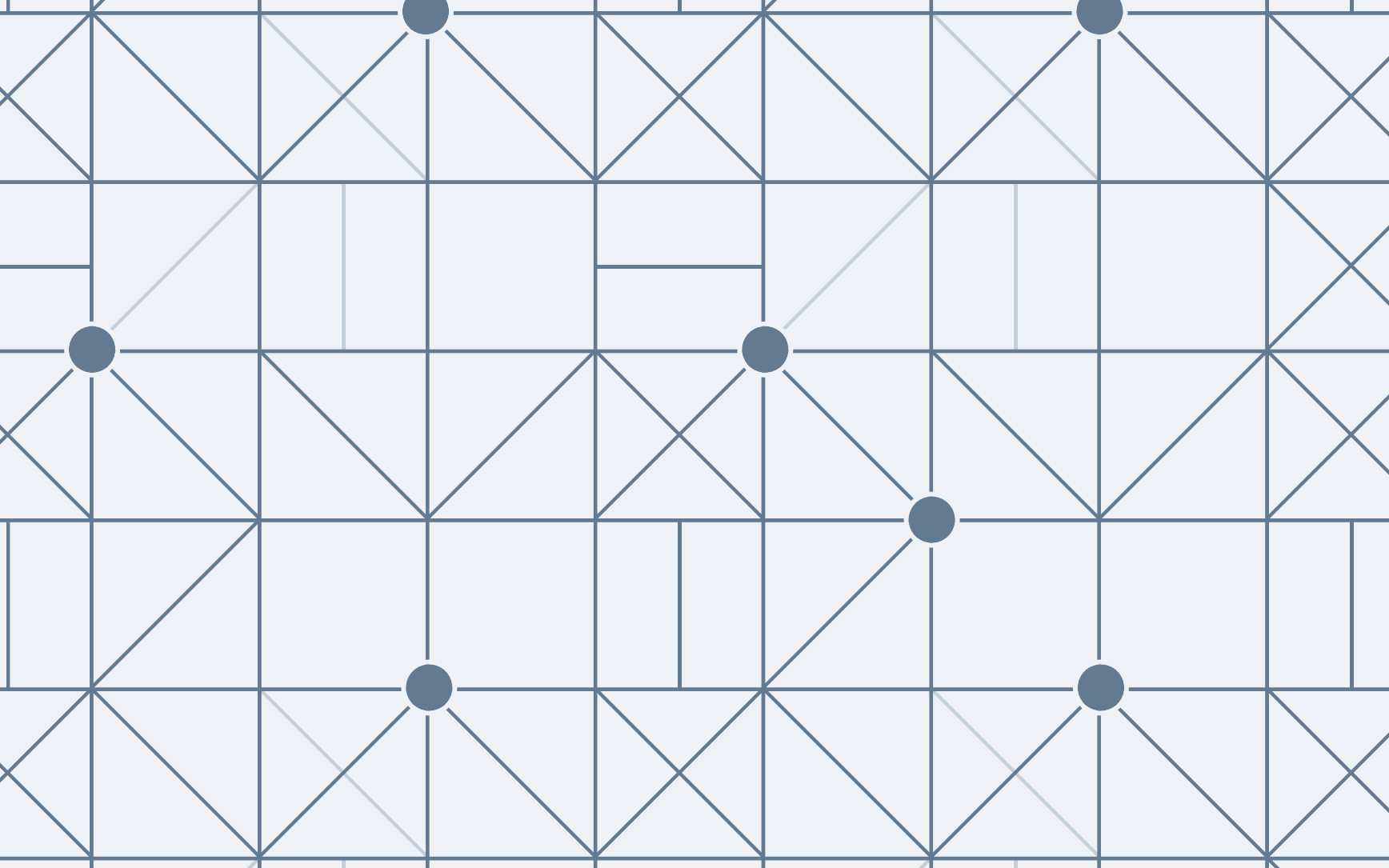


Binary search. { Algorithm } { Searching Algorithm }

int [] arr = { 1, 3, 7, 10, 11, 14, 20, 24 } target = 14

find?

Brute force.

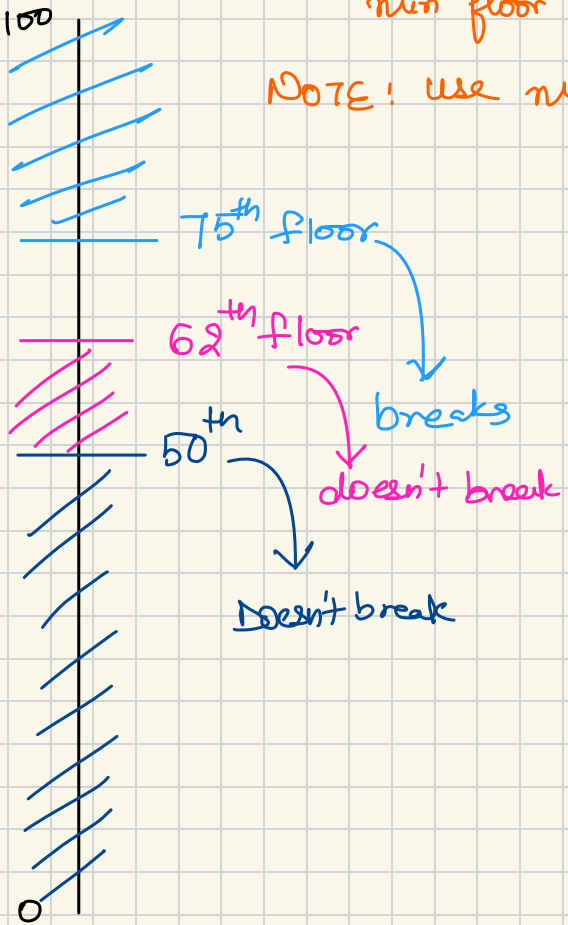
```
for(int i=0 ⟶ n)
{   if (arr[i] == target)
        return i;
}
```

} Linear search
TC : O(N)
SC : O(1) }

min floor from which the brick will break?

NOTE: use min^m no. of bricks.

Brute force:
100 Bricks needed



using 1st Brick, ↓ eliminated 50 floors

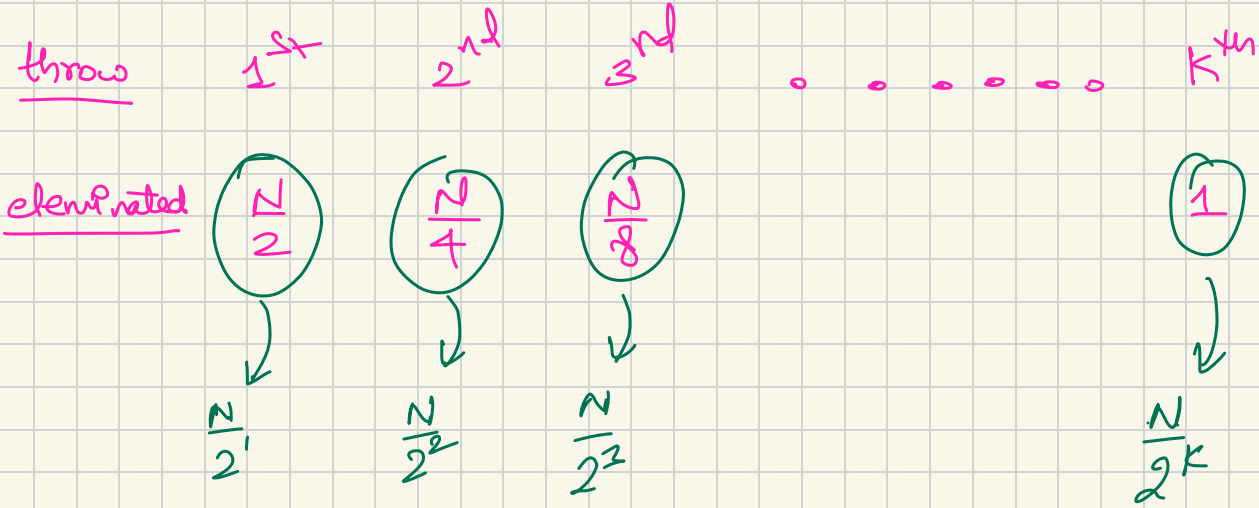
using 2nd Brick, ↓ eliminated 25 floors

using 3rd Brick, ↓ eliminated 12 floors

⋮ ⋮ ⋮

using kth Brick, ↓ eliminated 1 floor

N → floors



$$1 = \frac{N}{2^K}$$

$$2^K = N$$

taking $\log_2$ Both Sides

$$\log_2 2^K = \log_2 N$$

$$K \times \log_2 2 = \log_2 N$$

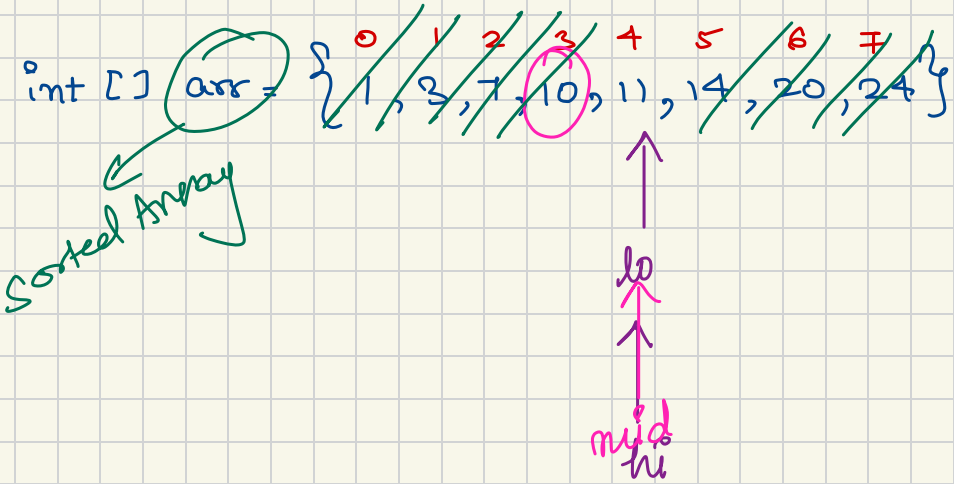
$$\log_2 2 = 1$$

$$K = \log_2 N$$

Hence, No. of throws | Brick is

$$= \log_2 N$$

$$\log_2(100) = 2\log_2 10 = 2 \times 3.1 \approx 7$$



Tc : $O(\log_2 N)$

Sc : $O(1)$

Case 1: arr[mid] == target

↳ got my ans!

Case 2: arr[mid] > target

↳ eliminate half having bigger values

Case 2: arr[mid] < target

↳ eliminate half having smaller elements

Binary Search Algorithm

1. Step 1: Define your range of search.
2. Step 2: Divide your range into 2 equal halves.
3. Step 3: by eliminating one of halves
4. Step 4: Update your range as per the remaining half.
5. Step 5: again go back to Step 2 till you find your ans

$TC: O(\log_2 N)$ $SC: O(1)$

Binary Search

→ search region should be sorted } ×

region should be in such way, where from starting at one point you can decide to eliminate one half and take another half

TC: O(log N) SC: O(1)

→ expected

99% of time @ques BS

Search insert position / ceil value / find just greater person

int[] arr = { 1 (0), 3 (1), 7 (2), 10 (3), 11 (4), 20 (5), 27 (6) }

Key = 2

Brute force

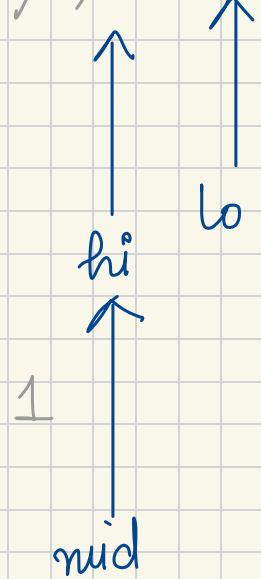
↳ Linear search { return first value greater than you, return that index }

TC : O(N)
SC : O(1)

int[] arr = { 1 , 3 , 7 , 10 , 11 , 20 , 27 }

0 1 2 3 4 5 6

Key = 2



ans = ~~7~~ ~~3~~ 1

case 1: arr[mid] == key

case 2: arr[mid] < key

move right;

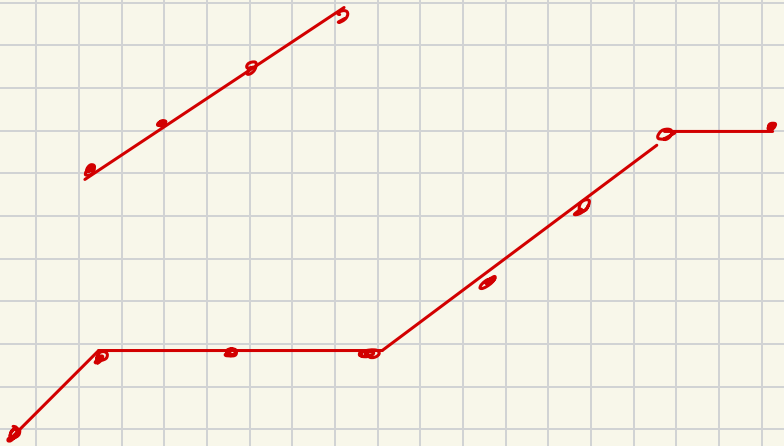
case 3: arr[mid] > key

ans = mid;

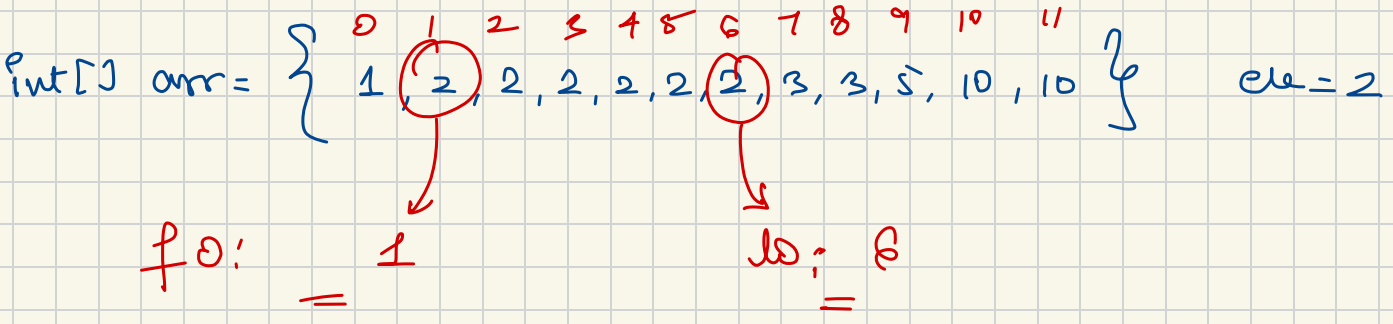
move left;

① inc. array
② non. dec. array

} NOT SAME



First and last Position of an Element



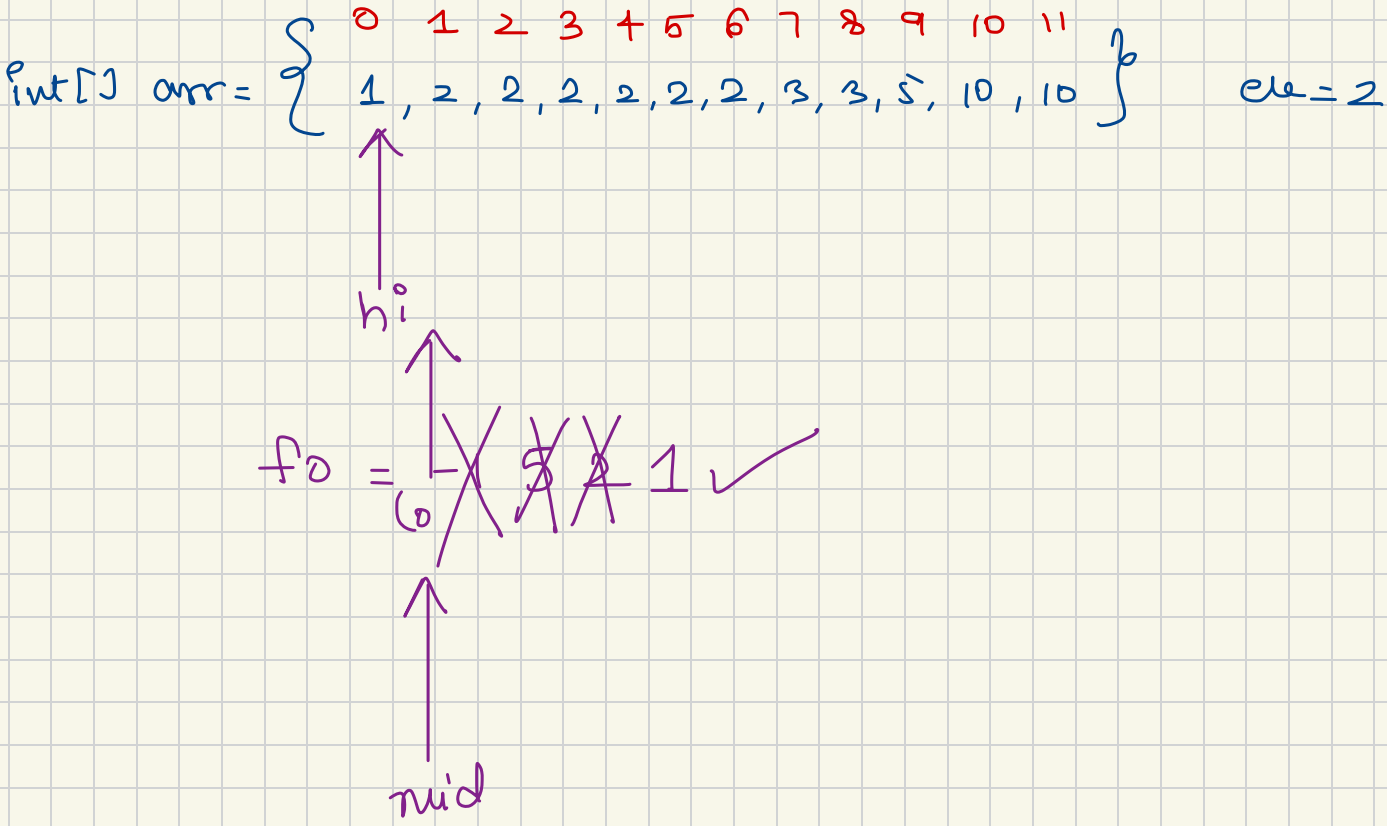
Brute force

```
int lo = -1;
for(int i = 0 → n)
    if(arr[i] == ele)
        lo = i;
```

TC: O(N)

SC: O(1)

int[] arr = { 1, 2, 2, 2, 2, 2, 2, 3, 3, 5, 10, 10 } ele = 2



Square Root

int x

find sqrt(x)

Case 1 : x is a perfect square $\longrightarrow \sqrt{x}$

Case 2 : x is not a perfect square $\longrightarrow$ floor($\sqrt{x}$)

$x = 100$

$ans = \sqrt{100} = 10$

$x = 9$

$ans = 3$

$x = 10$

$ans = 3$

$x = 30$

$ans = 5$

Square Root

Brute force

```
for(int i = 1; i <= x; i++)
{
    if(i * i <= x)
        ans = i;
}
```

TC : O(x)
SC : O(1)

Better.

```
int ans = -1;
for(int i=0; i*i<=x; i++)
{
    ans = i;
}
```

TC : O(sqrt(x))

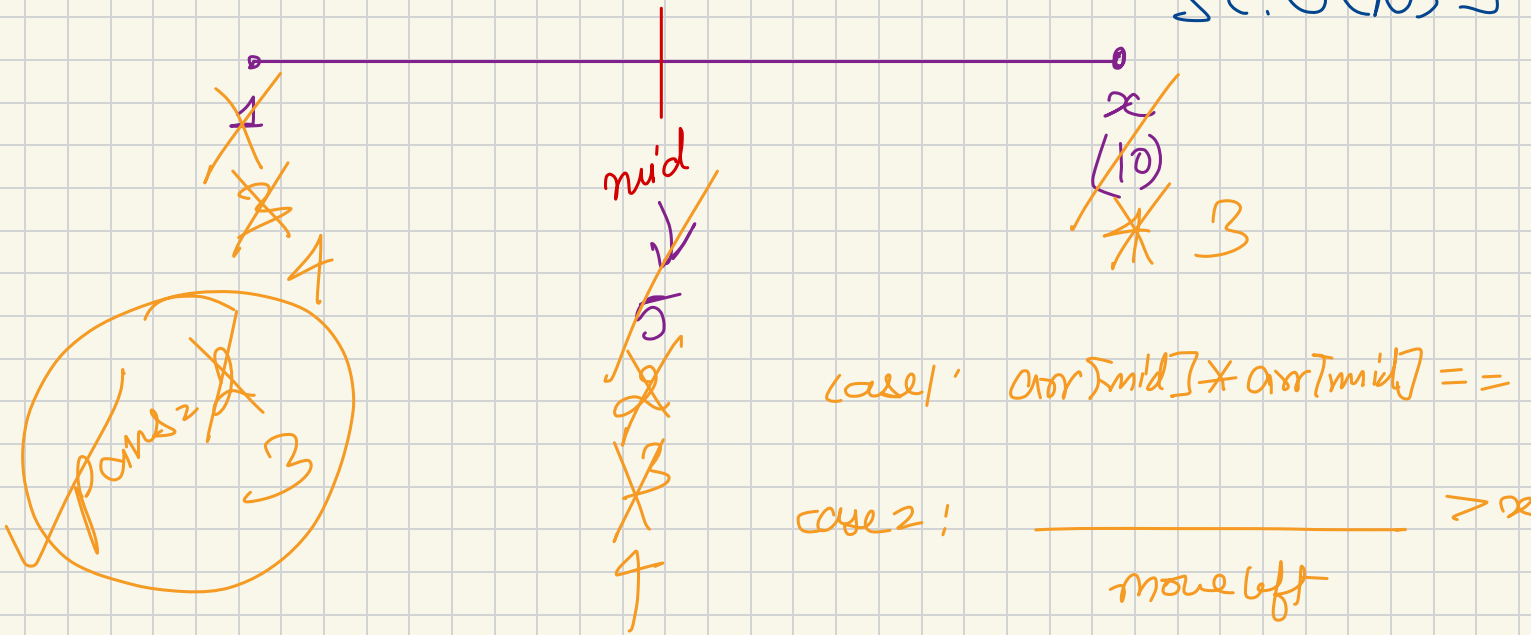
SC : O(1)

Better?

$x = 10$

$TC : O(\log_2 x)$

$SC : O(N)$



case1 : arr[mid] * arr[mid] == x

case2 : _____ > x

move left

Case 3 : _____ < x

pow = arr[mid]

move right.