Binary Search

```
int[] arr = [1, 3, 7, 10, 11, 14, 20]
            0  1  2  3   4   5   6
```

Sorted

target = 14

search() → 5

Brute-force

```
for(int i = 0; i < arr.length; i++)
{
    if (arr[i] == target)
        return i;
}
```

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

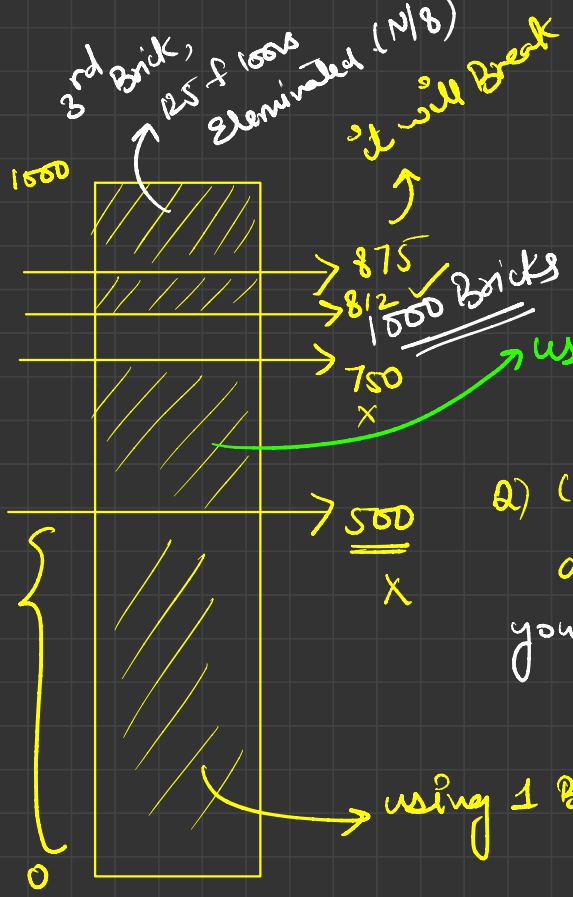
} Linear Search

Sorted Array ⇝ searching is possible in TC : O(logN)

↓

Binary Search

780



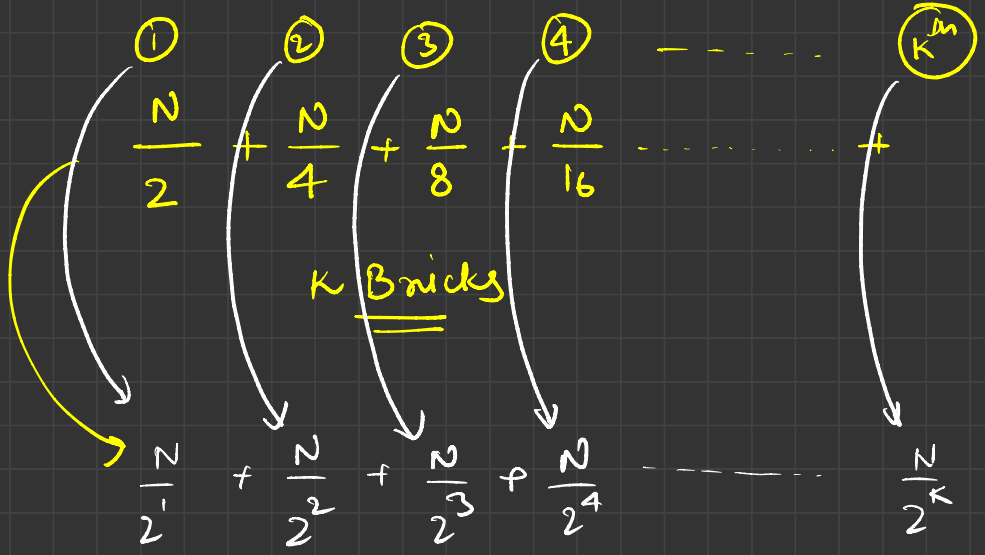
"it will Break"

1000 Bricks

using 2nd Brick, I eliminated 250 (N/4) floors

using 1 Brick, I eliminated 500 (N/2) floors

Q) Can you tell from which floor if you through a Brick, it will be break.
you have to min. number of Bricks.



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

$$N = 2^K$$

$$\log_2 N = K$$

$$\log_{10} N = \log_{10} 2^K$$

$$\log_{10} N = K \log_{10} 2$$

$$\frac{\log_{10} N}{\log_{10} 2} = K$$

$$\log_2 N = K$$

So, I can say we need $\log_2 N$ Bricks,

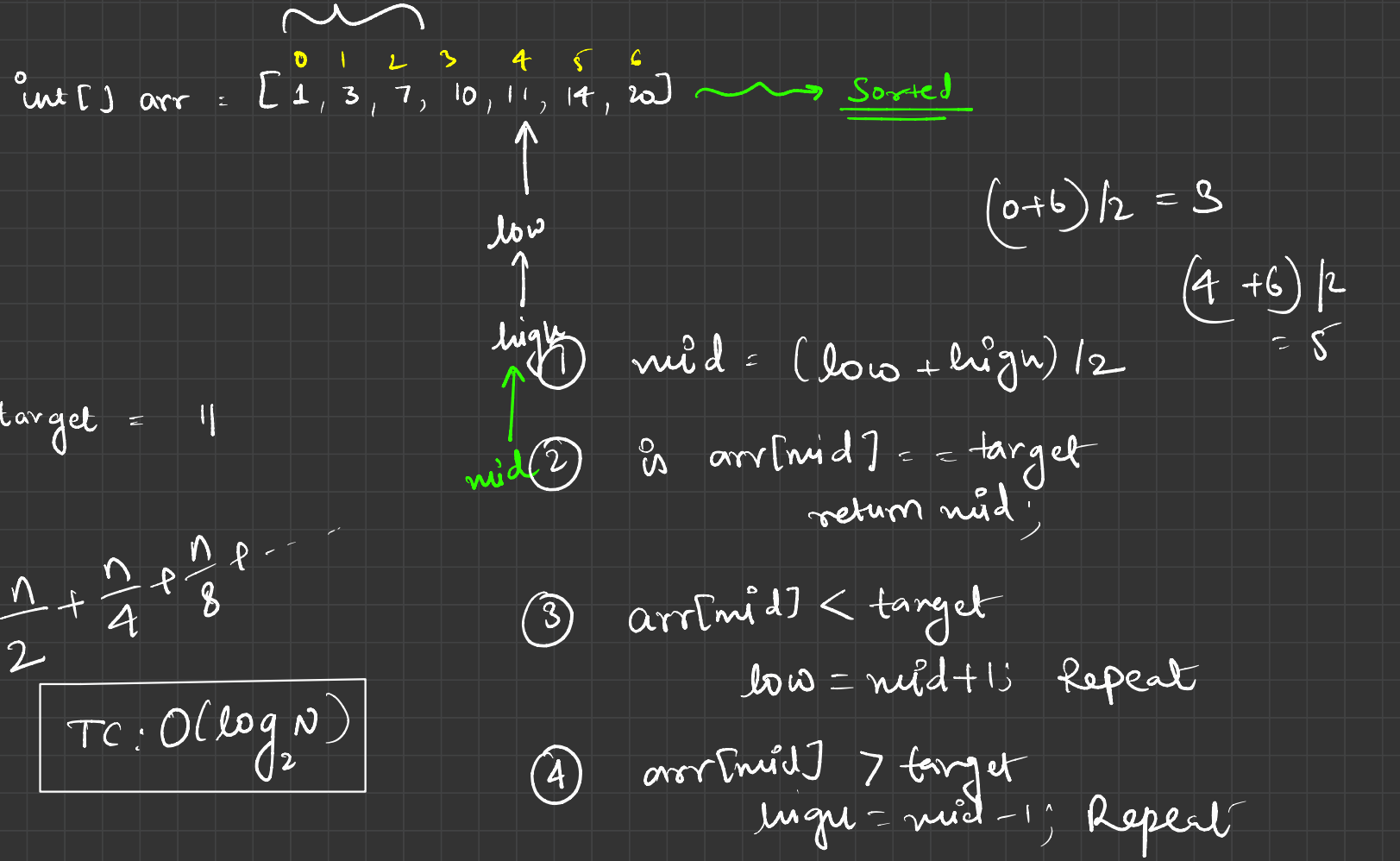
Hence, $\boxed{TC: O(\log_2 N)}$

$\log_2 1000 = \log_2 10^3$

$= 3 \times \log_2 10$

$= 3 \times 3.32$

$\approx 9.96 \approx 10$

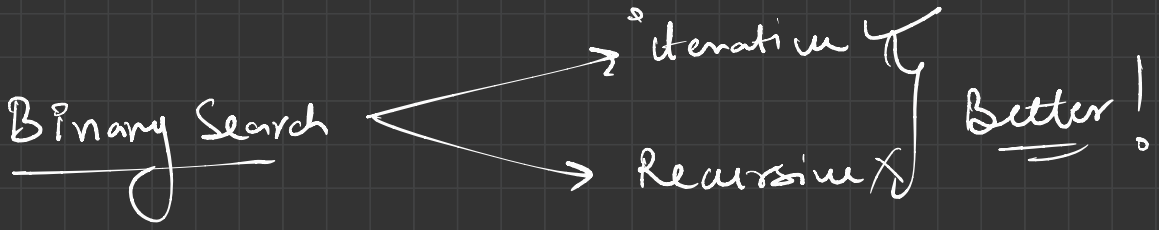

0 1 2 3 4 5 6
int[] arr = [1, 3, 7, 10, 11, 14, 20]
Sorted
low
high
mid
target = 11
(0+6)/2 = 3
(4+6)/2 = 5
① mid = (low + high)/2
② is arr[mid] == target
return mid;
③ arr[mid] < target
low = mid+1; Repeat
④ arr[mid] > target
high = mid-1; Repeat
n/2 + n/4 + n/8 + ...
TC: O(log2 N)

Binary Search

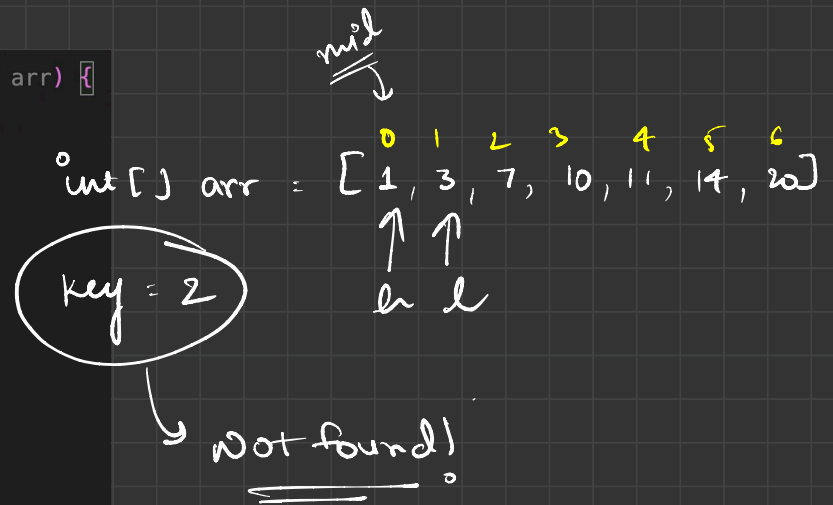
① the given array will be sorted.

- If given a sorted Array
- Expected TC: $O(\log_2 N)$

→ 99% it is Binary Search!


Binary Search
Iterative
Recursive
Better!

```java
public static int BinarySearchIterative(int key, int[] arr) {
    int low = 0;
    int high = arr.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] > key) {
            high = mid - 1;
        } else {
            low = mid + 1;
        }
    }

    return -1;
}
```



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

```java
public static int BinarySearchRecursive(int key, int[] arr, int low, int high) {
    if (low > high) {
        return -1;
    }

    int mid = (low + high) / 2;
    if (arr[mid] == key) {
        return mid;
    } else if (arr[mid] > key) {
        return BinarySearchRecursive(key, arr, low, mid - 1);
    } else {
        return BinarySearchRecursive(key, arr, mid + 1, high);
    }
}
```

① ② ③ ④ ⑤

low = 0, high = 6

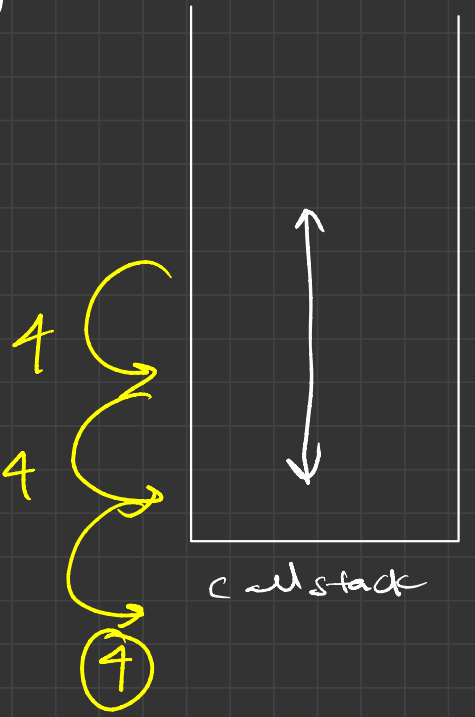
int[] arr = [1, 3, 7, 10, 11, 14, 20]

0 1 2 3 4 5 6

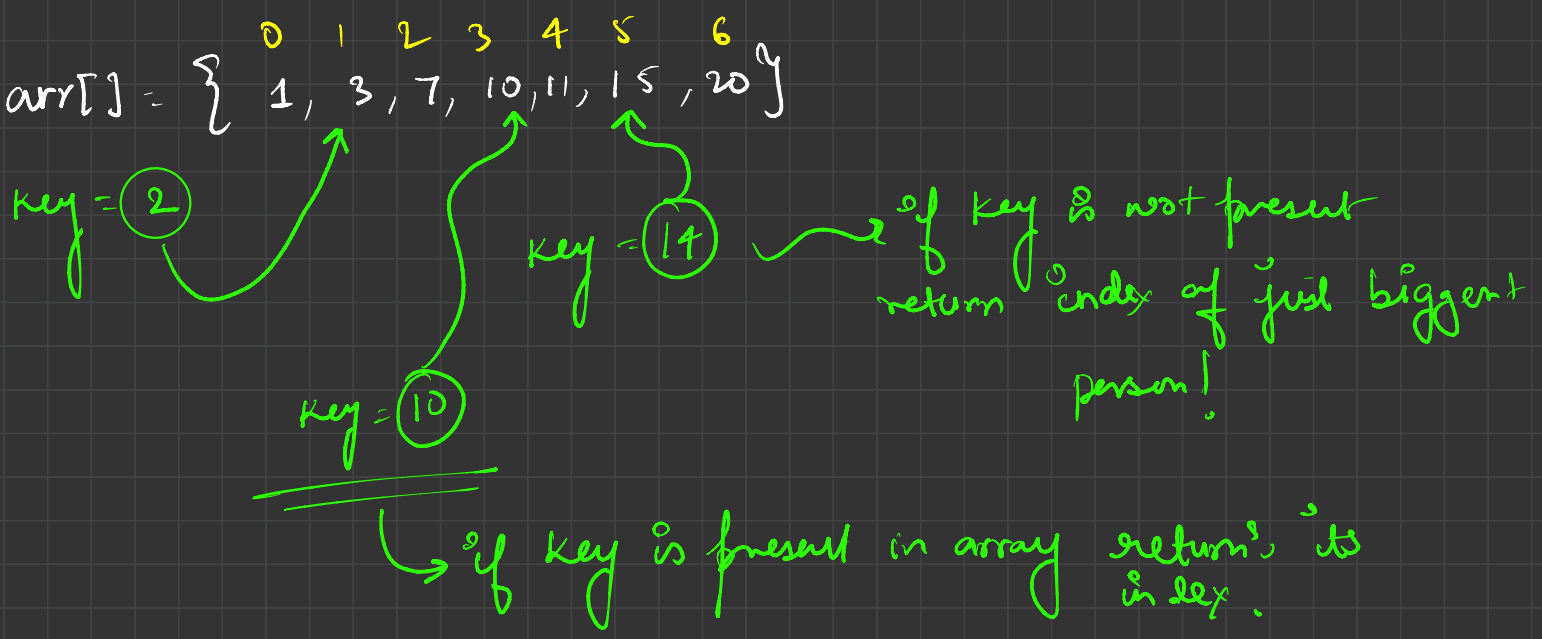
key = 11

$T(n): O(\log_2 n)$

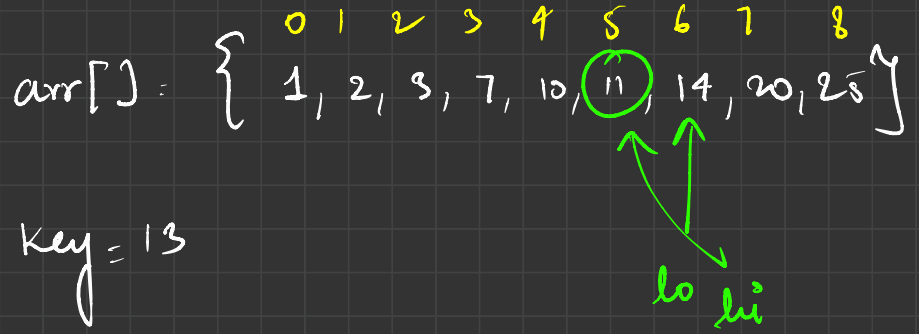
$S(n): O(\log_2 n)$



Search Insert Position



find ⟶ index of ceil value



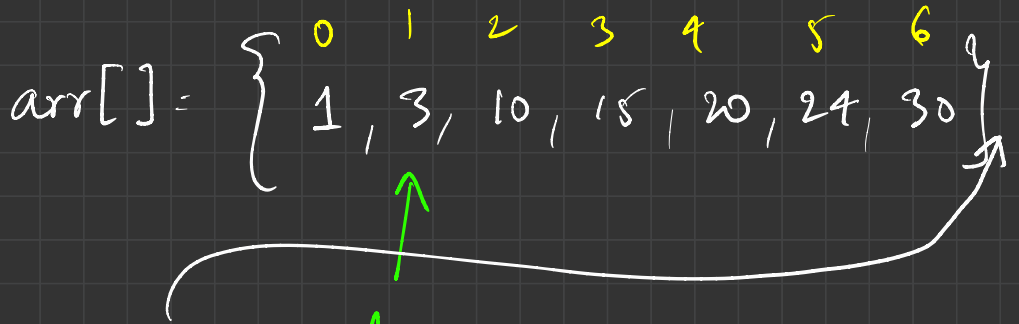
{ person just greater or equal to

Key = 13

potentialAns;

19

```
if (arr[mid] == key)

else if (arr[mid] > key)
    potentialAns = mid;
    hi = mid-1;
else
    lo = mid+1;
```



pAns

~~3~~ 1

```
if (arr[mid] == key)
    return mid
else if (arr[mid] > key)
    potentialAns = mid;
    hi = mid-1;
else
    lo = mid+1;
```

find first and last position of Element

	0	1	2	3	4	5	6	7	8	9	10	11
arr[] = {	1,	1,	2,	2,	2,	3,	4,	5,	7,	7,	10,	11 }

key = 2

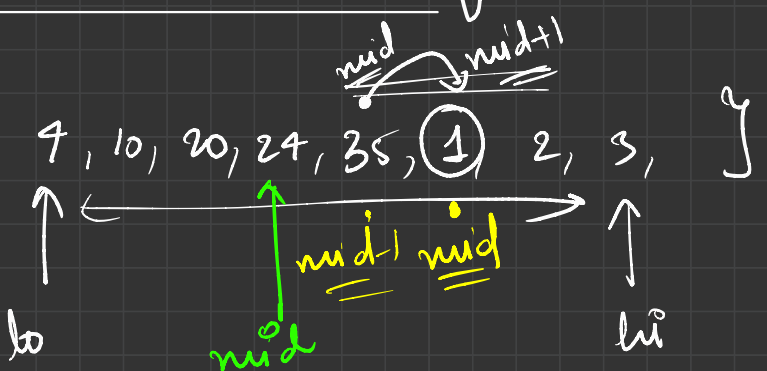
```
int fo = -1;
while(lo <= hi)
{
  int mid = (lo+hi)/2;
  if(arr[mid] == key){
      fo = mid;
      hi = mid-1;
  }
  else if(arr[mid] > key)
      hi = mid-1;
  else
      lo = mid+1;
}
```

{ TC : O(log N) base 2

SC : O(1)

Find Minimum in a Rotated Sorted Array

int[] = {



arr[mid+1] < arr[mid]

ans

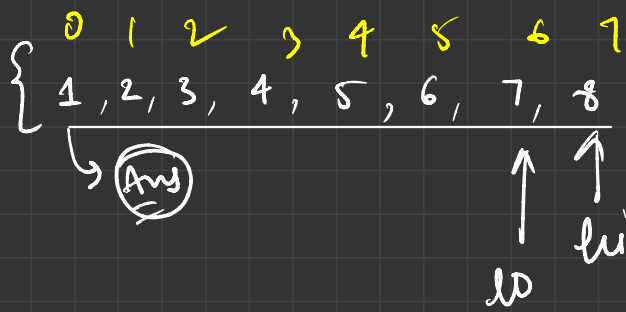
arr[mid-1] > arr[mid]

ans

$\log_2 N$

TC:

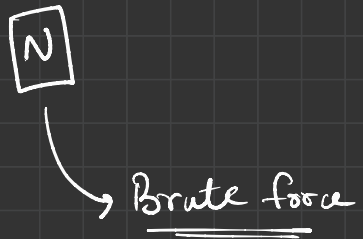
```
while ( lo <= hi)
{
    int mid = (lo+hi)/2;
    if( arr[mid+1] < arr[mid])
        return mid+1;
    else if( arr[mid-1] > arr[mid])
        return mid;
    else if ( arr[mid] >= arr[low])
        lo = mid+1;
    else
        hi = mid-1;
}
```




```
while ( lo <= hi)
{
    int mid = (lo+hi)/2;
    if( arr[mid+1] < arr[mid])
        return mid+1;
    else if( arr[mid-1] > arr[mid])
        return mid;
    else if ( arr[mid] >= arr[low])
        lo = mid+1;
    else
        hi = mid-1;
}
```

```
if (arr[0] <= arr[n-1])
    return 0;
```

Square Root Of A Number

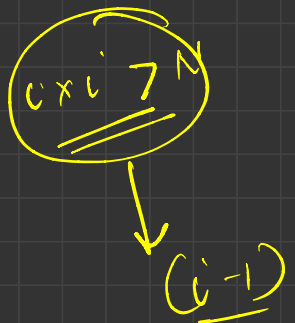


$1 \times 1 = 1 \leq 9$

$2 \times 2 = 4 \leq 9$

$3 \times 3 = 9 \leq 9$

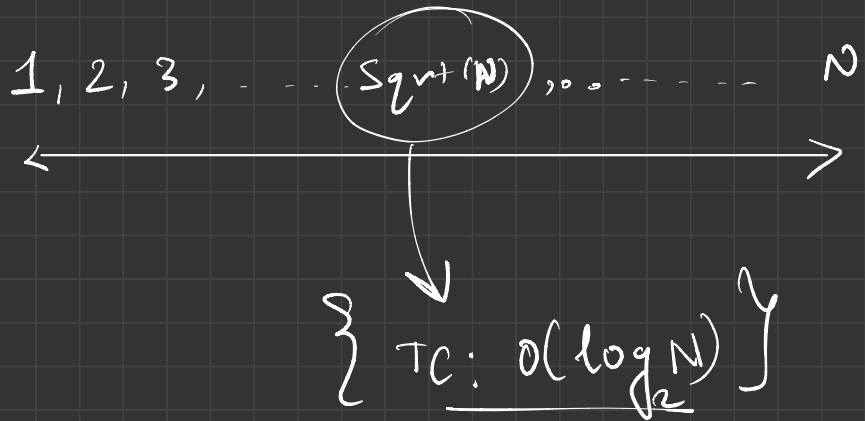
$4 \times 4 = 16 \leq 9$



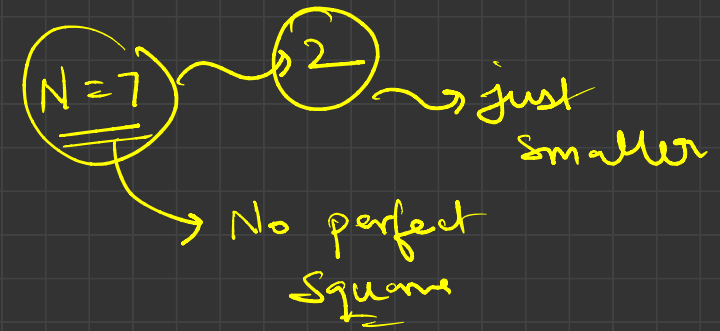

```
Put i = 1;
while( i*i <= N)
    i++;
return i-1;
```

TC: $O(\sqrt{N})$

1, 2, 3, 4 ..... Sqrt(N)


1, 2, 3, . . . Sqrt(N) . . . . . N
TC: O(log N)

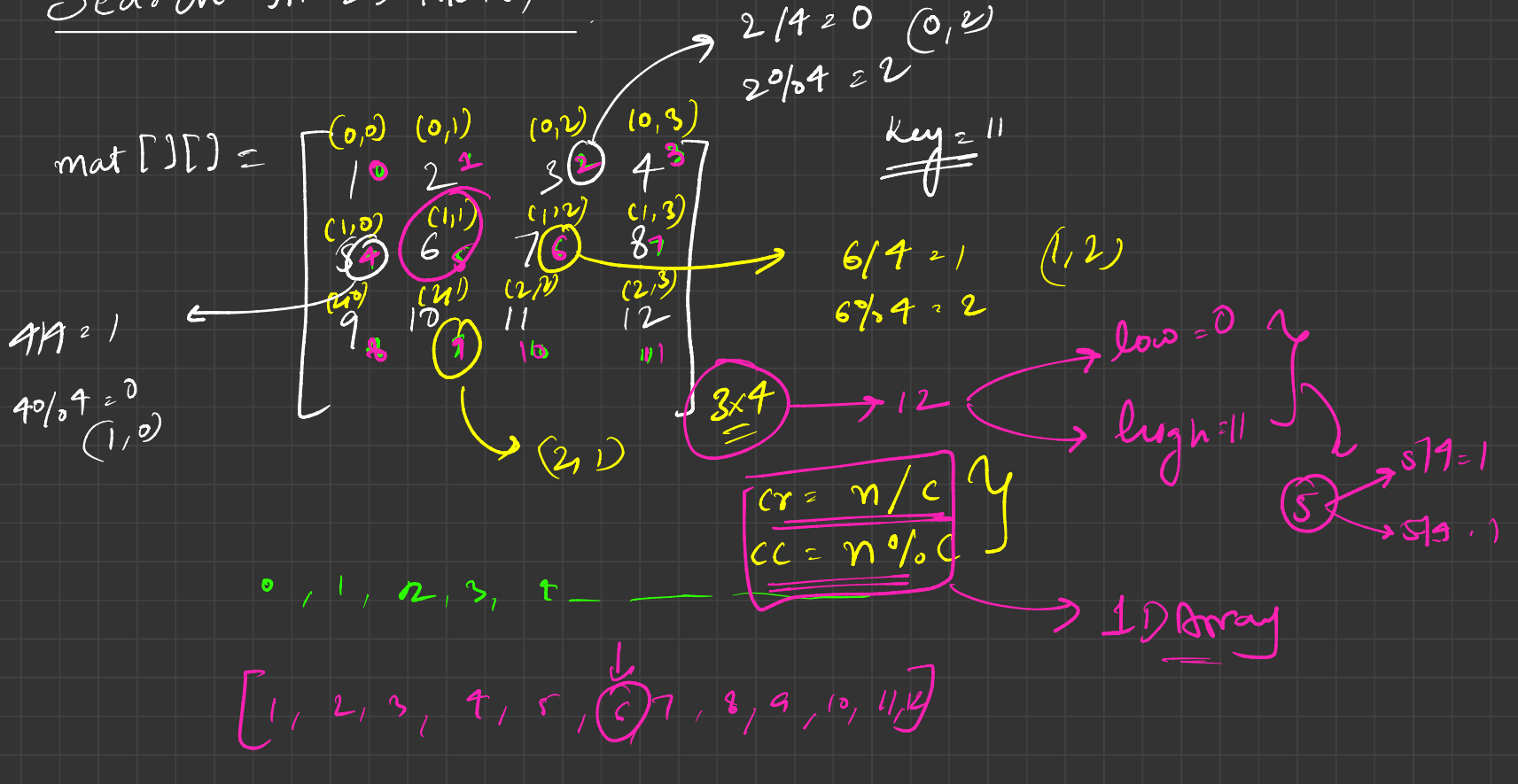
```
int psqrt = 0;
lo = 1, hi = N;

while( lo <= hi)
{ int mid = (lo + hi)/2;

  if(mid * mid == N)
      return mid;

  else if( mid*mid > N)
      hi = mid - 1;

  else psqrt = mid
       lo = mid +1;
}
```

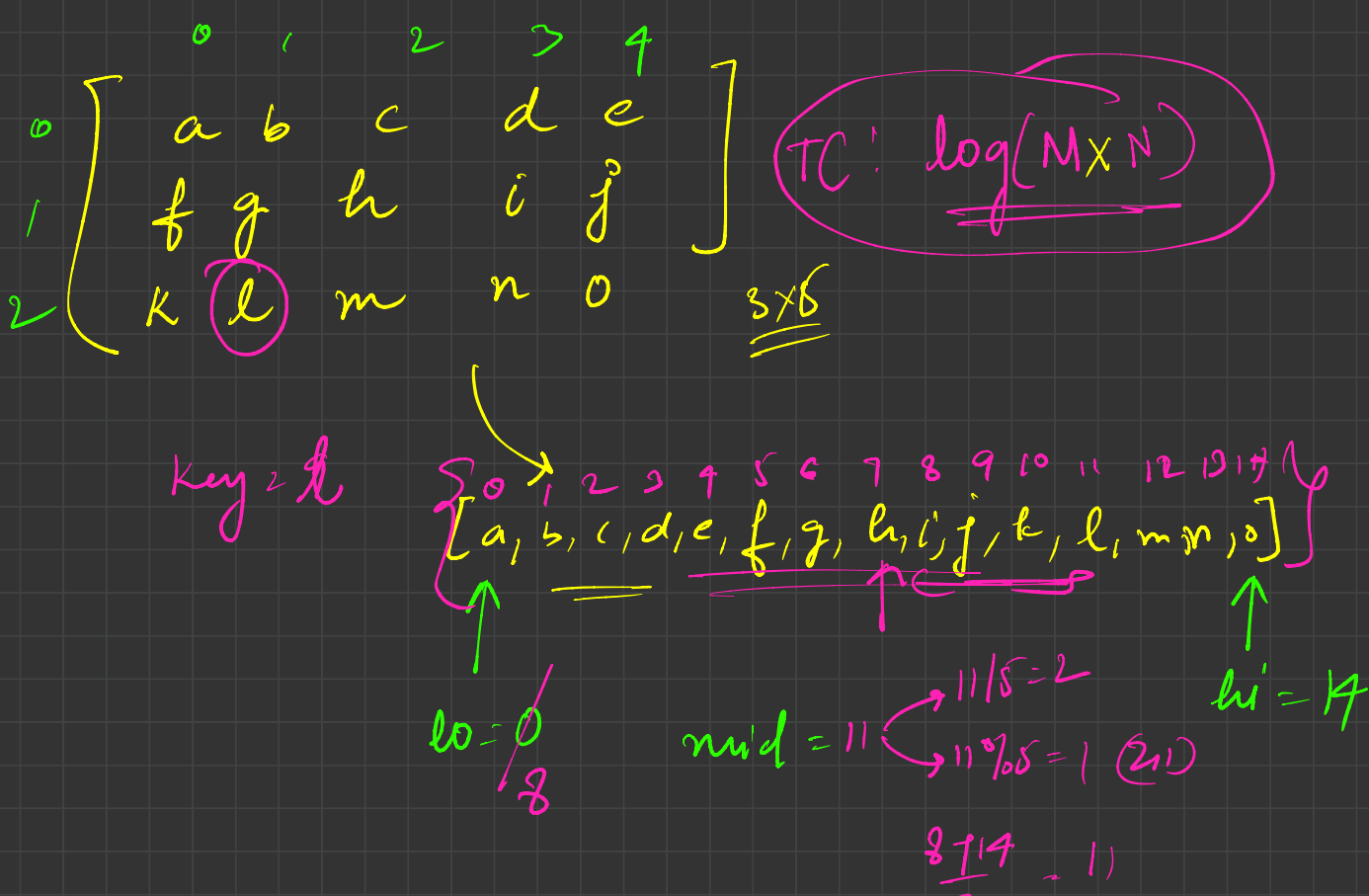


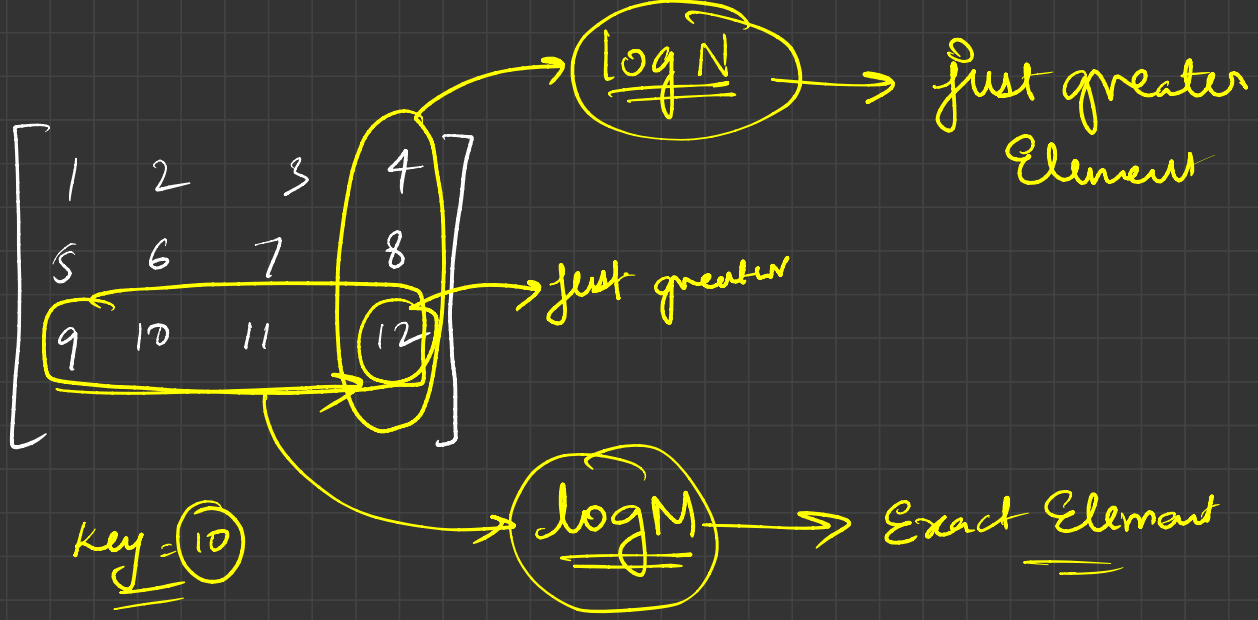
lo = 3, hi = 3/2

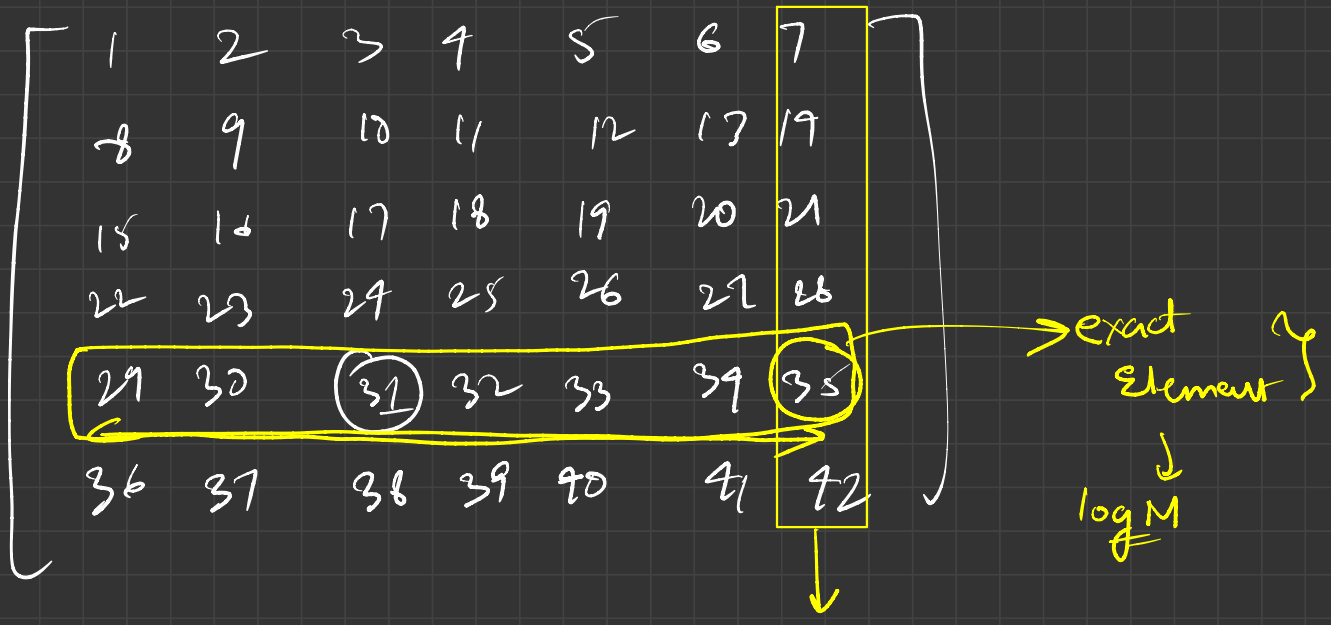
mid = 3

Search in 2D Matrix




0 1 2 3 4
0 a b c d e
1 f g h i j
2 k l m n o
3×5
TC: log(M×N)
Key = l
[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
[a, b, c, d, e, f, g, h, i, j, k, l, m, n, o]
lo = 0
mid = 11
11/5 = 2
11%5 = 1 (2,1)
hi = 14
8 14
2
11


logN
just greater Element
1 2 3 4
5 6 7 8
9 10 11 12
just greater
logM
Exact Element
Key = 10



TC: $O(\log N + \log M)$

$\hookrightarrow O(\log N * M)$