Binary Search.

$$\text{int [] arr} = \left\{ \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1, & 3, & 7, & 10, & 11, & 14, & 20, & 30 \end{array} \right\}$$

→ sorted array

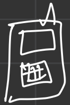
Key = 14

Brute force

linear Search

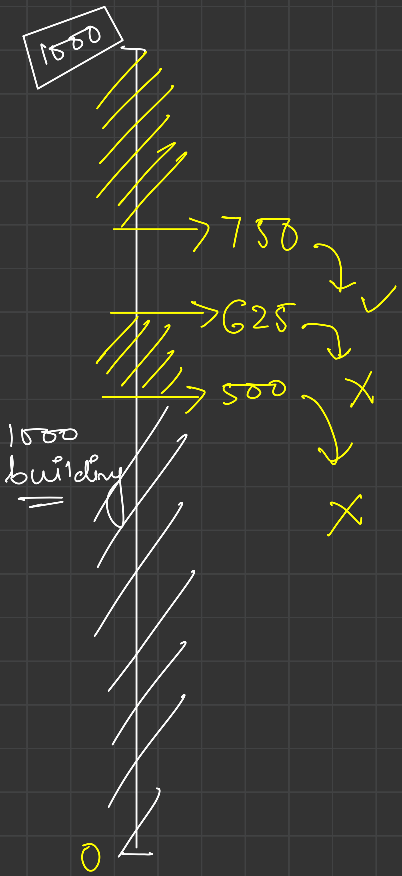
TC: O(N)
SC: O(1)

```
for (int i = 0 ; i < n; i++)
{
    if (arr[i] == key)
        return i;
}
return -1;
```



→ min^n floor from which phone will break

✓ 1000 → Brute force!



using only 1 phone you elemenated 500 floor

using 2nd phone you eleminated 250 floors

" 3rd " " " 125 floors

$$\begin{tikzcd}
\frac{N}{2} \arrow[d, bend right, "1"'] & \frac{N}{4} \arrow[d, bend right, "2"'] & \frac{N}{8} \arrow[d, bend right, "3"'] & \frac{N}{16} \arrow[d, bend right, "4"'] \arrow[r, dashed, no head] & {} \arrow[d, bend right, "K^{th}"'] \\
\frac{N}{2^1} & \frac{N}{2^2} & \frac{N}{2^3} & \frac{N}{2^4} & \frac{N}{2^K}
\end{tikzcd}$$

$K^{th} \rightarrow \boxed{\log N} \rightarrow$ phones

$\log_2 1000$

3.

$3 \log_2 10$

$3 \times 3\ 1$

$\approx \boxed{10}$ ← approx

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

$$N = 2^K$$

taking $\log_2$ both side

$$\log_2 N = \log_2 2^K \rightarrow \boxed{\log_2 N = K}$$

mid

$$\text{int [] arr} = \left\{ \begin{array}{cccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1, & 3, & 7, & 10, & 11, & 14, & 20, & 30 \end{array} \right\} \rightarrow \underline{\text{sorted array}}$$

Key = 11

si ✓

ei ✓

$$\log_2 8 = \log_2 2^3 = 3 \log_2 2 = \boxed{3}$$

TC: O(log N) }

SC: O(1)

Binary Search

```
{
    if arr[mid] == key
        return mid
    if arr[mid] < key          { move right }
        si = mid + 1;
    if arr[mid] > key
        ei = mid - 1;
}
```

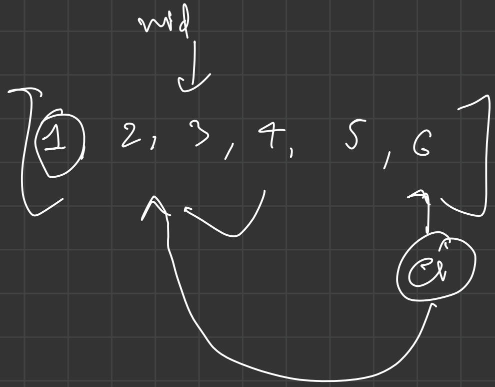
Binary Search

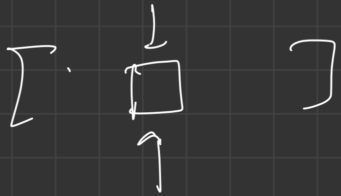
① It's always applicable over a sorted region }

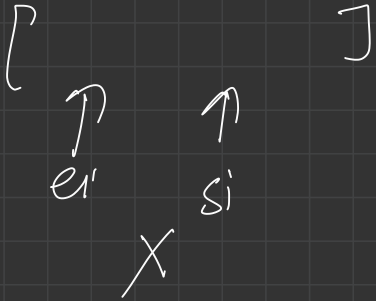
→ ✓ Sorted Region | Array

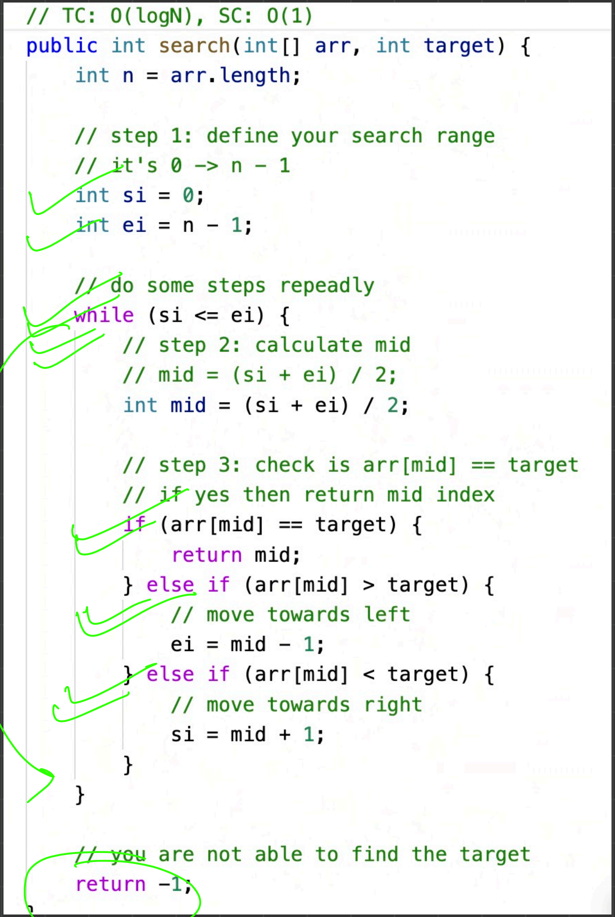
→ ✓ TC : O(log N) (Expected)

} 99% time it is a Binary Search


mid
1 2, 3, 4, 5, 6
ei




ei
si
X



```java
// TC: O(logN), SC: O(1)
public int search(int[] arr, int target) {
    int n = arr.length;

    // step 1: define your search range
    // it's 0 -> n - 1
    int si = 0;
    int ei = n - 1;

    // do some steps repeadly
    while (si <= ei) {
        // step 2: calculate mid
        // mid = (si + ei) / 2;
        int mid = (si + ei) / 2;

        // step 3: check is arr[mid] == target
        // if yes then return mid index
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] > target) {
            // move towards left
            ei = mid - 1;
        } else if (arr[mid] < target) {
            // move towards right
            si = mid + 1;
        }
    }

    // you are not able to find the target
    return -1;
}
```

0 1 2 3 4 5 6

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

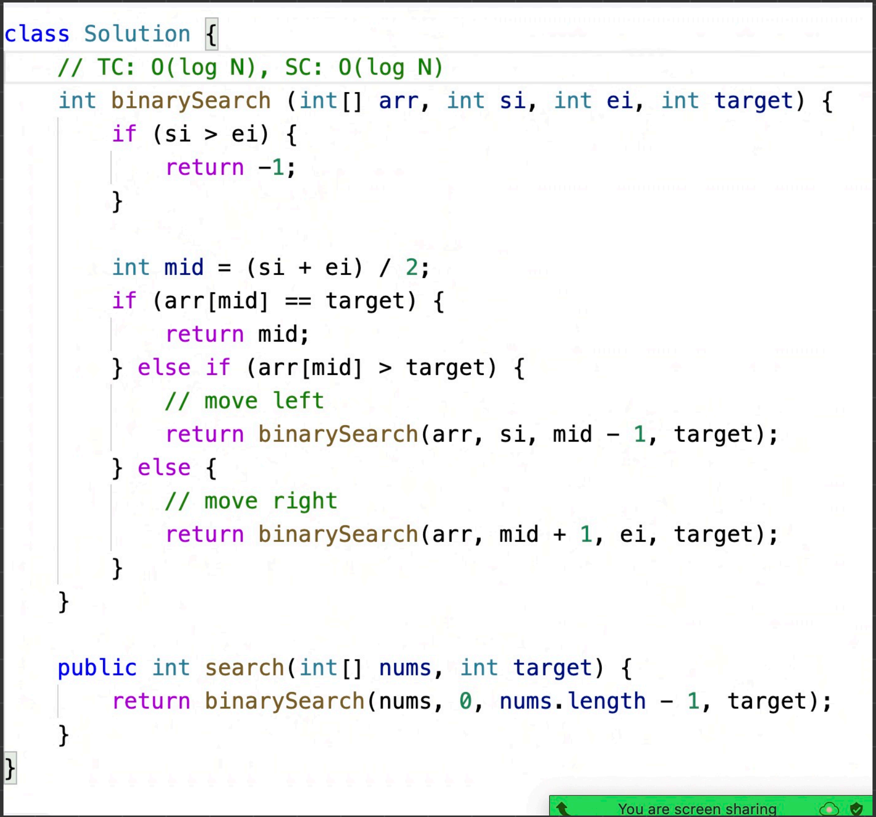
ei si

Key = 2

TC: $O(\log_2 N)$

SC: $O(1)$

Recursively



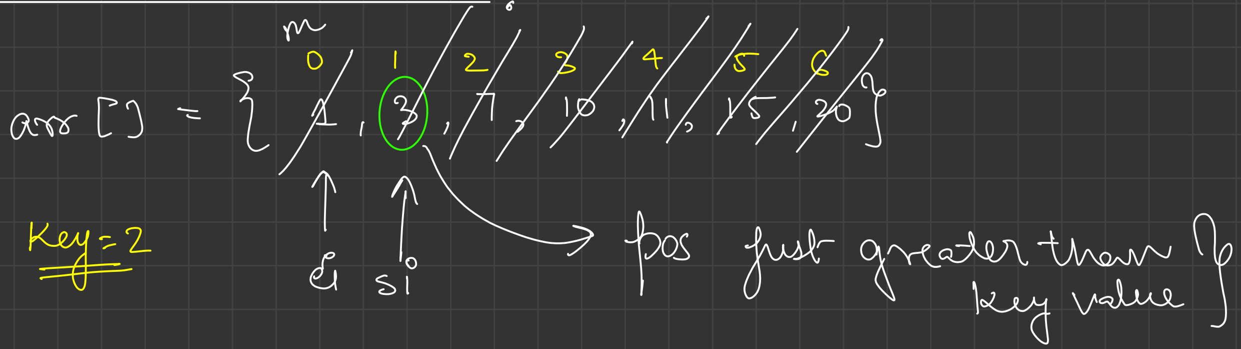
```java
class Solution {
    // TC: O(log N), SC: O(log N)
    int binarySearch (int[] arr, int si, int ei, int target) {
        if (si > ei) {
            return -1;
        }

        int mid = (si + ei) / 2;
        if (arr[mid] == target) {
            return mid;
        } else if (arr[mid] > target) {
            // move left
            return binarySearch(arr, si, mid - 1, target);
        } else {
            // move right
            return binarySearch(arr, mid + 1, ei, target);
        }
    }

    public int search(int[] nums, int target) {
        return binarySearch(nums, 0, nums.length - 1, target);
    }
}
```

Binary Search

→ iterative (is better)

→ recursive

Search Insert Position


m
0 1 2 3 4 5 6
arr[] = { ~~1~~ , 3 , ~~7~~ , ~~10~~ , ~~11~~ , ~~15~~ , ~~20~~ }
Key = 2
e
si
pos just greater than key value

{ceil value}

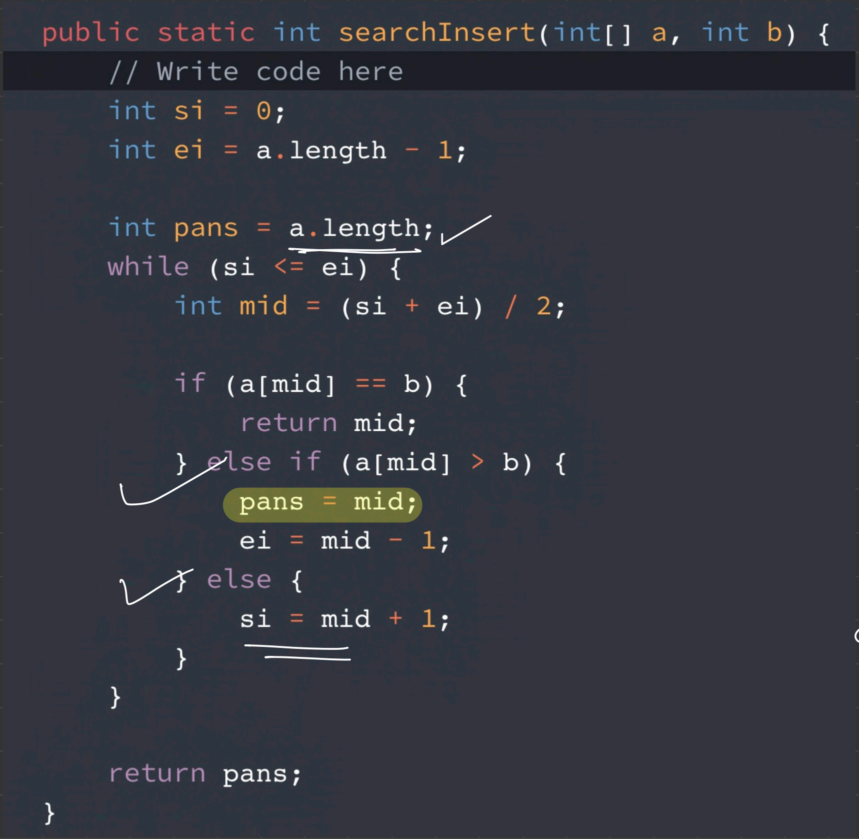
arr[mid] == key

arr[mid] > key

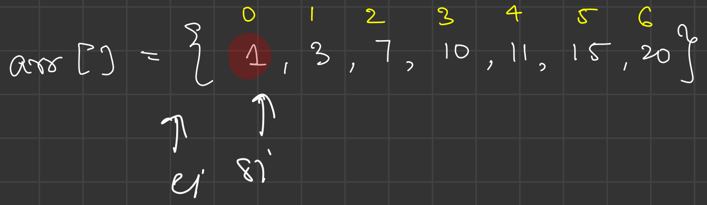
move left

arr[mid] < key

move right



```java
public static int searchInsert(int[] a, int b) {
    // Write code here
    int si = 0;
    int ei = a.length - 1;

    int pans = a.length;
    while (si <= ei) {
        int mid = (si + ei) / 2;

        if (a[mid] == b) {
            return mid;
        } else if (a[mid] > b) {
            pans = mid;
            ei = mid - 1;
        } else {
            si = mid + 1;
        }
    }

    return pans;
}
```



target = -10

{ TC: O($\log_2 N$)

SC: O(1) }

find first and last Position of Element .

inc $\{1, 2, 5, 10, - - -$

non dec $\{1, 1, 1, 2, 3, 5, 5, 10, 10, 11, 11 - - -$

arr[] = {

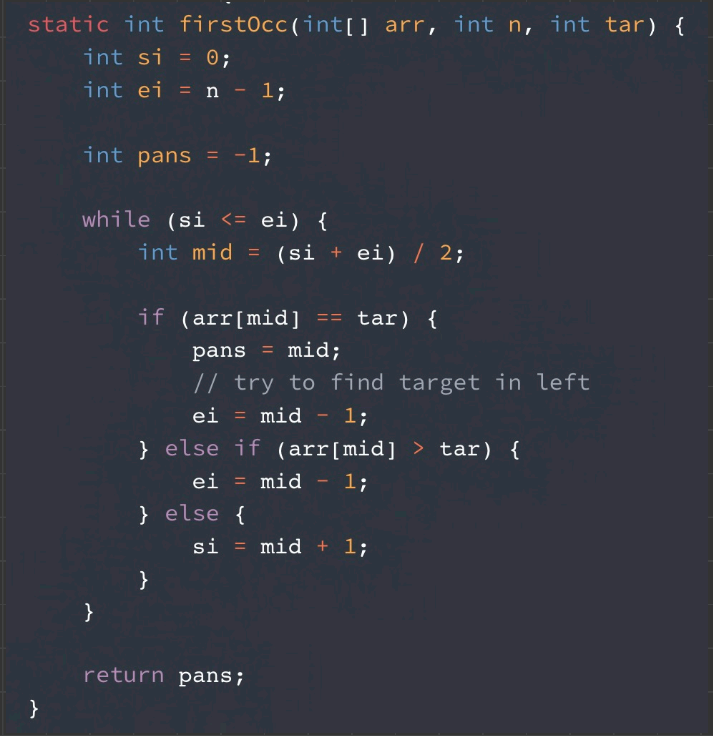
0	1	2	3	4	5	6	7	8	9	10	11
1,	1,	2,	2,	2,	3	4,	5,	7,	7,	10,	11

}

✓target - 2

{ first Occurance

```
arr[mid] == target
        pans = mid;
        ei = mid - 1;

arr[mid] > target
        ei = mid - 1;

arr[mid] < target
        si = mid + 1;
```



```
static int firstOcc(int[] arr, int n, int tar) {
    int si = 0;
    int ei = n - 1;

    int pans = -1;

    while (si <= ei) {
        int mid = (si + ei) / 2;

        if (arr[mid] == tar) {
            pans = mid;
            // try to find target in left
            ei = mid - 1;
        } else if (arr[mid] > tar) {
            ei = mid - 1;
        } else {
            si = mid + 1;
        }
    }

    return pans;
}
```

find Minimum in Rotated Sorted Array

arr[] :

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

```
if (arr[m] < arr[m-1])
        return m;
if (arr[m] > arr[m+1])
        return m+1;

if  arr[s] <= arr[m]          → left is sorted.

        si = m+1;

else

        ei = m-1;
```

left is not sorted ←

Square root of a Number

$N = 9 \longrightarrow \text{Sqrt}(9) = \boxed{3}$

Brute force :

```
for(i=1; i<=n; i++)
{
    if (i*i == N)
        return i;
}
```

TC : O(N)

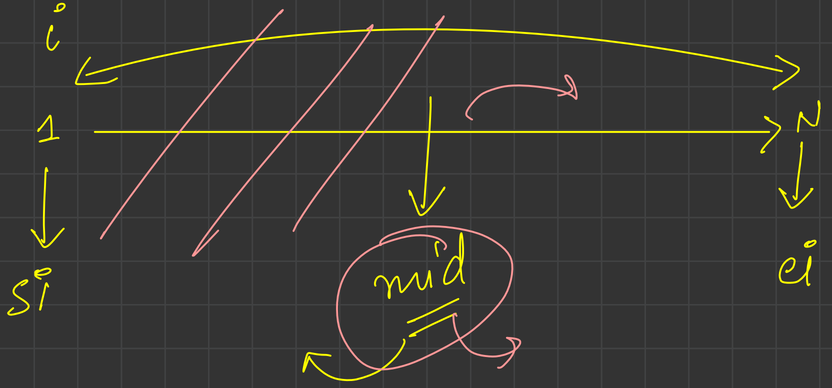
SC : O(1)

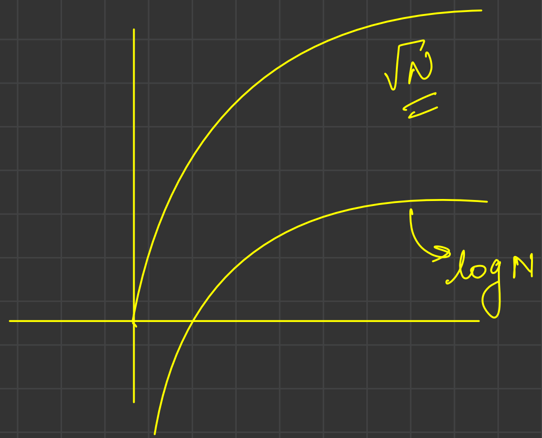
```
for(int i=1; i*i<=N; i++)
{
        if (i*i == N)
            return i;

}
```

✓

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

SC: $O(1)$





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

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

    else
        ans = mid* mid;
        si = mid+1;
}
```

TC: $O(\log_2 N)$

SC: $O(1)$

$\underline{\underline{N = 8}}$

$$\begin{tikzcd}
\text{sqrt}(\boxed{8}) = \boxed{2}. \quad \text{———} \arrow[d] \\
\checkmark \; \boxed{2}
\end{tikzcd}$$

Search in 2D Array

arr[ ][ ] :

$$\begin{array}{cccc}
(0,0) & (0,1) & (0,2) & (0,3) \\
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & \boxed{12} \\
13 & 14 & 15 & 16
\end{array}$$

$\rightarrow \log(N)$

$n \times m$

target = 10

$\log M$

$\log N + \log M$ ✓

$S.C. : O(1)$

$TC = O(\log N * M)$ ✓