Binary Search.

what is searching?

arr[] = { 1, 2, 5, 7, 13, 20, 25, 36 }

→ sorted (inc. order)

target = 20

find?

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

TC: O(N)

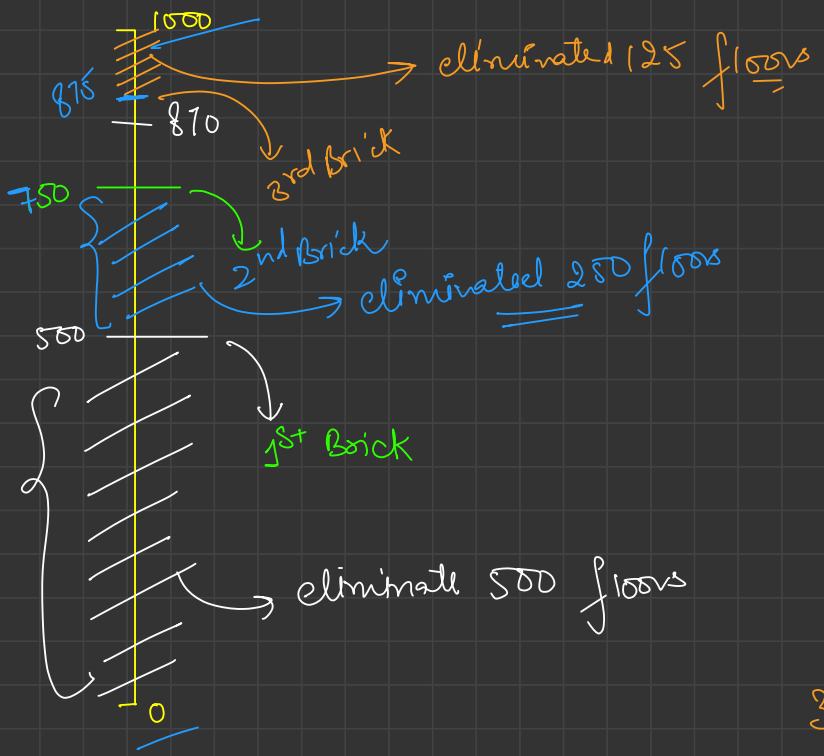
SC: O(1)

Linear Search!

Whenever you see sorted region, and you have to search something,

then 99% chances are this ques. is from BS

Tc : O(logN)



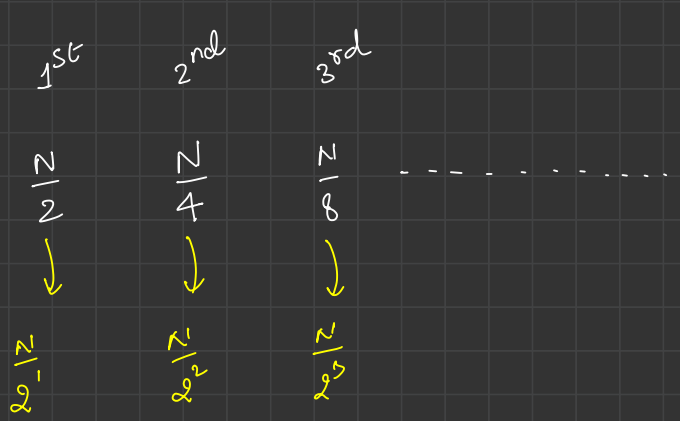
$\log_2(1000)$

$\log_2 10^3$

$3 \times \log_2 10$

$3 \times 3.1 \approx 9.3$

≈ 10 Bricks



Hence,

$\log_2 N$ Boats will be required!

In worst case, 1 person will be get eliminated and, rest 1 is your ans.

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

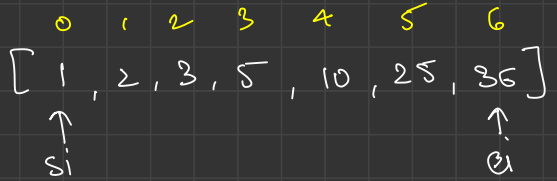
$$N = 2^K$$

Taking $\log_2$ both side

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

$$\log_2(N) = K \log_2 2 \quad (\log_2 2 = 1)$$

$$\boxed{\log_2 N = K}$$



target = 10

① Sorted region

↳ to Apply B.S

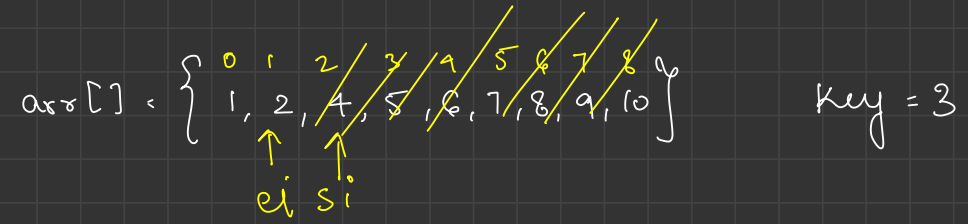
TC: $O(\log_2 N)$

SC: $O(1)$

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

    if (arr[mid] == target)
        return mid;

    else if (arr[mid] > target)
        ei = mid-1;

    else
        si = mid+1;
}
```

```java
// TC: O(log N), SC: O(1)
public static int findIndex(int key, int[] arr) {
    // step 1: define a search range
    int si = 0;
    int ei = arr.length - 1;

    // step 2: try finding till the search range is valid
    while (si <= ei) {

        // step 3: get mid point
        int mid = (si + ei) / 2;

        // step 4: check if arr[mid] equal to the key, else try eliminating half of the range
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] > key) {
            // eliminate right region and move left
            ei = mid - 1;
        } else {
            // eliminate left region and move right
            si = mid + 1;
        }
    }

    // key not present in the array
    return -1;
}
```

Find insert position in a Sorted Array



Basically we need index of ceil value of key

```
while (si <= ei)
{
    mid = (si+ei)/2;
    if (arr[mid] == key)
        return mid;
    else if (arr[mid+1] > key)
    {
        ceil = mid;
        ei = mid-1;
    }
    else
        si = mid+1;
}
```

```java
// TC: O(log N), SC: O(1)
public static int searchInsert(int[] a, int b) {
    // define search range
    int si = 0;
    int ei = a.length - 1;

    int ceil = a.length;

    while (si <= ei) {
        int mid = (si + ei) / 2;
        if (a[mid] == b) {
            return mid;
        } else if (a[mid] > b) {
            // move towards left, but as we are eliminating bigger values so update potential ceil index
            ceil = mid;
            ei = mid - 1;
        } else {
            // move right, and as we are eliminating smaller people so won't affect our ceil value
            si = mid + 1;
        }
    }

    return ceil;
}
```

find first Occ

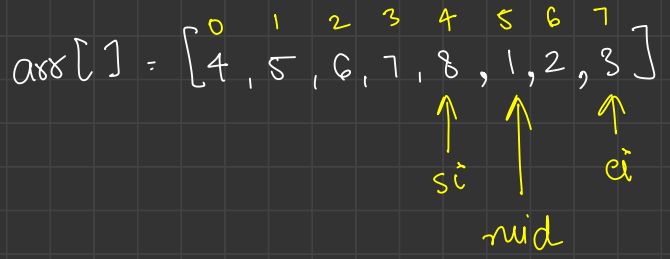
0	1	2	3	4	5	6	7	8	9	10
1	2	2	3	3	3	3	3	4	5	6

arr[] : [1, 2, 2, 3, 3, 3, 3, 3, 4, 5, 6]

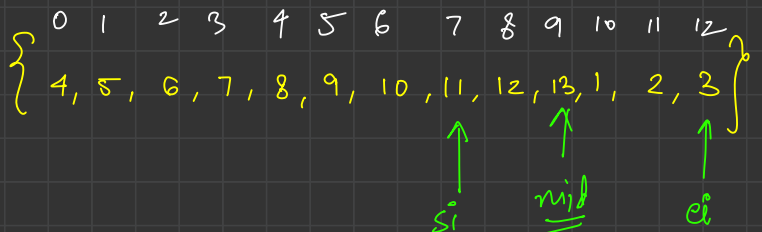
Key = 3

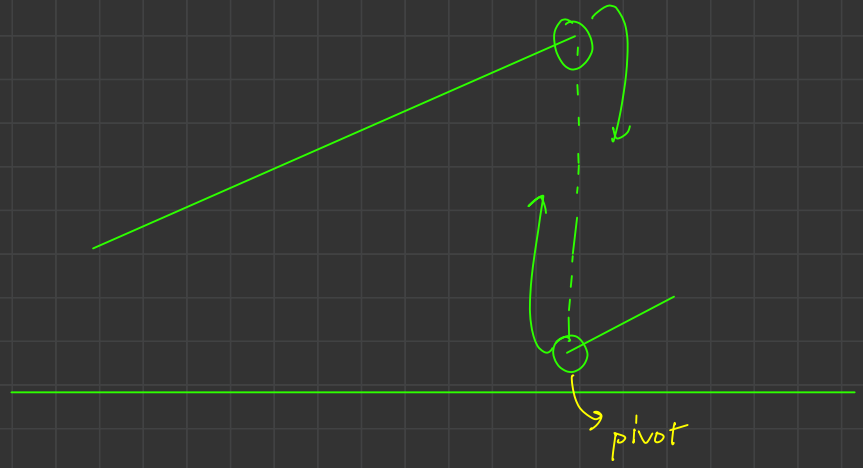
```
if (arr[mid] == key)
{
    ans = mid;
    ei = mid - 1;
}
```

find Minm in a rotated Sorted Array




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

else if (arr[si] <= arr[mid])
    si = mid+1;

else
    ei = mid-1;
```


0 1 2 3 4 5 6 7 8 9 10 11 12
4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1, 2, 3
si
mid
ei


pivot

Square Root

$N = 9$

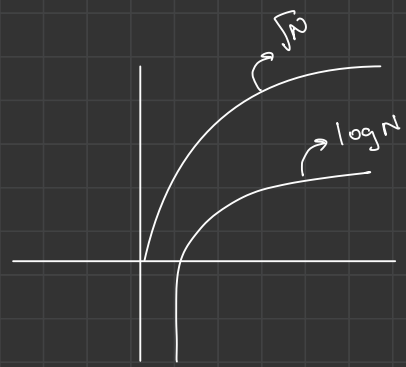
Brute force

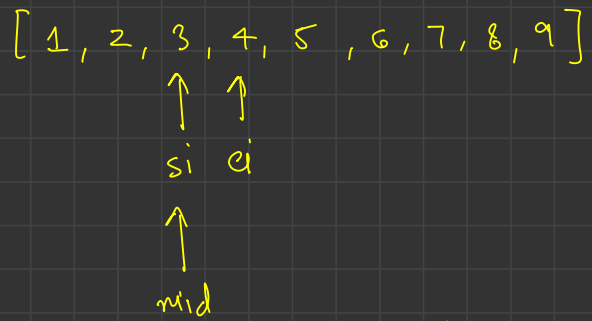
```
for(i = 1 → n)
{
    if (i * i == N)
        return i;
}
```

TC: $O(N)$

```
for(i = 0; i*i <= n; i++)
{
    if (i * i == N)
        return i;
}
```

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





TC: O(logN)

mid * mid == N
return mid

mid * mid > N
ei = mid - 1;

else
si = mid + 1;

$N = 10$