Binary Search (Algorithm)

```
                 0   1   2   3    4    5    6    7
int [] arr = {   1 , 3 , 7 , 10 , 11 , 14 , 20 , 40 }
```

Sorted

target = 14

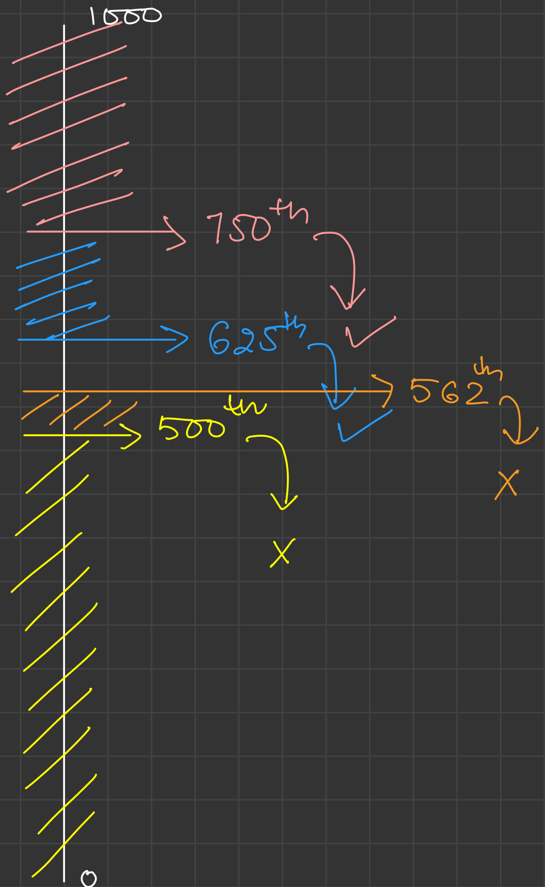
Brute force

linear Search

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

TC : O(N)

SC : O(1)



X 1000 Bricks → Brute force

with my smart approach

$\log_2(1000)$ → Bricks are req.

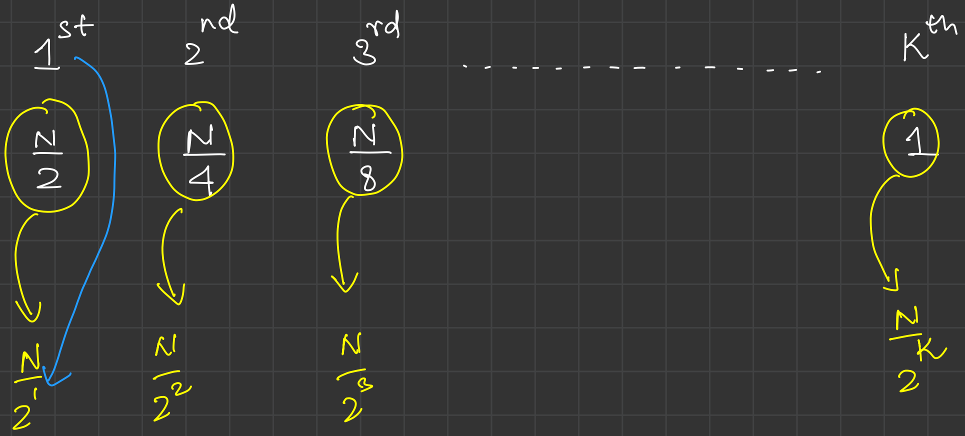
$\log_2 10^3 \Rightarrow 3\,\cancel{\log_2 10}^{\,3.1} \approx \boxed{10}$

→ using only 1 brick I eleminated 500 floors.

→ using 2nd brick I eliminated 250 floors.

→ using 3rd brick I eleminated 125 floors

→ using 4th brick I eleminated 62 floors



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

$$N = 2^K$$

takig $\log_2$ Both Side

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

$$\log_2 N = K\overset{1}{\log_2 2} \Rightarrow \boxed{K = \log_2 N}$$

required log N bricks

```
                0   1   2   3    4   5   6   7
int [] arr = {  1 , 3,  7 , 10 , 11, 14, 20, 40 }
                ↑           ↑                ↑
                si          mid              ei
```

target = 14

① define range

② get mid

③ try to eliminate half of range.

while

(si <= ei)

TC: O(logN)

SC: O(1)

while (si <= ei)

```
if (target == arr[mid])
{
    return mid;
} else if (arr[mid] < target)
{
    si = mid + 1;
} else
{
    ei = mid - 1;
}
```

{ Binary Search }

☆☆☆☆

Binary Search

It is always implemented over a sorted region.

→ Sorted Region ✓

→ TC : O(log N) (Expected)

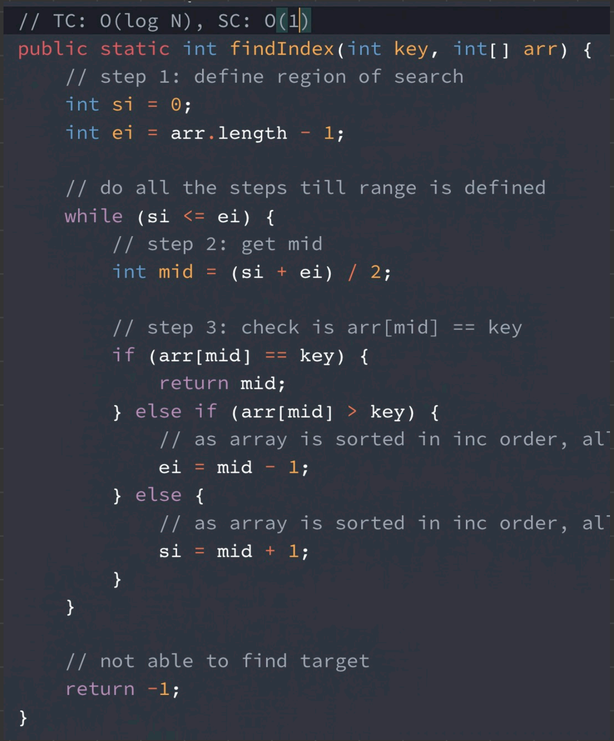
} 99%

↳ Binary Search

0 1 2 3 4 5 6

int[] arr = {1, 2, 10, 14, 20, 30, 100}

target = 90



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

    // do all the steps till range is defined
    while (si <= ei) {
        // step 2: get mid
        int mid = (si + ei) / 2;

        // step 3: check is arr[mid] == key
        if (arr[mid] == key) {
            return mid;
        } else if (arr[mid] > key) {
            // as array is sorted in inc order, al
            ei = mid - 1;
        } else {
            // as array is sorted in inc order, al
            si = mid + 1;
        }
    }

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

Binary Search

→ iterative TC: $O(\log_2 N)$ SC: $O(1)$ (Better)

→ recursive TC: $O(\log_2 N)$ SC: $O(\log_2 N)$ → callstack

(ceil) Search Insert Position

(find Pos of just greater)

```
           0   1   2   3    4    5    6
arr[] : {  1 , 3 , 7 , 10 , 11 , 20 , 40 }
        ↑  ↑
        ei si

key = -1
```

Brute force

TC: O(N)

```
{
    for(i → 0 → n)
        if(arr[i] > key)
            return i;
}
```

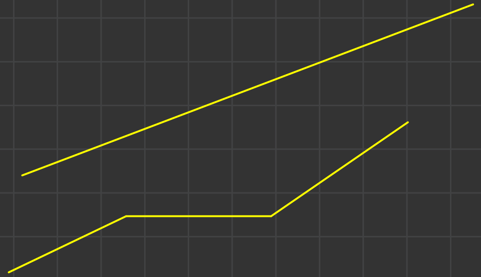
```
if (arr[mid] == key)
{
    return mid;
} else if (arr[mid] > key)
{
    pans = arr[mid]
    ei = mid - 1;
}
else
{
    si = mid + 1;
}
```

pans = ~~10~~
~~3~~
1

✓ find first and last Pos. of Ele

{ you are given inc

you are given non-dec }



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

↑ ei

✓ target = 2

↑ si

first Occ = ~~X~~ 1

↑ mid

```
if ( == )
{ pans = mid;
  ei = mid-1;
} else if
   ↓
```

Brute force

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

thought

↳ whenever I find key, store that place, and try find key again in left array, as we want leftmost occ. of key

Search In a 2D- Matrix

int[][] arr =
$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1, & 2, & 3, & 4 \\ 1 & 5, & 6, & 7, & \textcircled{8} \\ 2 & 9, & 10, & 11, & 12 \\ 3 & 13, & 14, & 15, & 16 \end{array}$$

(nxm)

target = 10

mxn → mxn → 1D Arr

(0 - 15) ⇝ indexing

range

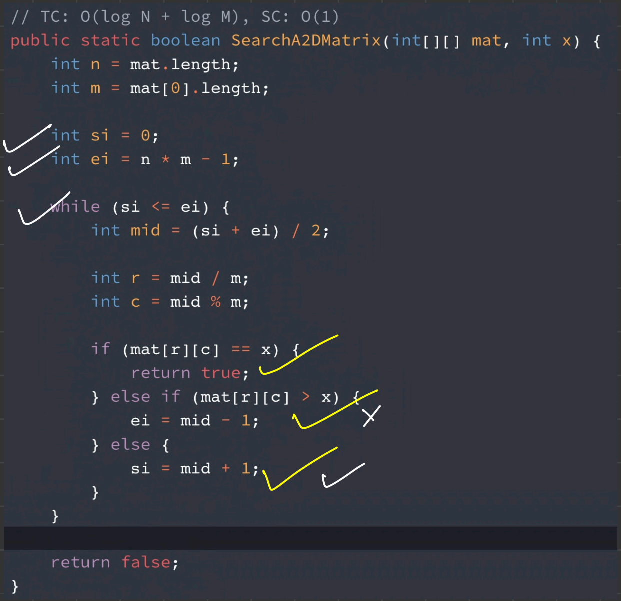
mid = 7

$$\left\{ \begin{aligned} r &= idx / m \\ c &= idx \% m \end{aligned} \right\}$$

✓

TC: log(NxM) = log N + log M

$$\begin{array}{cccccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\ 1, & 2, & 3, & 4, & 5 & ,6, & 7, & 8, & 9, & 10, & 11, & 12 \end{array}$$



```java
// TC: O(log N + log M), SC: O(1)
public static boolean SearchA2DMatrix(int[][] mat, int x) {
    int n = mat.length;
    int m = mat[0].length;

    int si = 0;
    int ei = n * m - 1;

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

        int r = mid / m;
        int c = mid % m;

        if (mat[r][c] == x) {
            return true;
        } else if (mat[r][c] > x) {
            ei = mid - 1;
        } else {
            si = mid + 1;
        }
    }

    return false;
}
```

int[][] arr =

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1, & 2, & 3, & 4 \\ 1 & 5, & 6, & 7, & 8 \\ 2 & 9, & 10, & 11, & 12 \end{array}$$

(n, m)

target = 10

$n = 3$

$m = 4$

$si = 9$, $ei = 9$

$mid = 9$

$r = 9/4 = 2$

$c = 9\%4 = 1$

int[][] arr =

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

target = 10

(n, m)