Data Structures and Algorithms Course

Searching Algorithms

Today’s Content

- Searching Algorithms
- Algorithm Questions

Searching Algorithms - Agenda

- Linear (Sequential) Search
- Binary Search
- Ternary Search
- Jump Search
- Exponential Search
- Algorithm questions with search

Linear Search (Sequential)

- Unordered or order is not known
- Scan complete array and see if element is there.

For example: I’m searching value “3”

	Best	**Worst**
Time Complexity	O(1)	O(n)

Linear Search (Sequential)

```
public int linearSearch (int[] Array, int data){
    for (int i=0;i<Array.length;i++) if(Array[i]==data) return i;
        return -1;
}
```

Lets switch to IntelliJ for implementation of Linear Search

Binary Search

Binary Search *is a searching algorithm used in* *<u>a sorted array</u>* *by* ***repeatedly dividing the search interval in half****.*
The idea of binary search is to use the information that the array is sorted and reduce the time complexity to O(Log n).

Algorithm:

- Calculate middle index.
- Compare x with the middle element.
- If x matches with the middle element, we return the mid index. (==)
- Else If x is greater than the mid element, then we recur for the right half. (>)
- Else (x is smaller) recur for the left half. (<)

- Caution: Works on sorted lists!

Binary Search

3	5	6	9	11	18	20	25	30	45
0	1	2	3	4	5	6	7	8	9

Assume we are searching for “6”

1 Calculate middle index

middle=(left+right)/2
= (0+9)/2 = 4

3	5	6	9	11	18	20	25	30	45
0	1	2	3	4	5	6	7	8	9

2 Compare value searched with the middle value

if (data < array[middle]) branch into left partition

3	5	6	9	11	18	20	25	30	45
0	1	2	3	4	5	6	7	8	9

3 Calculate new middle index

middle=(left+right)/2
= (0+3)/2 = 1

3	5	6	9
0	1	2	3

4 Compare value searched with the middle value

if (data > array[middle]) branch into right partition

3	5	6	9
0	1	2	3

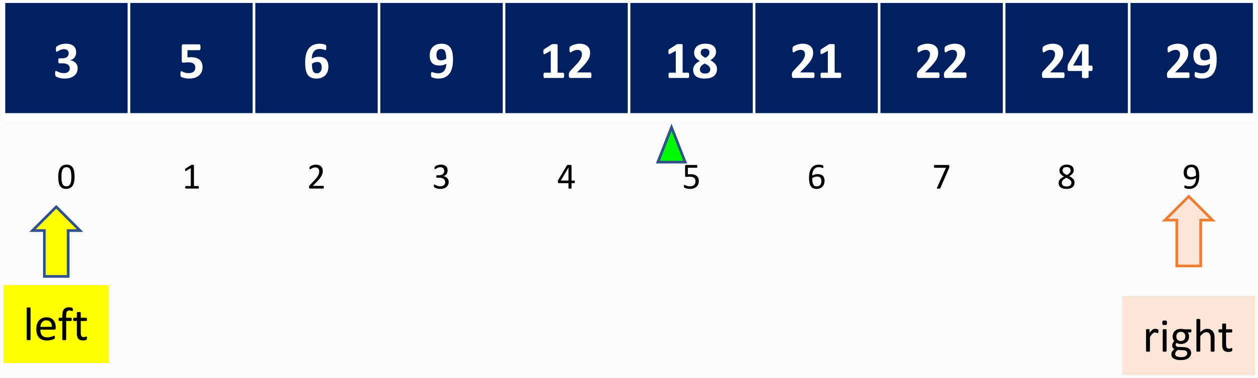
5 Calculate new middle index

middle=(left+right)/2
= (2+3)/2 = 2

6	9
2	3

6 if (data == array[middle]) return middle index; // return 2 – index of ‘6’

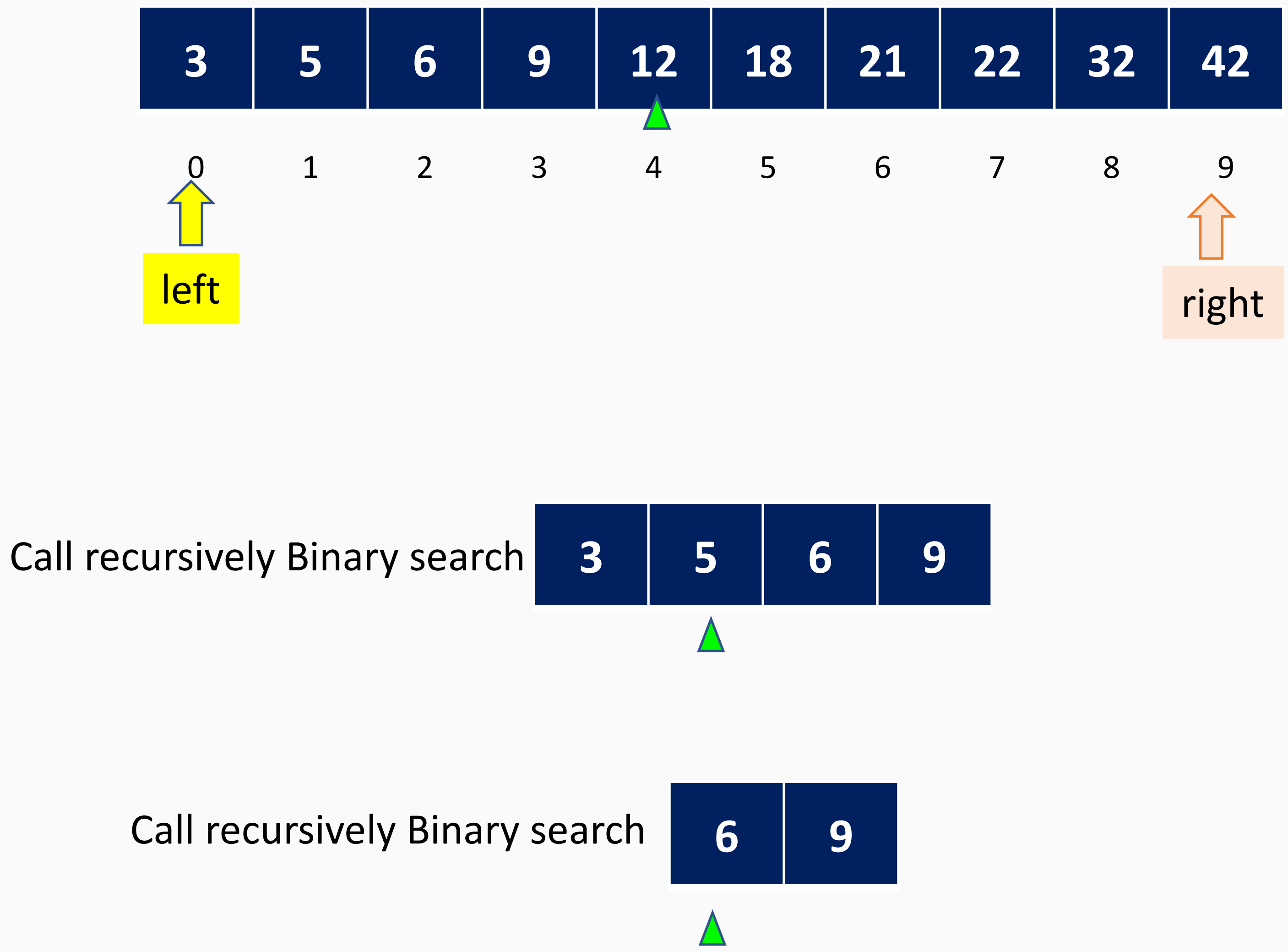
Binary Search (Iterative)



while (left <= right)

- **if (array[middle] == data) return middle;**
- **if (data < array[middle])   right = middle - 1;**
  else left = middle + 1;

Binary Search (Recursive)



Binary Search - Performance

Time Complexity : O (log n)

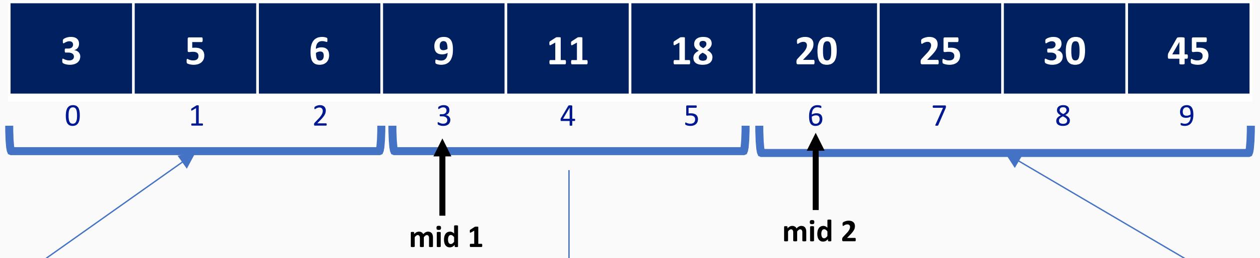
Recursive Space Complexity : O (log n)

Iterative Space Complexity : O (1)

Lets switch to IntelliJ for implementation of Binary Search

Ternary Search

- Similar to Binary search, instead of two parts Ternary search divides into 3 parts.



partitionSize = (right - left) / 3

mid1 = left + partitionSize

mid2 = right - partitionSize

if (data < array[mid1]) → **ternarySearch(left, mid1 - 1)**

ternarySearch(mid1+1, mid2-1)

if (data > array[mid2]) → **ternarySearch(mid2+1, right)**

Ternary Search-Performance

- Similar to Binary search, instead of two parts Ternary search divides into 3 parts.

Performance Comparison

BINARY SEARCH	TERNARY SEARCH
$\log_2 n$	$\log_3 n$
	Does not mean Ternary is faster. Makes to much comparison.

Ternary Search-Implementation

- Similar to Binary Search Recursive implementation.

1. Compare the key with the element at mid1. If equal, return mid1.
2. If not, compare the key with mid2. If equal, return mid2.
3. If not, then if the key is less than the element at mid1. Branch into the first part.
4. If not, then if the key is greater than the element at mid2. Branch into the third part.
5. If not, branch into the second (middle) part.

Lets switch to IntelliJ for implementation of Ternary Search

Searching Algorithms – Jump Search

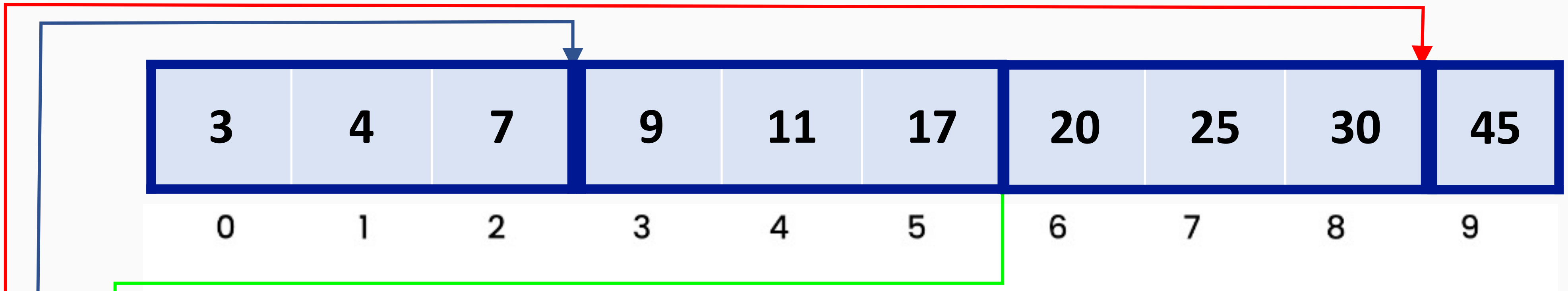
- Improvement to linear search but not as fast as Binary Search.
- For sorted arrays.
- The basic idea is to check fewer elements (than linear search) by jumping ahead by fixed steps or skipping some elements in place of searching all elements.

Block 1			Block 2			Block 3			Block 4
3	**4**	**7**	**9**	**11**	**17**	**20**	**25**	**30**	**45**
0	1	2	3	4	5	6	7	8	9

$$\text{blockSize} = \sqrt{n}$$
$$= 3$$

Searching Algorithms – Jump Search

- Improvement to linear search but not as fast as Binary Search.



$$\textbf{blockSize} = \sqrt{n}$$
$$= 3$$

- Assume we are searching 23.
- Compare 23 with the last item of block 1 (6). So it can’t be in this block.
- Compare 23 with the last item of block 2 (17). So it can’t be in this block.
- Compare 23 with the last item of block 3 (24). Its not in this block. So return -1.

Jump Search-Performance

Time Complexity : O($\sqrt{n}$)

Space Complexity : O(1)

Jump Search-Implementation

- **Calculate Block size :** blockSize = sqrt(array.length);
- **Set start point:** 0 for the first jump then +=blockSize
- **Jump Blocks:** while (start < array.length && array[nextBlockLast - 1] < data)

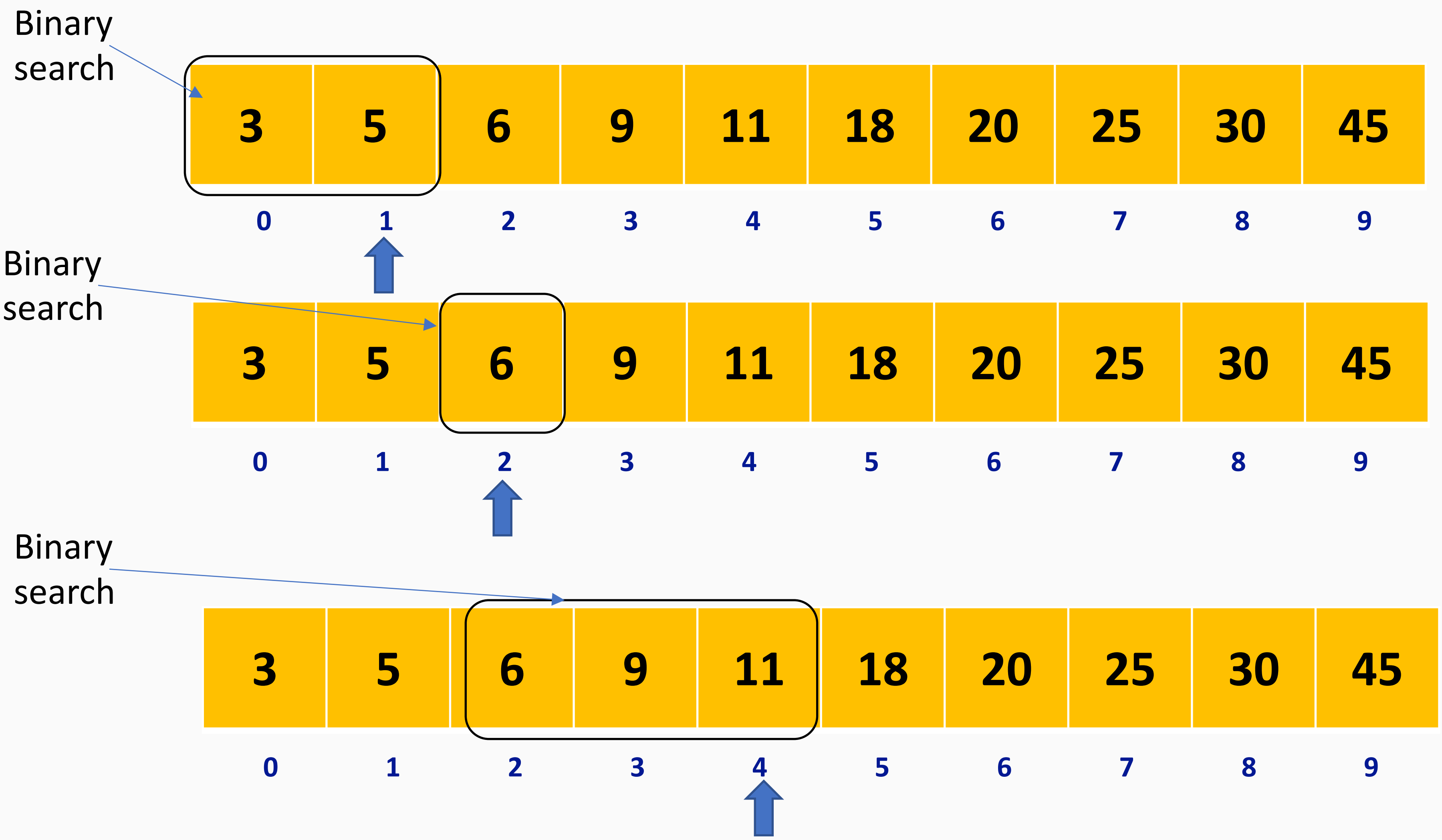
  start=next;

  next=next+blockSize;
- **If Block found :** Perform a **linear search** in that block.

Lets switch to IntelliJ for implementation of Jump Search

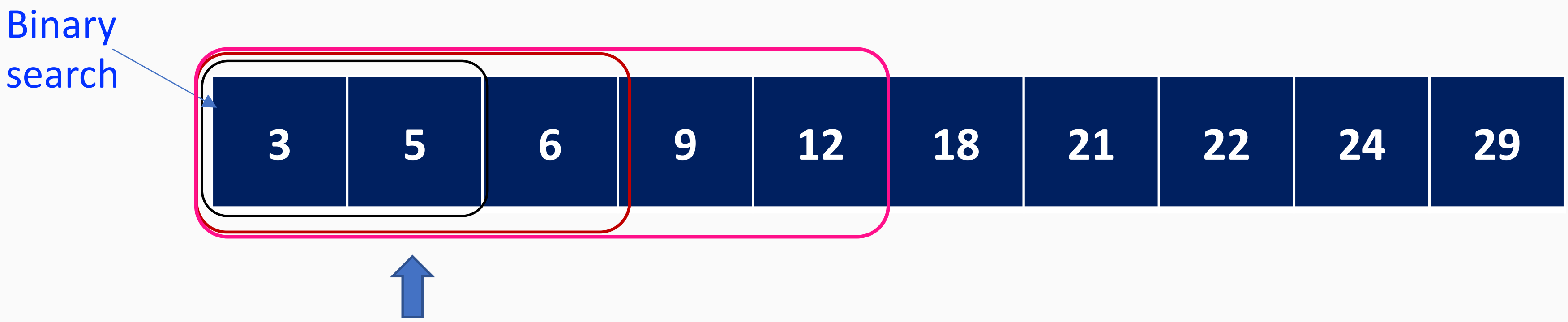
Exponential Search

- Start with a small range, if item does not exist double the search size.



Exponential Search-Performance

Exponential Search



Time Complexity : O(log n)

Exponential Search-Implementation

3	5	6	9	12	18	21	22	24	29

Start here → Bound=1

- **while (bound < array.length && array[bound] < data) bound *= 2;**
- **int left = bound / 2;**
- **int right = Math.min(bound, array.length - 1);**
- **return binarySearchRec(array, data, left, right)** ← Call binary search here

Lets switch to IntelliJ for implementation of Exponential Search

So what?

Why did we learn search algorithms, do we really need them?

Algorithm Question-1

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

You must write an algorithm with O(log n) runtime complexity.

Example 1:

Input: nums = [-1,0,3,5,9,12], target = 9

Output: 4

Explanation: 9 exists in nums and its index is 4

Example 2:

Input: nums = [-1,0,3,5,9,12], target = 2

Output: -1

Explanation: 2 does not exist in nums so return -1

Algorithm Question- Template

```
class Solution {
        public int search(int[] nums, int target) {

        }
}
```

Algorithm Question-2

You are given an m x n integer matrix matrix with the following two properties:

•Each row is sorted in non-decreasing order.

•The first integer of each row is greater than the last integer of the previous row.

Given an integer target, return true *if* target *is in* matrix *or* false *otherwise*.

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13
Output: false

Algorithm Question-2 Search a 2D Matrix

Lets switch to IntelliJ for Solution

Algorithm Question-2

You are given an m x n integer matrix matrix with the following two properties:

- Each row is sorted in non-decreasing order.
- The first integer of each row is greater than the last integer of the previous row.
- Given an integer target, return true *if* target *is in* matrix *or* false *otherwise*.
- You must write a solution in O(log(m * n)) time complexity.

1	3	5	7
10	11	16	20
23	30	34	60

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3
Output: true

Input: matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13
Output: false

How find row/col index?

- [[1, 3, 5, 7] ,[10, 11, 16, 20], [23, 30, 34, 60]]

	Col 0	Col 1	Col 2	Col 3
Row 0				
Row 1				
Row 2				