

■ Binary Search

The most important searching algorithm you'll ever learn

● Binary Search

■ EASY

■ What You Will Learn

- Why binary search is **100x faster** than linear search on large inputs
- The two-pointer trick: **left** and **right**
- How to calculate **mid** without overflow
- When to move left, when to move right
- How to recognize binary search problems in the wild

■ Intuition First

Imagine you're playing a number guessing game. I pick a number from 1 to 100. You guess, and I say 'higher' or 'lower'. A smart player never guesses 1, 2, 3... They guess **50 first**. If I say 'higher', they guess **75**. This is exactly binary search — cut the problem in half every time.

Real-world twin: Opening a dictionary. You don't read from page 1. You open the middle, see if you're too far left or right, and jump accordingly.

■ Intuition Box

Binary search only works on a **SORTED** array. If the array is not sorted, binary search will give wrong answers silently. Always verify sort order before applying it.

■ Visual Explanation

Picture the array as a row of numbered boxes. We track two pointers: left (L) and right (R). We always look at the middle element.

```

Index:   0    1    2    3    4    5    6
Array: [ 2 | 4 | 6 | 8 | 10 | 12 | 14 ]
          L                      R

          mid=3 (value=8)

Target = 10
8 < 10 → move L to mid+1

Array: [ 2 | 4 | 6 | 8 | 10 | 12 | 14 ]
                L          R

          mid=5 (value=12)
12 > 10 → move R to mid-1

Array: [ 2 | 4 | 6 | 8 | 10 | 12 | 14 ]
                L          R

          mid=4 (value=10)
10 == 10 → FOUND at index 4! ✓

```

■ ■ How It Works — Step by Step

1. Set **left = 0** and **right = array.length - 1**
2. While **left <= right**:
 - a. Calculate **mid = left + (right - left) / 2** ← safe from overflow
 - b. If **arr[mid] == target** → return mid (found!)
 - c. If **arr[mid] < target** → move left = mid + 1 (go right)
 - d. If **arr[mid] > target** → move right = mid - 1 (go left)
3. If loop ends → return -1 (not found)

■ ■ Common Mistake

Never write `mid = (left + right) / 2`. For very large arrays, `left + right` can overflow an int (exceed 2 billion). Always use: `mid = left + (right - left) / 2`.

■ Dry Run — Trace Through Example

Array: [1, 3, 5, 7, 9, 11, 13] | Target: 7

```
Step 1: L=0, R=6 → mid=3 → arr[3]=7 == target → FOUND at index 3
```

(One lucky shot! Let's try target=11 for a longer trace)

```
Step 1: L=0, R=6 → mid=3 → arr[3]=7 < 11 → L=4
```

```
Step 2: L=4, R=6 → mid=5 → arr[5]=11 == 11 → FOUND at index 5
```

■ Java Code

```
1 class Solution {
```

```

2   public int search(int[] nums, int target) {
3       int left = 0, right = nums.length - 1;
4
5       while (left <= right) {
6           // Safe mid calculation – avoids integer overflow
7           int mid = left + (right - left) / 2;
8
9           if (nums[mid] == target) {
10              return mid; // Found it!
11          } else if (nums[mid] < target) {
12              left = mid + 1; // Target is in the right half
13          } else {
14              right = mid - 1; // Target is in the left half
15          }
16      }
17      return -1; // Not found
18  }
19  }

```

■ Time & Space Complexity

Complexity	Value	What it means
Time	$O(\log n)$	If input doubles ($n \rightarrow 2n$), we only do 1 extra step. 1 million elements = ~20 steps!
Space	$O(1)$	We only use 3 variables (left, right, mid). No extra array needed.

■ Common Mistakes

■ Mistake 1

Using `left < right` instead of `left <= right`. This misses the case where the answer is the single remaining element.

■ Mistake 2

Setting `left = mid` instead of `left = mid + 1` when `arr[mid] < target`. This causes an infinite loop when `left == right`.

■ Pro Tips

■ Pro Tip 1

The pattern '`left <= right` with `left=mid+1` and `right=mid-1`' is the safest template. Memorize it first.

■ Pro Tip 2

Binary search works on any monotonic function, not just arrays. If something increases/decreases predictably, you can binary search on it!

■ Practice Problems

#	Problem	Difficulty
1	Binary Search (LeetCode 704)	Easy
2	Guess Number Higher or Lower (LeetCode 374)	Easy
3	Search Insert Position (LeetCode 35)	Easy
4	Count Negative Numbers in Sorted Matrix (LeetCode 1351)	Medium
5	Find Peak Element (LeetCode 162)	Medium

■ Quick Revision

✓	Only works on SORTED arrays
✓	left=0, right=n-1, loop while left<=right
✓	mid = left + (right-left)/2 (overflow-safe)
✓	arr[mid] < target → left=mid+1 arr[mid] > target → right=mid-1
✓	Time: O(log n), Space: O(1)
✓	Return -1 if not found

■ Visual Thinking

Imagine the array as a hallway of 1000 doors. Linear search walks every door. Binary search teleports to door 500, then 750, then 625... arriving in just 10 jumps. The 'window' (L to R) physically shrinks in half each step — like zooming into the right portion of a number line.

■ Think Like This

Animation idea: Show a shrinking highlighted bar over the array. Each step, half the bar disappears (the eliminated half). Remaining bar pulses green when target is found.