Modified Binary Find Pattern

Binary Find is a classic algorithm used to search for a value in a **sorted array** in O(log n) time by repeatedly dividing the search range in half.

Modified Binary Find means:

- The classic binary search is **tweaked to fit special problems** (not just “find exact value”), like:
    - Finding the **smallest number greater than or equal to a target** (the “ceiling”)
    - Finding the **first or last occurrence** of a value (when duplicates exist)
    - Finding **rotation points** in rotated arrays, or searching in “bitonic” (up-and-down) arrays

Modified Binary Search Pattern:

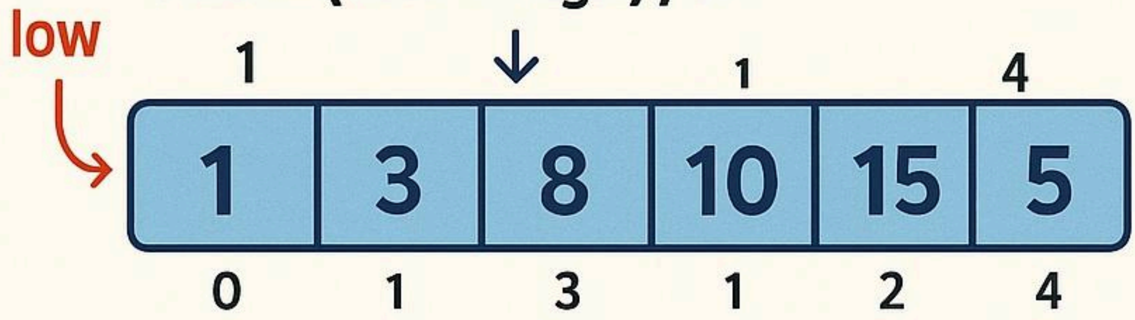
Find the Ceiling in a Sorted Array

Input Array: [1, 3, 8, 10, 15] **Target:** 9

Task: Find the smallest number in the array that greater than or equal to the target (the ceiling).

1 Start: Set low and high pointers at targt.

mid = (low + high) / 2

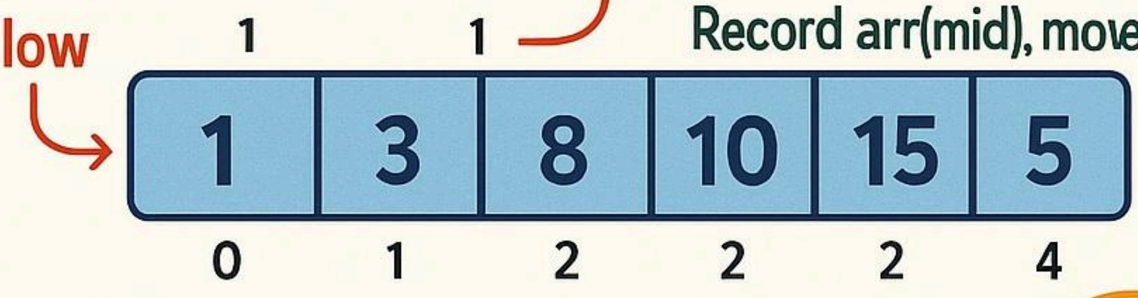


2 Move pointers based on comparison

mid = mid + 1 → 2

arr(mid)> target

Record arr(mid), move high



4 Repeat until pointers cross

5 Repeat until pointers cross

Ceiling = 10

Bonus: Modify binary search logic to fit special problems: first/last occurrence, ceiling/floor, rotation point, and more!

Modified binary search allows fast solutions for many sorted array

Classic Problem: Find the Ceiling of a Number

Problem Statement:
Given a **sorted array** and a target number, **find the smallest number in the array that is greater than or equal to the target** (the "ceiling"). If no such number exists, return -1.

Example:
Array: `[1, 3, 8, 10, 15]`, Target: `9`
Output: `10` (smallest number >= 9)

Explanation

1. **Set low and high pointers** (`low = 0`, `high = n-1`)
2. **While** `low <= high`:
   - Calculate mid: `mid = low + (high - low) / 2`
   - If `arr[mid] == target`, return `arr[mid]`
   - If `arr[mid] < target`, search the right half (`low = mid + 1`)
   - If `arr[mid] > target`, store mid as a possible answer, search left half (`high = mid - 1`)
3. At the end, if `low` is within bounds, `arr[low]` is the ceiling.
   Otherwise, return -1.

C Code Example: Ceiling in a Sorted Array

```
#include <stdio.h>

int findCeiling(int arr[], int n, int target) {
    int low = 0, high = n - 1;
    int result = -1; // Store answer

    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (arr[mid] == target)
            return arr[mid]; // Exact match
        else if (arr[mid] < target)
            low = mid + 1;   // Go right
        else {
            result = arr[mid]; // Possible answer
            high = mid - 1;    // Go left
        }
    }
    return result;
}

int main() {
    int arr[] = {1, 3, 8, 10, 15};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 9;
    int ceiling = findCeiling(arr, n, target);
    if (ceiling != -1)
        printf("Ceiling of %d is %d\n", target, ceiling);
    else
```

```
        printf("No ceiling found for %d\n", target);
    return 0;
}
```

How It Works

- Start with the full range.
- At each step, check the middle.
  - If exact, return.
  - If less than target, discard the left half.
  - If greater, record as possible answer and discard the right half.
- At the end, if you found a possible answer, that's the ceiling.

Other Modified Binary Find Problems

1. **First/Last Occurrence in a Sorted Array (with duplicates)**
   - Adjust search to keep going even after finding target (to find leftmost/rightmost).
2. **Bitonic Array Maximum**
   - An array that first increases, then decreases. Use modified binary search to find the peak.
3. **Find in Rotated Sorted Array**
   - Find the index of a value, even though the array is rotated (use properties of the sorted halves).

Practice Challenge

- Try to write a C function to **find the first and last occurrence of a value** in a sorted array with duplicates.
- Try to find the **peak element** in a “mountain” or “bitonic” array.

Key Takeaways

- Use **binary search**, but **tweak the logic** to fit special requirements.
- Always works only on **sorted arrays** (or with extra logic for rotated/bitonic arrays).