Day-1

1. Binary Search

Description:

Given a **sorted array** and a **target element**, return its index if found, else return `-1`.

Example:

```
Input: arr = [1, 3, 5, 7, 9], target = 5
Output: 2
```

Java Code:

```
public class ClassicBinarySearch {
    public int binarySearch(int[] arr, int target) {
        int low = 0, high = arr.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (arr[mid] == target) return mid;
            else if (arr[mid] < target) low = mid + 1;
            else high = mid - 1;
        }

        return -1; // not found
    }
}
```

Time Complexity:

- Time: O(log n)
- Space: O(1)

2. Count Occurrences of a Number in a Sorted Array

Description: Given a **sorted array**, find **how many times a target number occurs**.

Example:

```
Input: arr = [2, 4, 4, 4, 6], target = 4
Output: 3
```

Java Code:

```
public class CountOccurrences {
    public int countOccurrences(int[] arr, int target) {
        int first = findFirst(arr, target);
        if (first == -1) return 0; // not found
        int last = findLast(arr, target);
        return last - first + 1;
    }

    private int findFirst(int[] arr, int target) {
        int low = 0, high = arr.length - 1, result = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) {
                result = mid;
                high = mid - 1; // go left
            } else if (arr[mid] < target) low = mid + 1;
            else high = mid - 1;
        }
        return result;
    }

    private int findLast(int[] arr, int target) {
        int low = 0, high = arr.length - 1, result = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (arr[mid] == target) {
                result = mid;
                low = mid + 1; // go right
            } else if (arr[mid] < target) low = mid + 1;
            else high = mid - 1;
        }
        return result;
    }
}
```

Time Complexity:

- Time: O(log n)
- Space: O(1)

Day-2

1. Floor of a Number in Sorted Array

Description:

Given a **sorted array** and a number `x`, find the **floor** of `x` — the **greatest element ≤ x**.

Example:

```
Input: arr = [1, 2, 4, 6, 10], x = 5
Output: 4
```

Java Code:

```
public class FloorOfNumber {
    public int findFloor(int[] arr, int x) {
        int low = 0, high = arr.length - 1;
        int floor = -1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (arr[mid] == x) return arr[mid];
            else if (arr[mid] < x) {
                floor = arr[mid];
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return floor;
    }
}
```

Time Complexity:

- Time: O(log n)
- Space: O(1)

2. Ceiling of a Number in Sorted Array

Description:

Given a **sorted array** and a value `x`, find the **ceiling** — the **smallest element greater than or equal to `x`**.

Example:

```
Input: arr = [1, 2, 4, 6, 10], x = 5
Output: 6

Input: arr = [1, 2, 8, 10, 10, 12, 19], x = 3
Output: 8
```

Java Code:

```
public class CeilingOfNumber {
    public int findCeiling(int[] arr, int x) {
        int low = 0, high = arr.length - 1;
        int ceiling = -1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (arr[mid] == x) return arr[mid];
            else if (arr[mid] < x) {
                low = mid + 1;
            } else {
                ceiling = arr[mid];
                high = mid - 1;
            }
        }

        return ceiling;
    }
}
```

Time Complexity:

- Time: O(log n)
- Space: O(1)

Day-3

1. Find First Bad Version (Leetcode 278)

Description:

You are given `n` versions `[1, 2, ..., n]` and a function `isBadVersion(version)` which tells whether the version is bad.
Find the **first bad version**.

Example:

```
Input: n = 5, firstBad = 4
Output: 4
```

Java Code:

```
public class FirstBadVersion extends VersionControl {
    public int firstBadVersion(int n) {
        int low = 1, high = n;
        int ans = -1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (isBadVersion(mid)) {
                ans = mid;
                high = mid - 1; // go left
            } else {
                low = mid + 1; // go right
            }
        }

        return ans;
    }
}
```

Assumes `isBadVersion(int version)` is provided in the `VersionControl` class.

Time Complexity:

- Time: O(log n)
- Space: O(1)

Day-4

1. Find Smallest Letter Greater Than Target (Leetcode 744)

Description:

Given a **sorted list of letters** (in circular order), return the **smallest character strictly greater than the target**. If no such character exists, wrap around to the first letter.

Example:

```
Input: letters = ['c', 'f', 'j'], target = 'd'
Output: 'f'

Input: letters = ['c', 'f', 'j'], target = 'k'
Output: 'c'  // wrap around
```

Java Code:

```
public class SmallestLetter {
    public char nextGreatestLetter(char[] letters, char target) {
        int low = 0, high = letters.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (letters[mid] <= target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }

        return letters[low % letters.length]; // wrap around
    }
}
```

Time Complexity:

- Time: O(log n)
- Space: O(1)

2. Find First and Last Position of Element in Sorted Array

Description:

Given an array of integers `nums` sorted in non-decreasing order, find the starting and ending position of a given target value.

- If the target is not found, return `[-1, -1]`.
- Your algorithm must run in **O(log n)** time.

Example:

```
Input: nums = [5,7,7,8,8,10], target = 8
Output: [3,4]

Input: nums = [5,7,7,8,8,10], target = 6
Output: [-1,-1]
```

Java Code:

```
public class FirstLastPosition {
    public int[] searchRange(int[] nums, int target) {
        int first = findIndex(nums, target, true);
        int last = findIndex(nums, target, false);
        return new int[]{first, last};
    }

    private int findIndex(int[] nums, int target, boolean findFirst) {
        int low = 0, high = nums.length - 1, result = -1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] == target) {
                result = mid;
                if (findFirst) {
                    high = mid - 1;  // Move left
                } else {
                    low = mid + 1;   // Move right
                }
            } else if (nums[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return result;
    }
}
```