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:

```java
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. 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:

```java
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;
    }
}
```

2. Search Insert Position

Description:

Given a **sorted array** of distinct integers and a target value, return the index if the target is found. If not, return the index where it **would be inserted** in order.

- Must run in **O(log n)** time.

Example:

```
Input: nums = [1,3,5,6], target = 5
Output: 2

Input: nums = [1,3,5,6], target = 2
Output: 1

Input: nums = [1,3,5,6], target = 7
Output: 4
```

Java Code:

```java
public class SearchInsertPosition {
    public int searchInsert(int[] nums, int target) {
        int low = 0, high = nums.length - 1;
        while (low <= high) {
            int mid = low + (high - low) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return low;  // Insertion position
    }
}
```

Day-3

1. Square Root of a Number

Description:

Implement `int sqrt(int x)` that returns the **integer part** of the square root of `x` (i.e., $\lfloor\sqrt{x}\rfloor$), **without using built-in sqrt functions**.

Example:

```
Input: x = 4
Output: 2

Input: x = 8
Output: 2  // since sqrt(8) ≈ 2.828, and we return floor value
```

Java Code:

```java
public class SquareRootBinarySearch {
    public int mySqrt(int x) {
        if (x == 0 || x == 1)
            return x;

        int low = 1, high = x / 2, ans = 0;
        while (low <= high) {
            int mid = low + (high - low) / 2;

            // To prevent overflow use long
            long square = (long) mid * mid;

            if (square == x)
                return mid;
            else if (square < x) {
                ans = mid;      // store the floor value
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
        return ans;
    }}
```

☐ **Complexity:**

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

2. Guess Number Higher or Lower

◆ Description:

You are given an integer `n`. You have to guess a number between `1` to `n`. You call a predefined API:

```
int guess(int num)
```

which returns:

- `-1` if your guess is higher than the number
- `1` if your guess is lower than the number
- `0` if your guess is correct

Implement a function to guess the correct number using **binary search**.

Example:

```
Input: n = 10, pick = 6
Output: 6
```

Java Code:

```
public class GuessNumberGame extends GuessGame {
    public int guessNumber(int n) {
        int low = 1, high = n;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            int res = guess(mid); // Assume guess(mid) is given

            if (res == 0)
                return mid;
            else if (res < 0)
                high = mid - 1;  // guess is too high
            else
                low = mid + 1;   // guess is too low
        }
        return -1; }}

// You assume this class is given as part of the question
class GuessGame {
    int pick = 6;  // Example hidden number

    int guess(int num) {
        if (num == pick) return 0;
        return num < pick ? 1 : -1;
    }}
```

Day-4

1. Kth Smallest Element in a Sorted Matrix

Description:

Given an `n x n` matrix where each row and column is sorted in ascending order, return the **kth smallest element** in the matrix.

Example:

```
Input:
matrix = [
  [1, 5, 9],
  [10, 11, 13],
  [12, 13, 15]
],
k = 8

Output: 13
```

Approach (Binary Search on Value Range):

We binary search on the **range of values**, not indices.

1. Set `low = matrix[0][0]`, `high = matrix[n-1][n-1]`
2. For each `mid`, count how many elements ≤ `mid`
3. Adjust the search range based on count vs `k`

Java Code:

```
public class KthSmallestInMatrix {
    public int kthSmallest(int[][] matrix, int k) {
        int n = matrix.length;
        int low = matrix[0][0];
        int high = matrix[n - 1][n - 1];

        while (low < high) {
            int mid = low + (high - low) / 2;
            int count = countLessEqual(matrix, mid);

            if (count < k)
                low = mid + 1;
            else
```

```
                high = mid;
        }

        return low;
    }

    // Helper to count elements ≤ target
    private int countLessEqual(int[][] matrix, int target) {
        int count = 0, n = matrix.length;
        int row = n - 1, col = 0;

        while (row >= 0 && col < n) {
            if (matrix[row][col] <= target) {
                count += row + 1;
                col++;
            } else {
                row--;
            }
        }

        return count;
    }
}
```

Complexity:

- Time: O(n * log(max - min))
- Space: O(1)