Sheryians Coding School

Live Cohort

Day 86

Binary Search-Based Problems

Topic 1: Peak Index in Mountain Array

Definition:

In a mountain array (elements first increase and then decrease), the peak index is the index of the maximum element. This is also called finding the maximum in a bitonic array.

Code(Examples):

```
var peakIndexInMountainArray = function(arr) {
    let start = 0, end = arr.length - 1;

    while (start < end) {
        let mid = Math.floor((start + end) / 2);

        if (arr[mid] < arr[mid + 1]) {
            start = mid + 1; // Move right
        } else {
            end = mid; // Stay or move left
        }
    }

    return start; // or end
};
```

Use Case:

Used in arrays where elements strictly increase and then strictly decrease — e.g., mountain landscapes, performance curves, or trend analysis.

Interview Q&A:

Q: What approach is used to find the peak in a mountain array?

Definition:

A binary search is used. If arr[mid] < arr[mid + 1], we move right; else we move left. The loop stops when start == end.

Topic 2: Search in Rotated Sorted Array

Definition:

Given a rotated sorted array, the goal is to find the index of a given target element in O(log n) time.
💻 Code (Example):

Code(Examples):

```javascript
var search = function(nums, target) {
    let start = 0, end = nums.length - 1;

    while (start <= end) {
        let mid = Math.floor((start + end) / 2);

        if (nums[mid] === target) return mid;

        if (nums[start] <= nums[mid]) { // Left is sorted
            if (nums[start] <= target && target < nums[mid]) {
                end = mid - 1;
            } else {
                start = mid + 1;
            }
        } else { // Right is sorted
            if (nums[mid] < target && target <= nums[end]) {
                start = mid + 1;
            } else {
                end = mid - 1;
            }
        }
    }

    return -1;
};
```

Use Case:

Searching in arrays where the order is shifted — common in circular buffers or log-rotated data structures.

Interview Q&A:

Q: How do you decide which half of the array is sorted?

Definition:

If nums[start] <= nums[mid], then the left half is sorted; otherwise, the right half is sorted.

Topic 3: Book Allocation Problem

Definition:

Distribute n books among k students such that the maximum number of pages assigned to any student is minimized. Allocation must be contiguous and every student must get at least one book.

Code(Examples):

```
function isPossible(books, students, maxPages) {
    let sum = 0, count = 1;

    for (let pages of books) {
        if (pages > maxPages) return false;

        if (sum + pages > maxPages) {
            count++;
            sum = pages;
        } else {
            sum += pages;
        }
    }

    return count <= students;
}

function allocateBooks(books, students) {
    if (books.length < students) return -1;

    let start = Math.max(...books);
    let end = books.reduce((a, b) => a + b, 0);
    let result = -1;
```

```
    while (start <= end) {
        let mid = Math.floor((start + end) / 2);

        if (isPossible(books, students, mid)) {
            result = mid;
            end = mid - 1;
        } else {
            start = mid + 1;
        }
    }

    return result;
}
```

Use Case:

Task assignment in resource-constrained environments: printers, team task load balancing, or storage allocation.

Interview Q&A:

Q: What's the range of search space for this problem?

Between the maximum single book (max(books)) and total pages (sum(books)).

Topic 4: Capacity to Ship Packages Within D Days

Definition:

Determine the minimum ship capacity required to transport all packages within D days.

Code(Examples):

```
function canShip(weights, days, capacity) {
    let total = 0, d = 1;

    for (let w of weights) {
        if (total + w > capacity) {
            d++;
            total = w;
        } else {
            total += w;
        }
    }

    return d <= days;
}

var shipWithinDays = function(weights, D) {
    let left = Math.max(...weights);
    let right = weights.reduce((a, b) => a + b, 0);
    let ans = right;

    while (left <= right) {
        let mid = Math.floor((left + right) / 2);
        if (canShip(weights, D, mid)) {
            ans = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return ans;
};
```

Use Case:

Logistics and shipment optimization where daily capacity constraints are present — warehouse to store, delivery route planning, etc.

Interview Q&A:

Q: What's the logic for validating a capacity?

Simulate shipping using the capacity and count days required. If days exceed D, increase capacity.

Topic 5: Koko Eating Bananas

Definition:

Koko can eat k bananas per hour. Given piles of bananas, find the minimum eating speed k so she finishes in h hours.

Code(Examples):

```
function canEatAll(piles, h, speed) {
    let time = 0;

    for (let pile of piles) {
        time += Math.ceil(pile / speed);
    }

    return time <= h;
}

var minEatingSpeed = function(piles, h) {
    let left = 1, right = Math.max(...piles);
    let ans = right;

    while (left <= right) {
        let mid = Math.floor((left + right) / 2);
```

```
        if (canEatAll(piles, h, mid)) {
            ans = mid;
            right = mid - 1;
        } else {
            left = mid + 1;
        }
    }

    return ans;
};
```

Use Case:

Rate optimization problems — servers processing requests per second, consumption rate problems in factories, or battery drain rates.

Interview Q&A:

Q: What is the binary search space in this problem?

A: From 1 to max(piles), since speed can't be more than the largest pile.