Day	New Problem	Yesterday's Revision	Flash Review (older problems	Pattern	Status	Key Trick

1	Two Sum, Contains Duplicate	-	-	Two Sum - HashMap/complement pattern Contains Duplicate -HashSet(store numbers, check repetition)/ frequency check	→ Two sum : check compliment in seen numbers, return pair when found, →Contains Duplicate “Keep a set of numbers, return True if a repeat appears.”

5	Encode & Decode Strings, Longest Consecutive Sequence	Maximum Subarray, Maximum Product Subarray	Valid Anagram	HashMap / Set	Map / Set tricks

6	Revision Day: Solve 3 old problems	-	-	-	-
7	Pattern Recap Week 1	-	-	-	Write all HashMap tricks

8	Valid Palindrome, Two Sum II	Encode & Decode Strings, Longest Consecutive Sequence	Maximum Subarray	Two Pointers	Expand/shrink pointers

9	3Sum, Container With Most Water	Valid Palindrome, Two Sum II	Top K Frequent Elements	Two Pointers	Sorting + two pointers

10	Best Time to Buy & Sell Stock, Longest Substring Without Repeating Characters	3Sum, Container With Most Water	Product of Array Except Self	Sliding Window	Expand/shrink window

11	Minimum Window Substring, Sliding Window Maximum	Best Time to Buy Stock, Longest Substring w/o Repeating	Maximum Product Subarray	Sliding Window	Track counts / max sum

12	Subarray Sum Equals K, Longest Repeating Character Replacement	Minimum Window Substring, Sliding Window Maximum	Encode & Decode Strings	Sliding Window	Count / frequency window

13	Revision Day: Solve 3 old problems	-	-	-	-
14	Pattern Recap Week 2	-	-	-	Write all Sliding Window tricks

15	Reverse Linked List, Merge Two Sorted Lists	Subarray Sum Equals K, Longest Repeating Char Replacement	Minimum Window Substring	Linked List	Pointer manipulation

16	Detect Cycle in Linked List, Remove Nth Node From End	Reverse Linked List, Merge Two Sorted Lists	Sliding Window Maximum	Linked List	Fast/slow pointer tricks

17	Reorder List, Valid Parentheses	Detect Cycle, Remove Nth Node	Subarray Sum Equals K	Stack	Stack push/pop logic

18	Min Stack, Evaluate Reverse Polish Notation	Reorder List, Valid Parentheses	Longest Repeating Char Replacement	Stack	Track min / postfix eval

19	Daily Temperatures, Largest Rectangle in Histogram	Min Stack, Evaluate RPN	Detect Cycle	Stack / Monotonic Stack	Maintain stack property

20	Revision Day: Solve 3 old problems	-	-	-	-
21	Pattern Recap Week 3	-	-	-	Write all Linked List & Stack tricks

22	Binary Search, Search Insert Position	Daily Temperatures, Largest Rectangle	Min Stack	Binary Search	Mid-point check, bounds

23	Search in Rotated Sorted Array, Find Minimum in Rotated Sorted Array	Binary Search, Search Insert Position	Evaluate RPN	Binary Search	Adjust bounds with rotation

24	Find First & Last Position, Median of Two Sorted Arrays	Search in Rotated, Find Minimum	Largest Rectangle	Binary Search	Edge cases with duplicates

25	Climbing Stairs, House Robber	First & Last Position, Median	Daily Temperatures	DP	Base case + recurrence

26	Coin Change, Longest Increasing Subsequence	Climbing Stairs, House Robber	Largest Rectangle	DP	Tabulation / memoization

Your Example	Step-by-Step Notes	Code	Fullstack & React/Node	System Design & Databases

1. Two Sum : Step-by-Step:

Seen numbers: {}

First number 1 → complement = 9 - 1 = 8 → 8 in seen?
→ add 1 → seen = {1}

Next number 4 → complement = 9 - 4 = 5 → 5 in seen?
→ add 4 → seen = {1, 4}

Next number 5 → complement = 9 - 5 = 4 → 4 in seen?
Found pair: 4 and 5
Answer: [4, 5]

2. ContainsDuplicate: Step-by-Step:

Seen numbers: {}

First number 2 → add → seen = {2}

Next number 3 → add → seen = {2, 3}

Question

Explain how the Token Bucket rate-limiting algorithm works and how you would implement it at a high level for an API.
Answer -
Think of it like a water bucket:

Token Bucket

A bucket holds a fixed number of tokens (e.g., 10 tokens).

Each token allows one API request.

Token Refill

Tokens are added at a fixed rate (e.g., 1 token per second).

Bucket never exceeds its max size.

Handling Requests

When a request comes:

If token is available → consume token → allow request

If no token → reject or delay request

When Tokens Run Out

1. Approach A — HashMap / frequency count (most general, O (n))

Plain-English steps

If lengths differ → return False.

Create an empty frequency map freq (key = char, value = count).

Loop over each character ch in s: add 1 to freq[ch].

Loop over each character ch in t:

If ch not in freq or freq[ch] == 0 → t has an extra char → return False.

Else subtract 1 from freq[ch].

If you finish the loop without returning False, then every char in t matched the counts from s → return True.

Trace with s=" anagram", t=" nagaram"

1. Approach - A

Problem 1: Top K Frequent Elements

Given nums = [1, 1, 1, 2, 2, 3] and k = 2, return [1, 2] (the two most frequent numbers).
Counter counts frequency of each number (works like a HashMap).

heapq gives us a heap (to store top k frequent numbers efficiently).
We’ll store pairs like (frequency, number)
Why this order? Because heapq in Python sorts by first value (frequency)
Now heap fills up gradually:

1️⃣Push (3,1) → heap = [(3,1)]
2️⃣Push (2,2) → heap = [(2,2), (3,1)]
3️⃣Push (1,3) → heap = [(1,3), (3,1), (2,2)]

heapq

Day 2 - Question - You are designing a MERN-based backend service that is getting slow as traffic increases. Explain how you would identify where the bottleneck is (frontend, backend, database, or network) and what tools or techniques you would use.
Answer ->
Approach:

Start from the frontend

Use Chrome DevTools (Network tab) to measure API call latency.

Check if delays come from multiple API calls, large payloads, or frontend rendering.

This helps confirm whether the issue is frontend or backend.

Analyze backend performance

Measure API response times using logs and APM tools.

Track request throughput, error rates, and slow

1️⃣ Encode & Decode Strings

```
def encode(strs):
    return ''.join(f"{len(s)}#{s}" for s in 
strs)

def decode(s):
    res, i = [], 0
    while i < len(s):
        j = i
        while s[j] != '#':
            j += 1
        length = int(s[i:j])
        res.append(s[j+1:j+1+length])
        i = j + 1 + length
    return res

# Example
strings = ["hello","world"]
encoded = encode(strings)
decoded = decode(encoded)
print(encoded)  # "5#hello5#world"
print(decoded)  # ["hello","world"]
```

2️⃣ Longest Consecutive Sequence

```
def longestConsecutive(nums):
    num_set = set(nums)
    longest = 0

    for num in nums:
        if num - 1 not in num_set:  # 
start of a sequence
            current = num
            length = 1
            while current + 1 in num_set:
                current += 1
                length += 1
            longest = max(longest, 
length)

    return longest

# Example
print(longestConsecutive
([100,4,200,1,3,2]))  # Output: 4
```

A bucket holds a fixed numb

1 Valid Palindrome

```
def isPalindrome(s):
    left, right = 0, len(s) - 1
    while left < right:
        while left < right and not s[left].isalnum():
            left += 1
        while left < right and not s[right].isalnum():
            right -= 1
        if s[left].lower() != s[right].lower():
            return False
        left += 1
        right -= 1
    return True

# Example
print(isPalindrome("A man, a plan, a canal: Panama"))  # Output: True
```

2 Two Sum II (Input Array Sorted)

```
def twoSum(numbers, target):
    left, right = 0, len(numbers) - 1
    while left < right:
        current_sum = numbers[left] + numbers[right]
        if current_sum == target:
            return [left + 1, right + 1]  # 1-indexed
        elif current_sum < target:
            left += 1
        else:
            right -= 1

# Example
print(twoSum([2,7,11,15], 9))  # Output: [1,2]
```

1️⃣ 3Sum

```
def threeSum(nums):
    nums.sort()
    res = []
    n = len(nums)

    for i in range(n):
        if i > 0 and nums[i] == nums[i-
1]:  # skip duplicates
            continue
        left, right = i + 1, n - 1
        while left < right:
            total = nums[i] + nums[left] +
nums[right]
            if total == 0:
                res.append([nums[i], nums
[left], nums[right]])
                left += 1
                right -= 1
                while left < right and nums
[left] == nums[left-1]:
                    left += 1
                while left < right and nums
[right] == nums[right+1]:
                    right -= 1
            elif total < 0:
                left += 1
            else:
                right -= 1
    return res

# Example
print(threeSum([-1,0,1,2,-1,-4]))  #
Output: [[-1,-1,2],[-1,0,1]]
```

2️⃣ Container With Most Water

```
def maxArea(height):
    left, right = 0, len(height) - 1
    max_area = 0

    while left < right:
        area = min(height[left], height
[right]) * (right - left)
```

Each token allows **one API re**

1️⃣ Best Time to Buy & Sell Stock

```
def maxProfit(prices):
    min_price = float('inf')
    max_profit = 0
    for price in prices:
        min_price = min(min_price,
price)
        max_profit = max(max_profit,
price - min_price)
    return max_profit

# Example
print(maxProfit([7,1,5,3,6,4]))  #
Output: 5
```

2️⃣ Longest Substring Without Repeating Characters

```
def lengthOfLongestSubstring(s):
    char_set = set()
    left = 0
    max_len = 0

    for right in range(len(s)):
        while s[right] in char_set:
            char_set.remove(s[left])
            left += 1
        char_set.add(s[right])
        max_len = max(max_len, right -
left + 1)

    return max_len

# Example
print(lengthOfLongestSubstring
("abcabcbb"))  # Output: 3
```

1 Minimum Window Substring

```
from collections import Counter

def minWindow(s, t):
    if not s or not t:
        return ""

    dict_t = Counter(t)
    required = len(dict_t)
    left, right = 0, 0
    formed = 0
    window_counts = {}
    min_len = float('inf')
    min_window = (0, 0)

    while right < len(s):
        char = s[right]
        window_counts[char] =
window_counts.get(char, 0) + 1

        if char in dict_t and
window_counts[char] == dict_t
[char]:
            formed += 1

        while left <= right and formed
== required:
            char = s[left]
            if right - left + 1 < min_len:
                min_len = right - left + 1
                min_window = (left, right)

            window_counts[char] -= 1
            if char in dict_t and
window_counts[char] < dict_t[char]:
                formed -= 1
            left += 1

        right += 1

    return "" if min_len == float('inf')
else s[min_window[0]:min_window
[1]+1]
```

2. Token Refill

1️⃣Subarray Sum Equals K

```
def subarraySum(nums, k):
    count_map = {0:1}
    curr_sum = 0
    total = 0

    for num in nums:
        curr_sum += num
        if curr_sum - k in count_map:
            total += count_map[curr_sum
- k]
        count_map[curr_sum] =
count_map.get(curr_sum, 0) + 1

    return total

# Example
print(subarraySum([1,1,1], 2))  #
Output: 2
```

2️⃣Longest Repeating Character Replacement

```
def characterReplacement(s, k):
    count = {}
    left = 0
    max_count = 0
    max_len = 0

    for right in range(len(s)):
        count[s[right]] = count.get(s
[right], 0) + 1
        max_count = max(max_count,
count[s[right]])

        if (right - left + 1) - max_count >
k:
            count[s[left]] -= 1
            left += 1

        max_len = max(max_len, right -
left + 1)

    return max_len

# Example
```

1️⃣ **Reverse Linked List**

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def reverseList(head):
    prev = None
    curr = head
    while curr:
        next_node = curr.next
        curr.next = prev
        prev = curr
        curr = next_node
    return prev
```

2️⃣ **Merge Two Sorted Lists**

```
def mergeTwoLists(l1, l2):
    dummy = ListNode(0)
    tail = dummy

    while l1 and l2:
        if l1.val < l2.val:
            tail.next = l1
            l1 = l1.next
        else:
            tail.next = l2
            l2 = l2.next
        tail = tail.next

    if l1:
        tail.next = l1
    if l2:
        tail.next = l2

    return dummy.next
```

Tokens are added at a fixed r

1️⃣Detect Cycle in Linked List

```
def hasCycle(head):
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next
        if slow == fast:
            return True
    return False
```

2️⃣Remove Nth Node From End

```
def removeNthFromEnd(head, n):
    dummy = ListNode(0)
    dummy.next = head
    first = second = dummy

    # Move first n+1 steps ahead
    for _ in range(n + 1):
        first = first.next

    # Move both pointers
    while first:
        first = first.next
        second = second.next

    # Remove nth node
    second.next = second.next.next

    return dummy.next
```

1 Reorder List

```
def reorderList(head):
    if not head or not head.next:
        return

    # Step 1: Find middle
    slow = fast = head
    while fast and fast.next:
        slow = slow.next
        fast = fast.next.next

    # Step 2: Reverse second half
    prev, curr = None, slow.next
    slow.next = None
    while curr:
        nxt = curr.next
        curr.next = prev
        prev = curr
        curr = nxt

    # Step 3: Merge two halves
    first, second = head, prev
    while second:
        tmp1, tmp2 = first.next, second.
next
        first.next = second
        second.next = tmp1
        first, second = tmp1, tmp2
```

Bucket never exceeds its ma

2 Valid Parentheses

```
def isValid(s):
    stack = []
    mapping = {')':'(', '}':'{', ']':'['}

    for char in s:
        if char in mapping:
            top = stack.pop() if stack else
'#'
            if mapping[char] != top:
                return False
        else:
            stack.append(char)
```

1️⃣Min Stack

```
class MinStack:
    def __init__(self):
        self.stack = []
        self.min_stack = []

    def push(self, x: int):
        self.stack.append(x)
        if not self.min_stack or x <= self.
min_stack[-1]:
            self.min_stack.append(x)

    def pop(self):
        if self.stack.pop() == self.
min_stack[-1]:
            self.min_stack.pop()

    def top(self):
        return self.stack[-1]

    def getMin(self):
        return self.min_stack[-1]

# Example
minStack = MinStack()
minStack.push(-2)
minStack.push(0)
minStack.push(-3)
print(minStack.getMin())  # Output:
-3
minStack.pop()
print(minStack.top())     # Output: 0
print(minStack.getMin())  # Output:
-2
```

2️⃣Evaluate Reverse Polish Notation (RPN)

```
def evalRPN(tokens):
    stack = []
    for token in tokens:
        if token in '+-*/':
            b = stack.pop()
            a = stack.pop()
```

1 Daily Temperatures

```
def dailyTemperatures(T):
    res = [0] * len(T)
    stack = []  # store indices

    for i, temp in enumerate(T):
        while stack and T[i] > T[stack[-1]]:
            idx = stack.pop()
            res[idx] = i - idx
        stack.append(i)

    return res

# Example
print(dailyTemperatures([73,74,75,71,69,72,76,73]))  # Output: [1,1,4,2,1,1,0,0]
```

2 Largest Rectangle in Histogram

```
def largestRectangleArea(heights):
    stack = []
    max_area = 0
    heights.append(0)  # sentinel

    for i, h in enumerate(heights):
        while stack and h < heights[stack[-1]]:
            height = heights[stack.pop()]
            width = i if not stack else i - stack[-1] - 1
            max_area = max(max_area, height * width)
        stack.append(i)

    return max_area

# Example
print(largestRectangleArea([2,1,5,6,2,3]))  # Output: 10
```

3. Handling Requests

1️⃣ Binary Search

```
def binarySearch(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = left + (right - left) // 2
        if nums[mid] == target:
            return mid
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1

# Example
print(binarySearch([-1,0,3,5,9,12],
9))  # Output: 4
```

2️⃣ Search Insert Position

```
def searchInsert(nums, target):
    left, right = 0, len(nums)

    while left < right:
        mid = left + (right - left) // 2
        if nums[mid] < target:
            left = mid + 1
        else:
            right = mid

    return left

# Examples
print(searchInsert([1,3,5,6], 5))  #
Output: 2
print(searchInsert([1,3,5,6], 2))  #
Output: 1
```

1️⃣Search in Rotated Sorted Array

```
def search(nums, target):
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = left + (right - left) // 2
        if nums[mid] == target:
            return mid

        # Left half sorted
        if nums[left] <= nums[mid]:
            if nums[left] <= target < nums
[mid]:
                right = mid - 1
            else:
                left = mid + 1
        # Right half sorted
        else:
            if nums[mid] < target <=
nums[right]:
                left = mid + 1
            else:
                right = mid - 1

    return -1

# Example
print(search([4,5,6,7,0,1,2], 0))  #
Output: 4
```

2️⃣Find Minimum in Rotated Sorted Array

```
def findMin(nums):
    left, right = 0, len(nums) - 1

    while left < right:
        mid = left + (right - left) // 2
        if nums[mid] > nums[right]:
            left = mid + 1
        else:
            right = mid

    return nums[left]
```

When a request comes:

1️⃣Find First & Last Position of Element in Sorted Array

```
def searchRange(nums, target):
    def findBound(isFirst):
        left, right = 0, len(nums) - 1
        bound = -1
        while left <= right:
            mid = left + (right - left) // 2
            if nums[mid] == target:
                bound = mid
                if isFirst:
                    right = mid - 1
                else:
                    left = mid + 1
            elif nums[mid] < target:
                left = mid + 1
            else:
                right = mid - 1
        return bound

    return [findBound(True),
findBound(False)]

# Example
print(searchRange([5,7,7,8,8,10], 8))
# Output: [3,4]
```

2️⃣Median of Two Sorted Arrays

```
def findMedianSortedArrays(nums1,
nums2):
    if len(nums1) > len(nums2):
        nums1, nums2 = nums2, nums1
    x, y = len(nums1), len(nums2)
    low, high = 0, x

    while low <= high:
        partitionX = (low + high) // 2
        partitionY = (x + y + 1) // 2 -
partitionX

        maxLeftX = float('-inf') if
partitionX == 0 else nums1
[partitionX-1]
        minRightX = float('inf') if
```

1️⃣ Climbing Stairs

```
def climbStairs(n):
    if n == 1:
        return 1
    a, b = 1, 2
    for _ in range(3, n+1):
        a, b = b, a + b
    return b

# Example
print(climbStairs(5))  # Output: 8
```

2️⃣ House Robber

```
def rob(nums):
    if not nums:
        return 0
    if len(nums) == 1:
        return nums[0]

    prev2, prev1 = nums[0], max
(nums[0], nums[1])
    for i in range(2, len(nums)):
        prev2, prev1 = prev1, max
(prev1, prev2 + nums[i])

    return prev1

# Example
print(rob([2,7,9,3,1]))  # Output: 12
```

1️⃣ Coin Change

```
def coinChange(coins, amount):
    dp = [float('inf')] * (amount + 1)
    dp[0] = 0

    for i in range(1, amount + 1):
        for coin in coins:
            if i >= coin:
                dp[i] = min(dp[i], dp[i - coin] + 1)

    return dp[amount] if dp[amount] != float('inf') else -1

# Example
print(coinChange([1,2,5], 11))  # Output: 3
```

2️⃣ Longest Increasing Subsequence (LIS)

```
def lengthOfLIS(nums):
    dp = [1] * len(nums)

    for i in range(len(nums)):
        for j in range(i):
            if nums[i] > nums[j]:
                dp[i] = max(dp[i], dp[j] + 1)

    return max(dp)

# Example
print(lengthOfLIS([10,9,2,5,3,7,101,18]))  # Output: 4
```

Approach 2 (DP + Binary Search, O(n log n)):

```
import bisect

def lengthOfLIS(nums):
    sub = []
    for num in nums:
        i = bisect.bisect_left(sub, num)
        if i == len(sub):
            sub.append(num)
        else:
            sub[i]
```

If token is available → consu

If no token → reject or delay

4. When Tokens Run Out

Requests are throttled (HTTP

5. Distributed Systems

Token count stored in **Redis**

Ensures consistency across m

AI/ML & Prompt + Core Lang (JS/Pytho	**Testing**

Question 1 - “Which automation tool would you choose for a production system and why?”
Answer - >> “For modern web applications, I’d prefer Playwright because it supports multiple browsers, has built-in waits which reduce flakiness, and integrates well with CI pipelines. For legacy systems, Selenium is still useful.”
Question 2 - >If you had to choose ONE tool (Playwright / Cypress / Selenium) for a production-grade MERN application, which would you choose and why?
Answer ->>> I would choose Playwright for a production-grade MERN application because:

a> It is designed for modern web applications and supports true cross-browser testing across Chromium, Firefox, and WebKit using a single API.

b> Its built-in auto-waiting mechanism ensures elements are actionable (visible, enabled, and ready to receive events), which significantly reduces flaky tests in CI pipelines.

c> BrowserContext allows isolated sessions similar to incognito profiles, enabling parallel, independent test execution — essential for large regression

er of tokens (e.g., 10 tokens).

equest.

↓

Generate Report

ate (e.g., 1 token per second). ↓

x size.

me token → allow request

request

429: Too Many Requests)

ultiple servers