Practice Sheet 2

Bhai ye **LC 540 – Single Element in a Sorted Array** ek classic **Binary Search on Index Pattern** wala question hai. Agar tu ye samajh gaya na, toh bahut saare pattern-based BS questions clear ho jayenge 🔥

🧠 Problem Statement (Simple Language)

- Sorted array diya hai
- Har element **2 baar** aata hai
- Sirf **ek element 1 baar** aata hai
- O(log n) time me find karna hai

Example:

```
Input:  [1,1,2,3,3,4,4,8,8]
Output: 2
```

🔎 Step 1: Intuition (Sabse Important)

Normal case me:

```
Index: 0 1 2 3 4 5 6 7 8
Array: 1 1 2 3 3 4 4 8 8
```

Pairs ka pattern observe kar:

Before single element:

- First element pair ka **even index** par start hota hai

  (0,1) (2,3) (4,5)...

After single element:

- Pattern shift ho jaata hai

  Pair **odd index** par start hota hai

Example samajh:

```
[1,1,2,3,3,4,4,8,8]
        ↑
      single = 2
```

Left side pattern:

```
(0,1) 1,1
```

Right side pattern:

```
(3,4) 3,3
(5,6) 4,4
(7,8) 8,8
```

Yaha se ek powerful observation milta hai:

👉 **Before single element**

Even index == next element

👉 **After single element**

Even index != next element

Ye hi pura binary search ka base hai 💡

🐢 Brute Force Approach (O(n))

Idea:

Har pair check karo:

- Agar arr[i] != arr[i+1] → wahi answer
- i += 2

Java Code (Brute)

```
class Solution {
    public int singleNonDuplicate(int[] nums) {
        int n = nums.length;
```

```
        for (int i = 0; i < n - 1; i += 2) {
            if (nums[i] != nums[i + 1]) {
                return nums[i];
            }
        }

        return nums[n - 1];
    }
}
```

Time Complexity:

O(n)

Space:

O(1)

⚡ Better Approach (XOR Trick) – O(n)

Property:

```
a ^ a = 0
a ^ 0 = a
```

Toh saare numbers XOR karo → pairs cancel ho jayenge.

Java Code:

```
class Solution {
    public int singleNonDuplicate(int[] nums) {
        int xor = 0;

        for (int num : nums) {
            xor ^= num;
        }

        return xor;
```

```
        }
    }
```

Time:

O(n)

Space:

O(1)

⚠️ Lekin question O(log n) maang raha hai → ye fully optimal nahi.

🚀 Optimal Approach (Binary Search – O(log n))

Ab asli game yaha hai 🔥

Core Logic:

1. mid nikalo
2. Check karo:
    - Agar mid even hai:
        - nums[mid] == nums[mid+1]

          → single right side me hai
        - else

          → single left me hai (including mid)
    - Agar mid odd hai:
        - nums[mid] == nums[mid-1]

          → single right side me hai
        - else

          → single left me hai

Aur smart trick 🔥

Instead of even/odd alag check karne ke:

```
mid ^ 1
```

Ye magic operation hai 😎

- Agar mid even hai → mid ^ 1 = mid + 1
- Agar mid odd hai → mid ^ 1 = mid - 1

Example:

```
4 ^ 1 = 5
5 ^ 1 = 4
```

Toh directly compare kar sakte:

```
nums[mid] == nums[mid ^ 1]
```

🧠 Binary Search Algorithm

1. low = 0
2. high = n - 1
3. while (low < high)
4. mid = (low + high) / 2
5. If nums[mid] == nums[mid ^ 1]

   → low = mid + 1

   else

   → high = mid
6. return nums[low]

🔥 Java Code (Optimal)

```
class Solution {
    public int singleNonDuplicate(int[] nums) {
```

```
        int low = 0;
        int high = nums.length - 1;

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

            // Check pair using XOR trick
            if (nums[mid] == nums[mid ^ 1]) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }

        return nums[low];
    }
}
```

🎯 Why while(low < high)?

Kyuki:

- Hum ek single element pe converge kar rahe hain
- Infinite loop avoid karne ke liye
- Final answer low pe milega

🔎 Dry Run Example

Input:

```
[1,1,2,3,3,4,4,8,8]
```

Step 1:

```
low = 0
high = 8
```

```
mid = 4
```

nums[4] = 3

4 ^ 1 = 5

nums[5] = 4

3 != 4

→ high = mid

Next:

```
low = 0
high = 4
mid = 2
```

nums[2] = 2

2 ^ 1 = 3

nums[3] = 3

2 != 3

→ high = 2

Next:

```
low = 0
high = 2
mid = 1
```

nums[1] = 1

1 ^ 1 = 0

nums[0] = 1

Equal

→ low = mid + 1 = 2

Now:

```
low = 2
high = 2
```

Return nums[2] = 2 ✅

⏱️ Complexity

Time: O(log n)

Space: O(1)

🧠 Important Pattern Learnings

Ye question sikhata hai:

- Index parity pattern
- Binary search on pattern
- mid ^ 1 trick
- Shrinking search space correctly
- while(low < high) vs <= difference

🏆 Interview Ready Summary

Approach	Time	Space
Brute	O(n)	O(1)
XOR	O(n)	O(1)
Binary Search	O(log n)	O(1)

Bhai ye **LC 1095 – Find in Mountain Array** thoda interesting + tricky question hai 🔥

Ye simple binary search nahi hai — isme **3 binary search** lagte hain.

Agar tu isko samajh gaya, toh “search in bitonic array” type ke saare questions easy ho jayenge.

🧠 Problem Samajh

Mountain Array ka matlab:

1. Strictly increasing
2. Fir strictly decreasing

Example:

```
[1, 2, 3, 4, 5, 3, 1]
          ↑
        peak
```

Tujhe ek `target` diya hai.

Return uska index.

⚠️ Important:

- Direct array access allowed nahi
- Sirf `mountainArr.get(i)` use kar sakte ho
- Total calls limited hoti hain

🔥 Overall Strategy (3 Steps)

Step 1 → Peak element find karo

Step 2 → Left side (ascending) me binary search

Step 3 → Right side (descending) me binary search

🏔️ Step 1: Peak Find Karna

Peak element wo hai jo:

```
nums[mid] > nums[mid+1]
```

Binary Search logic:

- If mid < mid+1 → peak right me

- Else → peak left me (including mid)

Peak Code

```
int low = 0;
int high = n - 1;

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

    if (mountainArr.get(mid) < mountainArr.get(mid + 1)) {
        low = mid + 1;
    } else {
        high = mid;
    }
}

int peak = low;
```

🔎 Step 2: Ascending Part me Search

Simple binary search (normal increasing array)

```
int leftSearch(int low, int high, int target, MountainArray
mountainArr) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        int val = mountainArr.get(mid);

        if (val == target) return mid;

        if (val < target) low = mid + 1;
        else high = mid - 1;
    }
    return -1;
}
```

🔁 Step 3: Descending Part me Search

Yaha condition reverse ho jaati hai:

```
int rightSearch(int low, int high, int target, MountainArra
y mountainArr) {
    while (low <= high) {
        int mid = low + (high - low) / 2;
        int val = mountainArr.get(mid);

        if (val == target) return mid;

        if (val > target) low = mid + 1;   // reverse
        else high = mid - 1;
    }
    return -1;
}
```

🚀 Final Optimal Java Code

```
class Solution {
    public int findInMountainArray(int target, MountainArra
y mountainArr) {
        int n = mountainArr.length();

        // 1️⃣ Find Peak
        int low = 0;
        int high = n - 1;

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

            if (mountainArr.get(mid) < mountainArr.get(mid
+ 1)) {
                low = mid + 1;
            } else {
                high = mid;
```

```
            }
        }

        int peak = low;

        // 2️⃣ Search Left (Ascending)
        int index = binarySearch(mountainArr, target, 0, pe
ak, true);
        if (index != -1) return index;

        // 3️⃣ Search Right (Descending)
        return binarySearch(mountainArr, target, peak + 1,
n - 1, false);
    }

    private int binarySearch(MountainArray mountainArr, int
target, int low, int high, boolean isAscending) {
        while (low <= high) {
            int mid = low + (high - low) / 2;
            int val = mountainArr.get(mid);

            if (val == target) return mid;

            if (isAscending) {
                if (val < target) low = mid + 1;
                else high = mid - 1;
            } else {
                if (val > target) low = mid + 1;
                else high = mid - 1;
            }
        }
        return -1;
    }
}
```

⏱️ Complexity

- Peak find → O(log n)
- Left search → O(log n)
- Right search → O(log n)

Total → **O(log n)**

Space → O(1)

🧠 Important Interview Concepts

Ye question sikhata hai:

- Binary search on mountain array
- Peak finding using BS
- Ascending + Descending search difference
- Search space splitting
- Multiple binary searches

🎯 Dry Run Example

Input:

```
arr = [1,2,3,4,5,3,1]
target = 3
```

Peak = 5 (index 4)

Left search → index 2

Return 2 (smallest index required)

Bhai ye **LC 1060 – Missing Element in Sorted Array** ek solid **Binary Search on Answer / Missing Count Pattern** wala question hai 🔥

Agar tu isko deeply samajh gaya, toh “kth missing number” type ke saare questions crack ho jayenge.

🧠 Problem Samajh

Tujhe diya hai:

- Ek **strictly increasing sorted array**
- Ek integer `k`
- Tujhe **k-th missing number** find karna hai

Example:

```
nums = [4,7,9,10]
k = 1
Output = 5
```

Missing numbers:

```
5,6,8,...
```

1st missing = 5

🔎 Sabse Important Observation

Index `i` tak kitne numbers missing hue hain?

Expected number at index i agar continuous hota:

```
nums[0] + i
```

But actual value hai:

```
nums[i]
```

Toh missing count at index i:

```
missing(i) = nums[i] - nums[0] - i
```

Example

```
nums = [4,7,9,10]
```

At index 0:

```
4 - 4 - 0 = 0
```

At index 1:

```
7 - 4 - 1 = 2   (5,6 missing)
```

At index 2:

```
9 - 4 - 2 = 3   (5,6,8 missing)
```

At index 3:

```
10 - 4 - 3 = 3
```

🧠 Case Analysis

Case 1:

Agar total missing < k

Toh answer array ke bahar hoga.

Example:

```
nums = [4,7,9,10]
k = 5
```

Total missing = 3

Toh answer:

```
nums[last] + (k - totalMissing)
```

Case 2:

k missing array ke andar hi mil jayega

Toh binary search use karenge.

🚀 Optimal Approach (Binary Search)

Goal:

Smallest index find karo jaha

```
missing(index) >= k
```

🔥 Algorithm

1. low = 0
2. high = n-1
3. Check:

   ```
   totalMissing = nums[n-1] - nums[0] - (n-1)
   ```

4. Agar totalMissing < k:

   ```
   return nums[n-1] + (k - totalMissing)
   ```

5. Warna binary search:

```
while(low < high)
    mid = ...
    if missing(mid) < k
        low = mid + 1
    else
        high = mid
```

1. Final answer:

```
nums[0] + low + k - 1
```

🧠 Why Final Formula Works?

low pe aake hum us first index pe khade hain jaha:

```
missing(low) >= k
```

Toh answer hoga:

```
nums[0] + low + k - 1
```

Ye derived formula hai jo continuous sequence logic se aata hai.

🔥 Full Java Code (Optimal)

```
class Solution {
    public int missingElement(int[] nums, int k) {
        int n = nums.length;

        // Function to calculate missing numbers till index
i
        // missing(i) = nums[i] - nums[0] - i

        int totalMissing = nums[n - 1] - nums[0] - (n - 1);

        // Case: kth missing beyond array
        if (totalMissing < k) {
            return nums[n - 1] + (k - totalMissing);
        }

        int low = 0;
        int high = n - 1;

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

            int missing = nums[mid] - nums[0] - mid;
```

```
            if (missing < k) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }

        return nums[0] + low + k - 1;
    }
}
```

🐢 Brute Force Approach (O(n + k))

Idea:

- Previous number track karo
- Gap count karo
- Missing count karte jao

Code:

```
class Solution {
    public int missingElement(int[] nums, int k) {
        for (int i = 1; i < nums.length; i++) {
            int gap = nums[i] - nums[i - 1] - 1;

            if (k <= gap) {
                return nums[i - 1] + k;
            }

            k -= gap;
        }

        return nums[nums.length - 1] + k;
    }
}
```

⏱️ Complexity

Approach	Time	Space
Brute	O(n)	O(1)
Optimal	O(log n)	O(1)

🧠 Interview Pattern

Ye question sikhata hai:

- Missing count formula derive karna
- Binary search on answer
- Search space shrinking
- While(low < high) logic

🎯 Dry Run Example

```
nums = [4,7,9,10]
k = 3
```

missing(0) = 0

missing(1) = 2

missing(2) = 3

Binary search will land at index 2

Final:

```
4 + 2 + 3 - 1 = 8
```

Answer = 8 ✅

Sahi baat boli bhai 👍

Approach build karni hai toh **brute** → **better** → **optimal** full journey samajhni padegi.

Chal ab **LC 1095 – Find in Mountain Array** ko properly todte hain 🔥

🏔️ LC 1095 – Find in Mountain Array

Problem Summary

- Array mountain type hai:
  - Strictly increasing
  - Fir strictly decreasing
- Ek `target` diya hai
- Return smallest index of target
- Direct array access allowed nahi (sirf `get()`)

Example:

```
arr = [1,2,3,4,5,3,1]
target = 3
Output = 2
```

🧠 Step 0: Structure Samjho

Mountain array ka shape:

```
      peak
        ^
1 2 3 4 5 3 1
```

Isme 3 logical parts hain:

1. Increasing part
2. Peak
3. Decreasing part

Toh natural thought process:

👉 Pehle peak dhoondo

👉 Fir dono side search karo

🐢 Brute Force Approach (O(n))

Idea:

Linear search karo.

Since direct index access nahi hai, toh:

```
for i = 0 to n-1
    if get(i) == target
         return i
```

Java Code (Conceptual)

```
class Solution {
    public int findInMountainArray(int target, MountainArra
y mountainArr) {
        int n = mountainArr.length();

        for (int i = 0; i < n; i++) {
            if (mountainArr.get(i) == target) {
                return i;
            }
        }

        return -1;
    }
}
```

Time:

O(n)

Problem:

LeetCode me `get()` calls limited hoti hain

Ye approach TLE ya limit exceed karega ❌

⚡ Better Approach (Peak + Linear Search)

Thoda dimaag lagate hain.

Step 1:

Peak dhoondo (O(log n))

Step 2:

Left side linear search (O(n))

Step 3:

Right side linear search (O(n))

Total: O(n)

Still not optimal but better than blind search.

Peak Find Logic

Binary search:

- If mid < mid+1 → peak right
- Else → peak left

```
int low = 0, high = n - 1;

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

    if (mountainArr.get(mid) < mountainArr.get(mid + 1)) {
        low = mid + 1;
    } else {
        high = mid;
    }
}

int peak = low;
```

Then Linear Search

```
// left side
for (int i = 0; i <= peak; i++) {
    if (mountainArr.get(i) == target) return i;
}

// right side
for (int i = peak + 1; i < n; i++) {
    if (mountainArr.get(i) == target) return i;
}
```

Time: O(n)

🚀 Optimal Approach (3 Binary Searches)

Ab real thinking start 🔥

Mountain array me:

- Left side sorted increasing
- Right side sorted decreasing

Toh dono side pe binary search lag sakta hai.

🧠 Final Plan

Step 1 → Peak find (O(log n))

Step 2 → Binary search left (O(log n))

Step 3 → Binary search right (O(log n))

Total: O(log n)

🔥 Full Optimal Java Code

```
class Solution {
    public int findInMountainArray(int target, MountainArra
y mountainArr) {
        int n = mountainArr.length();

        // 1️⃣ Find Peak
        int low = 0, high = n - 1;

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

            if (mountainArr.get(mid) < mountainArr.get(mid
+ 1)) {
                low = mid + 1;
            } else {
                high = mid;
            }
        }

        int peak = low;

        // 2️⃣ Binary Search Left (Ascending)
        int index = binarySearch(mountainArr, target, 0, pe
ak, true);
        if (index != -1) return index;

        // 3️⃣ Binary Search Right (Descending)
        return binarySearch(mountainArr, target, peak + 1,
n - 1, false);
    }

    private int binarySearch(MountainArray mountainArr, int
target,
                             int low, int high, boolean asc
ending) {

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

```
            int val = mountainArr.get(mid);

            if (val == target) return mid;

            if (ascending) {
                if (val < target) low = mid + 1;
                else high = mid - 1;
            } else {
                if (val > target) low = mid + 1;
                else high = mid - 1;
            }
        }

        return -1;
    }
}
```

🔎 Dry Run

```
arr = [1,2,3,4,5,3,1]
target = 3
```

Peak = index 4

Left binary search → finds 3 at index 2

Return 2 ✅

⏱️ Complexity Comparison

Approach	Time	Space
Brute	O(n)	O(1)
Better	O(n)	O(1)
Optimal	O(log n)	O(1)

🧠 Important Concepts

Ye question sikhata hai:

- Binary search on peak
- Bitonic array search
- Binary search in descending array
- Search space splitting

Sahi hai bhai 👌

Ab mai **LC 1060 – Missing Element in Sorted Array** ka full build-up dunga:

- Brute
- Better
- Optimal (Binary Search)
- Aur **har important line pe comment** bhi hoga ki kyu likha hai 🔥

🧠 Problem Recap

Given:

```
nums = sorted strictly increasing array
k = kth missing number
```

Return the kth missing number starting from nums[0].

Example:

```
nums = [4,7,9,10]
k = 3

Missing numbers:
5,6,8,...

Output = 8
```

🐢 1️⃣ Brute Force (Gap Counting) – O(n)

Idea

Har consecutive pair ke beech gap check karo:

```
gap = nums[i] - nums[i-1] - 1
```

Ye batata hai kitne numbers missing hain dono ke beech.

✅ Brute Code with Detailed Comments

```
class Solution {
    public int missingElement(int[] nums, int k) {

        // Traverse from 2nd element
        for (int i = 1; i < nums.length; i++) {

            // Calculate how many numbers are missing betwe
en nums[i-1] and nums[i]
            // Example: 4 and 7 → gap = 7 - 4 - 1 = 2 (5,6
missing)
            int gap = nums[i] - nums[i - 1] - 1;

            // If kth missing lies inside this gap
            if (k <= gap) {
                // Answer will be k steps after nums[i-1]
                return nums[i - 1] + k;
            }

            // Otherwise subtract this gap from k
            // because we have already skipped these missin
g numbers
            k -= gap;
        }

        // If kth missing not found inside array
        // It means answer is beyond last element
```

```
            return nums[nums.length - 1] + k;
        }
    }
```

⏱️ Time Complexity:

O(n)

⚡ 2️⃣ Better Understanding Before Optimal

Ab sabse important concept samajh:

Missing count till index i:

Agar array continuous hota:

```
Expected at index i = nums[0] + i
```

But actual value hai:

```
nums[i]
```

So missing till i:

```
missing(i) = nums[i] - nums[0] - i
```

Example:

```
nums = [4,7,9,10]
```

Index 1:

```
7 - 4 - 1 = 2
(5,6 missing)
```

Ye formula hi pura binary search ka base hai 🔥

🚀 3️⃣ Optimal Approach – Binary Search (O(log n))

Step 1: Check Total Missing

Total missing till last index:

```
totalMissing = nums[n-1] - nums[0] - (n-1)
```

Agar:

```
totalMissing < k
```

Toh answer array ke baahar hoga.

✅ Optimal Code with Full Comments

```
class Solution {
    public int missingElement(int[] nums, int k) {

        int n = nums.length;

        // Calculate total missing numbers till last index
        // missing(n-1) = nums[n-1] - nums[0] - (n-1)
        int totalMissing = nums[n - 1] - nums[0] - (n - 1);

        // If kth missing is beyond last element
        if (totalMissing < k) {

            // We move forward from last element
            // remaining missing = k - totalMissing
            return nums[n - 1] + (k - totalMissing);
        }

        int low = 0;
        int high = n - 1;
```

```
        // Binary search to find smallest index
        // where missing(index) >= k
        while (low < high) {

            // Standard mid calculation (overflow safe)
            int mid = low + (high - low) / 2;

            // Missing numbers till mid index
            int missing = nums[mid] - nums[0] - mid;

            if (missing < k) {
                // kth missing is on right side
                low = mid + 1;
            } else {
                // kth missing is on left side (including m
id)
                high = mid;
            }
        }

        // At this point, low == high
        // This is first index where missing(low) >= k

        // Final formula derived from continuous pattern
        return nums[0] + low + k - 1;
    }
}
```

🧠 Why Final Formula Works?

We know:

```
Continuous number at index low would be:
nums[0] + low
```

Since k missing numbers exist before this index,

Final answer:

```
nums[0] + low + k - 1
```

🔎 Dry Run

```
nums = [4,7,9,10]
k = 3
```

missing(0) = 0

missing(1) = 2

missing(2) = 3

Binary search stops at index 2

Final:

```
4 + 2 + 3 - 1 = 8
```

Answer = 8 ✅

⏱️ Complexity

Approach	Time	Space
Brute	O(n)	O(1)
Optimal	O(log n)	O(1)

Bhai ye **LC 1283 – Find the Smallest Divisor Given a Threshold** ek classic

🔥 **Binary Search on Answer** problem hai.

Isko properly build karte hain:

- Brute
- Better thinking
- Optimal (Binary Search)
- Java code with detailed comments

🧠 Problem Samajh

Given:

```
nums = [1,2,5,9]
threshold = 6
```

Humko smallest divisor `d` find karna hai such that:

```
ceil(1/d) + ceil(2/d) + ceil(5/d) + ceil(9/d) <= threshold
```

Example:

If d = 5

```
ceil(1/5)=1
ceil(2/5)=1
ceil(5/5)=1
ceil(9/5)=2

Total = 5 <= 6 ✅
```

Answer = 5

🧠 Observation

- Jaise divisor bada hota hai → sum chhota hota hai
- Jaise divisor chhota hota hai → sum bada hota hai

👉 Monotonic relation hai

Isliye **Binary Search on divisor** laga sakte hain.

🐢 1️⃣ Brute Force Approach

Try all divisors from 1 to max(nums)

For each divisor:

- Calculate sum of ceil(nums[i] / divisor)

- If sum <= threshold → return divisor

Brute Code (Java)

```
class Solution {
    public int smallestDivisor(int[] nums, int threshold) {

        int max = 0;

        // Find maximum value in array
        for (int num : nums) {
            max = Math.max(max, num);
        }

        // Try all divisors from 1 to max
        for (int d = 1; d <= max; d++) {

            int sum = 0;

            for (int num : nums) {
                // ceil division trick
                sum += (num + d - 1) / d;
            }

            if (sum <= threshold) {
                return d;
            }
        }

        return -1; // theoretically won't happen
    }
}
```

⏱️ Time Complexity:

O(n * max(nums)) ❌

Very slow if max(nums) = 10^6

⚡ 2️⃣ Better Thinking

Search space kya hai?

Divisor minimum = 1

Divisor maximum = max(nums)

Answer definitely isi range me hai.

Since monotonic behavior hai:

```
Small divisor → sum big
Big divisor → sum small
```

Binary search apply kar sakte hain.

🚀 3️⃣ Optimal Approach – Binary Search on Answer

Step 1: Search Space

```
low = 1
high = max(nums)
```

Step 2: Mid Divisor Try karo

Compute:

```
sum = Σ ceil(nums[i] / mid)
```

If:

```
sum <= threshold
```

→ mid valid ho sakta hai

→ try smaller divisor (high = mid - 1)

Else:

→ divisor chhota hai

→ increase it (low = mid + 1)

🔥 Important Ceil Trick

Instead of:

```
Math.ceil((double)num / mid)
```

Use:

```
(num + mid - 1) / mid
```

Why?

Because integer math me ye ceil ka kaam karta hai.

Example:

```
9 / 5 → 1
(9 + 4)/5 = 13/5 = 2 ✅
```

✅ Optimal Java Code (With Full Comments)

```
class Solution {
    public int smallestDivisor(int[] nums, int threshold) {

        int low = 1;   // Minimum possible divisor

        int high = 0;

        // Find maximum value to set upper bound
        for (int num : nums) {
            high = Math.max(high, num);
        }
```

```
        int answer = high;  // Store smallest valid divisor

        while (low <= high) {

            // Standard mid calculation (overflow safe)
            int mid = low + (high - low) / 2;

            int sum = 0;

            // Calculate sum of ceil divisions
            for (int num : nums) {

                // Ceil trick:
                // (num + mid - 1) / mid
                // This ensures proper rounding up
                sum += (num + mid - 1) / mid;
            }

            if (sum <= threshold) {

                // mid works, but we want smallest divisor
                answer = mid;

                // Try to find even smaller valid divisor
                high = mid - 1;

            } else {

                // Sum too large → divisor too small
                // Increase divisor
                low = mid + 1;
            }
        }

        return answer;
    }
}
```

🔎 Dry Run

```
nums = [1,2,5,9]
threshold = 6
```

Search space: 1 → 9

mid = 5

sum = 1+1+1+2 = 5 <= 6

Valid → try smaller

Eventually answer = 5

⏱️ Complexity

Approach	Time	Space
Brute	O(n * max) ❌	
Optimal	O(n log max) ✅	

🧠 Pattern Category

Ye question aata hai:

- Binary Search on Answer
- Minimum possible value
- Monotonic function search
- Same pattern as:
  - Koko Eating Bananas
  - Capacity to Ship Packages
  - Allocate Books
  - Painter Partition

Bhai ye **LC 560 – Subarray Sum Equals K** bahut hi important question hai 🔥

Ye **Prefix Sum + HashMap** ka gold standard problem hai.

Isko samajh liya toh sliding window aur prefix dono clear ho jayenge.

Chal step by step build karte hain:

- Brute
- Better
- Optimal (Prefix + HashMap)
- Java code with detailed comments

🧠 Problem Samajh

Given:

```
nums = [1,1,1]
k = 2
```

Find number of **continuous subarrays** whose sum = k

Output = 2

Subarrays:

```
[1,1] (index 0-1)
[1,1] (index 1-2)
```

🐢 1️⃣ Brute Force – O(n^3)

Idea:

- Har possible subarray generate karo
- Har subarray ka sum calculate karo

Code:

```
class Solution {
    public int subarraySum(int[] nums, int k) {

        int count = 0;
```

```
        int n = nums.length;

        for (int i = 0; i < n; i++) {

            for (int j = i; j < n; j++) {

                int sum = 0;

                // Calculate sum from i to j
                for (int x = i; x <= j; x++) {
                    sum += nums[x];
                }

                if (sum == k) {
                    count++;
                }
            }
        }

        return count;
    }
}
```

⏱️ Time: O(n³) ❌

⚡ 2️⃣ Better Approach – O(n²)

Observation:

Instead of recalculating sum again and again,

Running sum use kar sakte hain.

Idea:

Fix start index

End move karo aur running sum maintain karo

Code:

```
class Solution {
    public int subarraySum(int[] nums, int k) {

        int count = 0;
        int n = nums.length;

        for (int i = 0; i < n; i++) {

            int sum = 0;  // Reset for every new start

            for (int j = i; j < n; j++) {

                sum += nums[j];  // Add next element

                if (sum == k) {
                    count++;
                }
            }
        }

        return count;
    }
}
```

⏱️ Time: O(n²)

Better but still slow for n = 10^5 ❌

🚀 3️⃣ Optimal – Prefix Sum + HashMap (O(n))

Now real thinking 🔥

🧠 Core Idea

Let:

```
prefixSum[i] = sum from index 0 to i
```

If:

```
prefixSum[j] - prefixSum[i] = k
```

That means subarray (i+1 to j) has sum = k

Rearrange:

```
prefixSum[j] - k = prefixSum[i]
```

👉 So for every prefixSum[j],

we need to check:

```
(prefixSum[j] - k) pehle kab mila tha?
```

Ye hi hashmap karega 🔥

🎯 Algorithm

1. HashMap me store karenge:

   ```
   prefixSum -> frequency
   ```

2. Start with:

   ```
   map.put(0,1)
   ```

   Why?

   Because agar prefixSum khud k ke equal ho,

   toh subarray starting from index 0 count ho.

3. Loop through array:
   - prefixSum += nums[i]
   - If (prefixSum - k) exists in map → count += freq

- Store prefixSum in map

✅ Optimal Code (With Detailed Comments)

```
import java.util.HashMap;

class Solution {
    public int subarraySum(int[] nums, int k) {

        // Map to store prefixSum and its frequency
        HashMap<Integer, Integer> map = new HashMap<>();

        // Important:
        // prefixSum = 0 occurs once before starting
        // This handles cases where subarray starts from in
dex 0
        map.put(0, 1);

        int prefixSum = 0;
        int count = 0;

        for (int num : nums) {

            // Add current number to prefix sum
            prefixSum += num;

            // Check if (prefixSum - k) already exists
            // If yes, we found a valid subarray
            if (map.containsKey(prefixSum - k)) {

                // Add its frequency
                count += map.get(prefixSum - k);
            }

            // Store current prefixSum in map
            // If already present, increase frequency
```

```
            map.put(prefixSum, map.getOrDefault(prefixSum,
0) + 1);
        }

        return count;
    }
}
```

🔎 Dry Run Example

```
nums = [1,1,1]
k = 2
```

Start:

map = {0:1}

Iteration 1:

prefixSum = 1

1 - 2 = -1 (not found)

map = {0:1, 1:1}

Iteration 2:

prefixSum = 2

2 - 2 = 0 (found 1 time)

count = 1

map = {0:1, 1:1, 2:1}

Iteration 3:

prefixSum = 3

3 - 2 = 1 (found 1 time)

count = 2

Answer = 2 ✅

⏱️ Complexity

Approach	Time	Space
Brute	$O(n^3)$	
Better	$O(n^2)$	
Optimal	O(n)	O(n)

🧠 Important Insight

Ye sliding window se solve nahi hota ❌

Kyuki:

- Negative numbers ho sakte hain
- Window expand/shrink monotonic nahi hai

Isliye prefix + hashmap hi correct approach hai.

Bhai ye **LC 525 – Contiguous Array** prefix + hashmap ka next level version hai 🔥

LC 560 me hum **count** kar rahe the,

yaha hum **maximum length** nikal rahe hain.

Chal proper build-up karte hain:

- Brute
- Better
- Optimal (Prefix + HashMap)
- Java code with detailed comments

🧠 Problem Samajh

Given binary array:

```
nums = [0,1]
Output = 2
```

Find longest subarray with **equal number of 0s and 1s**

Example:

```
nums = [0,1,0]
Output = 2
```

Subarray:

```
[0,1]  or  [1,0]
```

🐢 1️⃣ Brute Force – O(n³)

Generate all subarrays

Count 0s and 1s

If equal → update max length

Code

```
class Solution {
    public int findMaxLength(int[] nums) {

        int n = nums.length;
        int maxLen = 0;

        for (int i = 0; i < n; i++) {

            for (int j = i; j < n; j++) {

                int zero = 0;
                int one = 0;

                for (int k = i; k <= j; k++) {

                    if (nums[k] == 0) zero++;
                    else one++;
                }

                if (zero == one) {
```

```
                    maxLen = Math.max(maxLen, j - i + 1);
                }
            }
        }

        return maxLen;
    }
}
```

⏱️ O(n³) ❌

⚡ 2️⃣ Better – O(n²)

Instead of inner loop for counting,

running count maintain karte hain.

```
class Solution {
    public int findMaxLength(int[] nums) {

        int n = nums.length;
        int maxLen = 0;

        for (int i = 0; i < n; i++) {

            int zero = 0;
            int one = 0;

            for (int j = i; j < n; j++) {

                if (nums[j] == 0) zero++;
                else one++;

                if (zero == one) {
                    maxLen = Math.max(maxLen, j - i + 1);
                }
            }
        }
```

```
            return maxLen;
        }
    }
```

⏱️ $O(n^2)$

Still slow for $n = 10^5$ ❌

🚀 3️⃣ Optimal – Prefix + HashMap (O(n))

Ab real thinking 🔥

🧠 Key Trick

Replace:

```
0 → -1
1 → +1
```

Why?

Because if we do that:

Equal number of 0 and 1 means:

```
Total sum = 0
```

Example:

```
[0,1]
→ [-1,+1]
→ sum = 0
```

🧠 Prefix Logic

If:

```
prefixSum[j] == prefixSum[i]
```

That means:

Subarray (i+1 to j) ka sum = 0

Which means equal 0 and 1.

So idea:

Store first occurrence of each prefixSum in hashmap.

Whenever prefixSum repeats,

length = currentIndex - firstIndex

🎯 Algorithm

1. map.put(0, -1)

   (Important base case)
2. Traverse array:
   - If nums[i] == 0 → treat as -1
   - Else → +1
   - Update prefixSum
   - If prefixSum seen before:

     length = i - firstIndex
   - Else store it

✅ Optimal Code (With Detailed Comments)

```
import java.util.HashMap;

class Solution {
    public int findMaxLength(int[] nums) {

        HashMap<Integer, Integer> map = new HashMap<>();

        // Important:
```

```
        // prefixSum 0 at index -1
        // Handles case when subarray starts from index 0
        map.put(0, -1);

        int prefixSum = 0;
        int maxLen = 0;

        for (int i = 0; i < nums.length; i++) {

            // Replace 0 with -1
            if (nums[i] == 0) {
                prefixSum += -1;
            } else {
                prefixSum += 1;
            }

            // If prefixSum seen before
            if (map.containsKey(prefixSum)) {

                // Subarray sum between previous index+1 an
d current index is 0
                int length = i - map.get(prefixSum);

                maxLen = Math.max(maxLen, length);
            } else {

                // Store first occurrence only
                map.put(prefixSum, i);
            }
        }

        return maxLen;
    }
}
```

🔎 Dry Run

```
nums = [0,1,0]
```

Convert:

```
[-1, +1, -1]
```

Start:

map = {0:-1}

i=0 → prefix=-1

map store -1:0

i=1 → prefix=0

Seen before at -1

length = 1 - (-1) = 2

maxLen=2

i=2 → prefix=-1

Seen before at 0

length = 2 - 0 = 2

Answer = 2 ✅

⏱️ Complexity

Approach	Time	Space
Brute	$O(n^3)$	
Better	$O(n^2)$	
Optimal	O(n)	O(n)

🧠 Important Difference (560 vs 525)

LC 560	LC 525
Count subarrays	Max length
prefixSum - k	prefixSum repeat

LC 560	LC 525
No transformation	0 → -1 trick

Bhai 🔥

LC 523 – Continuous Subarray Sum prefix + hashmap ka ek aur strong variation hai.

Ye 560 aur 525 ka cousin hai, lekin twist hai **modulo (remainder)** ka.

Chal full build-up karte hain:

- Brute
- Better
- Optimal (Prefix % k + HashMap)
- Java code with detailed comments

🧠 Problem Samajh

Given:

```
nums = [23,2,4,6,7]
k = 6
```

Check karo:

Koi continuous subarray hai jiska:

```
sum % k == 0
```

AND subarray length >= 2

Example:

```
[2,4] → sum = 6 → 6 % 6 = 0 ✅
```

Return true

🐢 1️⃣ Brute Force – $O(n^2)$

Generate all subarrays

Sum calculate karo

Check:

```
sum % k == 0 AND length >= 2
```

Brute Code

```
class Solution {
    public boolean checkSubarraySum(int[] nums, int k) {

        int n = nums.length;

        for (int i = 0; i < n; i++) {

            int sum = 0;

            for (int j = i; j < n; j++) {

                sum += nums[j];

                // Check length >= 2
                if (j - i + 1 >= 2 && sum % k == 0) {
                    return true;
                }
            }
        }

        return false;
    }
}
```

⏱️ $O(n^2)$

Large n me slow ❌

⚡ 2️⃣ Better Thinking (Prefix Idea)

Let:

```
prefixSum[i] = sum from 0 to i
```

If:

```
(prefixSum[j] - prefixSum[i]) % k == 0
```

Means:

```
prefixSum[j] % k == prefixSum[i] % k
```

💥 Important Insight:

👉 Agar do prefix sums ka remainder same hai,

toh unke beech ka sum divisible by k hoga.

🚀 3️⃣ Optimal – Prefix % k + HashMap (O(n))

🧠 Core Idea

Store:

```
remainder → first index
```

If same remainder dubara mile:

```
currentIndex - firstIndex >= 2
```

Then answer = true

🎯 Why First Index Only?

Because:

Humein maximum gap chahiye

Isliye first occurrence store karte hain.

🔥 Special Case: k = 0

Agar k = 0:

We need sum == 0

That means:

Check if 2 consecutive zeros exist.

✅ Optimal Code (With Full Comments)

```
import java.util.HashMap;

class Solution {
    public boolean checkSubarraySum(int[] nums, int k) {

        // Special case: if k == 0
        // We need at least two consecutive zeros
        if (k == 0) {
            for (int i = 1; i < nums.length; i++) {
                if (nums[i] == 0 && nums[i - 1] == 0) {
                    return true;
                }
            }
            return false;
        }

        HashMap<Integer, Integer> map = new HashMap<>();

        // Important:
        // remainder 0 at index -1
        // handles case when subarray starts from index 0
        map.put(0, -1);
```

```
        int prefixSum = 0;

        for (int i = 0; i < nums.length; i++) {

            prefixSum += nums[i];

            // Take remainder
            int remainder = prefixSum % k;

            // Handle negative remainder
            if (remainder < 0) {
                remainder += k;
            }

            if (map.containsKey(remainder)) {

                // Ensure subarray length >= 2
                if (i - map.get(remainder) >= 2) {
                    return true;
                }

            } else {

                // Store first occurrence only
                map.put(remainder, i);
            }
        }

        return false;
    }
}
```

🔎 Dry Run

```
nums = [23,2,4,6,7]
```

```
k = 6
```

Prefix sums:

```
23 → remainder 5
25 → remainder 1
29 → remainder 5
```

Remainder 5 appeared before

Gap >= 2 → TRUE ✅

⏱️ Complexity

Approach	Time	Space
Brute	$O(n^2)$	
Optimal	O(n)	O(n)

🧠 Compare 560 vs 523 vs 525

Problem	Trick
560	prefixSum - k
525	0 → -1 transformation
523	prefixSum % k repeat

Bhai 🔥

LC 135 – Candy greedy ka ek classic hard-level question hai.

Isme prefix ya binary search nahi — pure **greedy thinking + two pass logic** ka game hai.

Chal proper build-up karte hain:

- Brute
- Better thinking
- Optimal (2-pass greedy)
- Java code with detailed comments

🧠 Problem Samajh

Given:

```
ratings = [1,0,2]
```

Rules:

1. Har child ko kam se kam 1 candy milegi
2. Jiska rating zyada hai apne adjacent se → usko zyada candy milni chahiye

Goal: minimum candies distribute karo.

Example:

```
ratings = [1,0,2]

Valid distribution:
[2,1,2]

Total = 5
```

🐢 1️⃣ Brute Force (Iterative Adjusting)

Idea:

- Sabko 1 candy do
- Jab tak condition violate ho rahi hai
  - Update candies

Ye multiple passes lega.

Conceptual Code (Not Efficient)

```
class Solution {
    public int candy(int[] ratings) {

        int n = ratings.length;
```

```
        int[] candies = new int[n];

        // Give everyone 1 candy
        for (int i = 0; i < n; i++) {
            candies[i] = 1;
        }

        boolean changed = true;

        while (changed) {
            changed = false;

            for (int i = 0; i < n; i++) {

                if (i > 0 && ratings[i] > ratings[i - 1]
                    && candies[i] <= candies[i - 1]) {

                    candies[i] = candies[i - 1] + 1;
                    changed = true;
                }

                if (i < n - 1 && ratings[i] > ratings[i +
1]
                    && candies[i] <= candies[i + 1]) {

                    candies[i] = candies[i + 1] + 1;
                    changed = true;
                }
            }
        }

        int sum = 0;
        for (int c : candies) sum += c;

        return sum;
    }
}
```

⏱️ Worst case $O(n^2)$ ❌

⚡ 2️⃣ Better Thinking

Observe carefully:

Do types of constraints hain:

Left neighbor condition

If:

```
ratings[i] > ratings[i-1]
```

Then:

```
candies[i] = candies[i-1] + 1
```

Right neighbor condition

If:

```
ratings[i] > ratings[i+1]
```

Then:

```
candies[i] = candies[i+1] + 1
```

Problem:

Agar sirf left se check kiya → right violation reh jayega.

So solution:

👉 Do passes karo.

🚀 3️⃣ Optimal – Two Pass Greedy (O(n))

Step 1: Initialize

Sabko 1 candy.

Step 2: Left → Right pass

Ensure:

```
ratings[i] > ratings[i-1]
```

Step 3: Right → Left pass

Ensure:

```
ratings[i] > ratings[i+1]
```

Important:

Take max to avoid breaking previous condition.

✅ Optimal Java Code (With Full Comments)

```
class Solution {
    public int candy(int[] ratings) {

        int n = ratings.length;

        // Step 1: Give everyone 1 candy initially
        int[] candies = new int[n];

        for (int i = 0; i < n; i++) {
            candies[i] = 1;
        }

        // Step 2: Left to Right pass
        // If current rating > previous rating,
        // then current candies should be previous + 1
        for (int i = 1; i < n; i++) {
```

```
            if (ratings[i] > ratings[i - 1]) {

                candies[i] = candies[i - 1] + 1;
            }
        }

        // Step 3: Right to Left pass
        // If current rating > next rating,
        // then ensure candies[i] > candies[i+1]
        for (int i = n - 2; i >= 0; i--) {

            if (ratings[i] > ratings[i + 1]) {

                // Take max to avoid overriding left condit
ion
                candies[i] = Math.max(candies[i], candies[i
+ 1] + 1);
            }
        }

        // Step 4: Sum up candies
        int total = 0;

        for (int c : candies) {
            total += c;
        }

        return total;
    }
}
```

🔎 Dry Run

```
ratings = [1,0,2]
```

Initial:

```
[1,1,1]
```

Left pass:

```
[1,1,2]
```

Right pass:

```
[2,1,2]
```

Sum = 5 ✅

⏱️ Complexity

Approach	Time	Space
Brute	$O(n^2)$	
Optimal	O(n)	O(n)

🧠 Why Two Pass Works?

Because constraints independent hain:

- Left dependency
- Right dependency

Ek direction me solve karne se dusra violate ho sakta hai.

Isliye dono direction me enforce karna zaroori hai.

Bhai ye **LC 128 – Longest Consecutive Sequence** hashing ka classic question hai 🔥

Isme tumhari thinking brute → better → optimal tak develop honi chahiye.

🔷 Problem Statement

Given an unsorted array of integers `nums`, return the length of the longest consecutive elements sequence.

⚠️ Condition: O(n) time me solve karna hai.

Example:

```
Input: [100,4,200,1,3,2]
Output: 4
Explanation: [1,2,3,4] is longest sequence
```

🧠 Intuition Samjho Pehle

Consecutive ka matlab:

```
x, x+1, x+2, x+3 ...
```

Important:

- Array sorted nahi hai
- Order matter nahi karta
- Sirf consecutive numbers ka group find karna hai

🔴 1️⃣ Brute Force Approach

💡 Idea

Har element ke liye check karo:

- Kya `num + 1` array me exist karta hai?
- Fir `num + 2`
- Fir `num + 3`
- Jab tak milta rahe

❌ Problem

Har check ke liye pura array scan karna padega.

Time Complexity:

```
O(n^2)
```

🔹 Java Code (Brute)

```
class Solution {
    public int longestConsecutive(int[] nums) {
        int n = nums.length;
        int longest = 0;

        for (int i = 0; i < n; i++) {

            int current = nums[i];
            int count = 1;

            // check next consecutive numbers
            while (contains(nums, current + 1)) {
                current = current + 1;
                count++;
            }

            longest = Math.max(longest, count);
        }

        return longest;
    }

    // linear search helper
    private boolean contains(int[] nums, int target) {
        for (int num : nums) {
            if (num == target) return true;
        }
        return false;
    }
}
```

🟡 2️⃣ Better Approach (Sorting)

💡 Idea

1. Sort the array
2. Traverse once
3. Count consecutive numbers

🧠 Example

```
[100,4,200,1,3,2]

Sort → [1,2,3,4,100,200]
```

Ab simple adjacent compare karo.

⚠️ Handle Duplicates

Agar duplicate aaya:

```
[1,2,2,3]
```

Ignore karo.

🔹 Java Code (Better)

```
import java.util.Arrays;

class Solution {
    public int longestConsecutive(int[] nums) {

        if (nums.length == 0) return 0;

        Arrays.sort(nums);  // sorting

        int longest = 1;
```

```
        int currentStreak = 1;

        for (int i = 1; i < nums.length; i++) {

            // duplicate ignore
            if (nums[i] == nums[i - 1]) {
                continue;
            }

            // consecutive
            if (nums[i] == nums[i - 1] + 1) {
                currentStreak++;
            } else {
                currentStreak = 1; // reset
            }

            longest = Math.max(longest, currentStreak);
        }

        return longest;
    }
}
```

Time Complexity:

```
O(n log n)
```

🟢 3️⃣ Optimal Approach (HashSet – O(n))

🔥 Ye interview favourite hai.

💡 Core Idea

Instead of sorting,

Use HashSet for O(1) lookup.

Main trick:

👉 Sequence tab start karo jab `num - 1` exist na karta ho.

Matlab:

```
if (!set.contains(num - 1))
```

Ye batata hai ki ye sequence ka START hai.

🧠 Example

```
[100,4,200,1,3,2]

Set = {100,4,200,1,3,2}
```

Check each:

100 → 99 not present → start

4 → 3 present → not start

1 → 0 not present → start

Sirf 1 se hi full sequence milegi.

🔹 Java Code (Optimal)

```
import java.util.HashSet;

class Solution {
    public int longestConsecutive(int[] nums) {

        HashSet<Integer> set = new HashSet<>();

        // Step 1: put all elements into set
        for (int num : nums) {
            set.add(num);
        }

        int longest = 0;

        // Step 2: iterate through set
```

```
        for (int num : set) {

            // only start counting if num is starting point
            // starting point means num - 1 does NOT exist
            if (!set.contains(num - 1)) {

                int currentNum = num;
                int count = 1;

                // count consecutive numbers
                while (set.contains(currentNum + 1)) {
                    currentNum++;
                    count++;
                }

                longest = Math.max(longest, count);
            }
        }

        return longest;
    }
}
```

🧠 Why This Is O(n)?

Har number:

- Ek baar check hota hai
- Consecutive loop me bhi har number ek hi baar visit hota hai

Total work:

```
O(n)
```

🧠 Important Edge Cases

1. Empty array → return 0

2. Duplicates present
3. Negative numbers
4. Single element

🏆 Interview Me Kya Bolna

Step wise bolo:

1. Brute force O(n²)
2. Sorting O(n log n)
3. Hashing O(n)

Aur main trick explain karo:

👉 "We only start counting when num - 1 does not exist."

Ye line interviewer ko impress karti hai 😎

Bhai ye **LC 680 – Valid Palindrome II** two-pointer ka classic interview question hai 🔥

Isme tumhari approach build honi chahiye:

Brute → Better → Optimal (Greedy + Two Pointer)

🔷 Problem Statement

Given a string `s`, return `true` if it can be palindrome after deleting **at most one character**.

Example:

```
Input: "aba"       → true
Input: "abca"      → true  (delete 'c')
Input: "abc"       → false
```

⚠️ Only **one deletion allowed**.

🧠 Intuition

Normal palindrome check:

- Left pointer (i)
- Right pointer (j)
- Compare s[i] and s[j]

But yaha twist hai:

👉 Agar mismatch aaye, to tum ek character delete kar sakte ho.

Matlab:

Mismatch pe do option:

1. Skip left character
2. Skip right character

Agar dono me se koi bhi palindrome bana de → return true

🔴 1️⃣ Brute Force Approach

💡 Idea

Har index pe try karo:

- Ek character remove karo
- Check karo ki baaki string palindrome hai ya nahi

Time Complexity:

```
O(n²)
```

🔹 Java Code (Brute)

```
class Solution {
    public boolean validPalindrome(String s) {

        // check original first
        if (isPalindrome(s)) return true;
```

```
        // try removing each character
        for (int i = 0; i < s.length(); i++) {

            String newStr = s.substring(0, i) + s.substring
(i + 1);

            if (isPalindrome(newStr)) {
                return true;
            }
        }

        return false;
    }

    private boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left) != str.charAt(right))
                return false;
            left++;
            right--;
        }
        return true;
    }
}
```

❌ Problem:

- Har index pe substring bana rahe ho
- O(n^2) time

🟡 2️⃣ Better Approach (Two Pointer + Check Twice)

💡 Idea

Instead of removing every char,

Use two pointers directly.

When mismatch happens:

- Check substring (left+1, right)
- OR check substring (left, right-1)

Time Complexity:

```
O(n)
```

🔹 Java Code (Better / Optimal Same Concept)

```
class Solution {

    public boolean validPalindrome(String s) {

        int left = 0;
        int right = s.length() - 1;

        while (left < right) {

            if (s.charAt(left) != s.charAt(right)) {

                // try skipping left OR skipping right
                return isPalindrome(s, left + 1, right) ||
                       isPalindrome(s, left, right - 1);
            }

            left++;
            right--;
        }

        return true;
    }

    // helper function to check palindrome in range
```

```
    private boolean isPalindrome(String s, int left, int ri
ght) {

        while (left < right) {

            if (s.charAt(left) != s.charAt(right))
                return false;

            left++;
            right--;
        }

        return true;
    }
}
```

🧠 Step-by-Step Dry Run

Example:

```
s = "abca"
```

Compare:

```
a == a  → ok
b != c  → mismatch
```

Now try:

1. Skip left (b) → check "aca" → palindrome ✅
2. Skip right (c) → check "aba" → palindrome ✅

Return true.

🧠 Why Only One Mismatch Allowed?

Because:

- Question says at most one deletion
- So first mismatch pe hi decision lena hoga
- Uske baad second mismatch allowed nahi hai

🏆 Why This Is Optimal?

- Har character maximum 2 times check hota hai
- Overall time: O(n)
- Space: O(1)

🧠 Edge Cases

1. Length = 1 → true
2. Already palindrome → true
3. Two mismatches → false
4. All same characters → true

🔥 Pattern Recognition

Ye kis topic ka hai?

✔️ Two Pointer

✔️ Greedy decision

✔️ Palindrome check

✔️ String manipulation

Interview me bolna:

> "We use two pointers. At first mismatch, we try skipping either left or right once. If any of them forms a palindrome, return true."

Agar chahe to main iska tricky test cases bhi de sakta hoon jaha log galti karte hain 😎

Bhai ye **LC 392 – Is Subsequence** ek basic but very important Two Pointer pattern hai 🔥

Interview me frequently pucha jata hai, especially as warm-up.

🔷 Problem Statement

Given two strings:

- `s` (small string)
- `t` (large string)

Return `true` if `s` is a **subsequence** of `t`.

🔎 Subsequence kya hota hai?

Characters same order me hone chahiye

But continuous hona zaroori nahi.

Example:

```
s = "abc"
t = "ahbgdc"

Output: true
```

Because:

```
a _ b _ _ c
```

Example 2:

```
s = "axc"
t = "ahbgdc"

Output: false
```

🧠 Intuition

Subsequence check karne ka simplest way:

👉 `s` ko left se scan karo

👉 `t` ko left se scan karo

👉 Jab character match ho, `s` ka pointer aage badhao

Agar end me `s` pura match ho gaya → true

🔴 1️⃣ Brute Force Approach (Recursion)

💡 Idea

Har character ke liye:

- Match karo
- Ya skip karo

Basically subsequence recursion tree.

Time Complexity:

```
O(2^n)
```

🔹 Java Code (Brute Recursion)

```
class Solution {

    public boolean isSubsequence(String s, String t) {
        return helper(s, t, 0, 0);
    }

    private boolean helper(String s, String t, int i, int
j) {

        // if we matched all characters of s
        if (i == s.length()) return true;

        // if t finished but s not finished
        if (j == t.length()) return false;
```

```
        if (s.charAt(i) == t.charAt(j)) {
            // match found, move both
            return helper(s, t, i + 1, j + 1);
        } else {
            // skip character in t
            return helper(s, t, i, j + 1);
        }
    }
}
```

❌ Exponential time

Not practical for large input.

🟡 2️⃣ Better Approach (Two Pointer – Iterative)

💡 Core Idea

- One pointer `i` for `s`
- One pointer `j` for `t`

Loop until `t` khatam:

If match:

```
i++
```

Always:

```
j++
```

🔹 Java Code (Optimal)

```
class Solution {
    public boolean isSubsequence(String s, String t) {
```

```
        int i = 0; // pointer for s
        int j = 0; // pointer for t

        while (i < s.length() && j < t.length()) {

            if (s.charAt(i) == t.charAt(j)) {
                i++;  // move s pointer only if match
            }

            j++; // always move t pointer
        }

        // if we matched all characters of s
        return i == s.length();
    }
}
```

🧠 Dry Run

```
s = "abc"
t = "ahbgdc"
```

Step-by-step:

```
a == a  → i=1
h        skip
b == b  → i=2
g        skip
d        skip
c == c  → i=3
```

Now `i == s.length()` → return true

⏱️ Time & Space

Time:

```
O(n)
```

Space:

```
O(1)
```

🧠 Edge Cases

1. s empty → true
2. t empty but s not empty → false
3. s longer than t → false
4. Same strings → true

🏆 Interview Follow-Up (Important)

Interviewer puch sakta hai:

👉 Agar multiple queries ho?

Example:

- 1 string t
- 10,000 strings s

Then simple two pointer slow hoga.

🔥 Advanced Optimization (Binary Search + Preprocessing)

1. Store all indices of each character in t
2. For each char in s, binary search next valid index

Time:

```
Preprocessing: O(n)
```

```
Each query: O(m log n)
```

Agar chahe to ye advanced version bhi detail me samjha deta hoon.

Bhai ye **LC 1011 – Capacity To Ship Packages Within D Days** ek **Binary Search on Answer** ka solid question hai 🔥

Ye Painter's Partition / Allocate Books / Koko Eating Bananas family ka hi bhai hai.

Aaj isko **Brute → Better → Optimal** full build-up ke saath samjhte hain.

🔷 Problem Statement

Given:

- `weights[]` (packages in order)
- `days`

Har din tum continuous packages ship kar sakte ho.

Ship capacity fixed hogi per day.

Find **minimum capacity** so that all packages ship ho jaye in `D` days.

Example

```
weights = [1,2,3,4,5,6,7,8,9,10]
days = 5

Answer = 15
```

🧠 Intuition

Capacity kam hogi → zyada din lagenge

Capacity zyada hogi → kam din lagenge

👉 Capacity aur days ka **inverse relation** hai.

Iska matlab:

👉 **Monotonic property present hai**

👉 Binary search lag sakta hai.

🔴 1️⃣ Brute Force Approach

💡 Idea

Capacity ko 1 se lekar sum(weights) tak try karo.

Har capacity ke liye check karo:

- Kitne din lagenge?

Minimum capacity jo ≤ D days me kaam kar de → answer.

🔥 Helper Logic (Important)

Given capacity `cap`:

```
days = 1
currentLoad = 0

for each weight:
    if currentLoad + weight <= cap
         add
    else
         days++
         currentLoad = weight
```

❌ Time Complexity

Capacity range = up to sum(weights)

Worst case:

```
O(n * sum)
```

Very slow.

🟡 2️⃣ Better Approach (Smart Brute)

Observation:

Minimum capacity kya ho sakti hai?

👉 At least max(weights)

Because ek package to ship karna hi padega.

Maximum capacity kya ho sakti hai?

👉 sum(weights)

To search space:

```
[max(weights), sum(weights)]
```

But agar still linear try karoge to slow hi rahega.

🟢 3️⃣ Optimal Approach (Binary Search on Answer)

🔥 Core Idea

Binary search on capacity.

Because:

- Capacity badhegi → required days kam honge
- Capacity kam hogi → required days badhenge

Monotonic behaviour confirmed.

🔹 Algorithm Steps

1. `low = max(weights)`
2. `high = sum(weights)`

While `low <= high`:

- mid = possible capacity
- Calculate required days using helper
- If days <= D:

→ try smaller capacity

→ high = mid - 1

- Else:

→ capacity small hai

→ low = mid + 1

Return low

🧠 Why return low?

Because we want **minimum valid capacity**

Binary search end hone ke baad:

`low` hi first valid capacity hota hai.

✅ Java Code (Optimal – Fully Commented)

```
class Solution {

    public int shipWithinDays(int[] weights, int days) {

        int low = 0;
        int high = 0;

        // low should be maximum weight
        // because we must ship at least one package
        for (int w : weights) {
            low = Math.max(low, w);
            high += w;  // high is sum of all weights
        }

        // Binary Search on capacity
        while (low <= high) {

            int mid = low + (high - low) / 2;  // possible
```

```
capacity

            int requiredDays = calculateDays(weights, mid);

            if (requiredDays <= days) {
                // capacity works
                // try to minimize capacity
                high = mid - 1;
            } else {
                // capacity too small
                low = mid + 1;
            }
        }

        return low; // minimum valid capacity
    }

    // Helper function to calculate how many days needed
    private int calculateDays(int[] weights, int capacity)
{

        int days = 1;        // at least 1 day required
        int currentLoad = 0;

        for (int w : weights) {

            if (currentLoad + w <= capacity) {
                currentLoad += w;  // add to current day
            } else {
                days++;            // new day
                currentLoad = w;   // start new day load
            }
        }

        return days;
    }
}
```

🧠 Dry Run Example

```
weights = [1,2,3,4,5,6,7,8,9,10]
days = 5
```

low = 10

high = 55

Binary search karega aur answer 15 milega.

⏱️ Time Complexity

```
O(n log(sum))
```

Where:

- n = number of packages
- log(sum) = binary search range

Space:

```
O(1)
```

🔥 Pattern Recognition

Ye same pattern kis-kis me aata hai?

✔️ LC 875 – Koko Eating Bananas

✔️ LC 410 – Split Array Largest Sum

✔️ Painter Partition

✔️ Allocate Books

✔️ Aggressive Cows

Sab me same structure:

1. Search space define karo
2. Monotonic check function banao

3. Binary search on answer

🧠 Interview Me Kaise Explain Kare

Bolna:

> "We binary search on shipping capacity. Since required days decrease as capacity increases, monotonic property exists. We find the minimum capacity such that requiredDays ≤ D."

Agar chahe to main iska **diagrammatic explanation bhi bana du jaise painter partition me kiya tha**, taki binary search intuition crystal clear ho jaye 😎

Bhai 🔥

Ye **LC 410 – Split Array Largest Sum** binary search on answer ka **king-level question** hai.

Painter Partition ka direct variant hai.

Aaj isko full build-up se samjhte hain:

Brute → Better → Optimal (with full intuition + dry run + Java code)

🔷 Problem Statement

Given:

- `nums[]`
- `k` (number of subarrays)

Split array into exactly `k` continuous subarrays

Such that **largest subarray sum is minimum**

Return that minimum possible largest sum.

🔎 Example

```
nums = [7,2,5,10,8]
k = 2
```

```
Output = 18
```

Why?

Split like this:

```
[7,2,5]  [10,8]
sum1 = 14
sum2 = 18
max = 18
```

If you split any other way, max sum bigger ho jayega.

🧠 Intuition

Hum kya minimize karna chahte hain?

👉 Maximum subarray sum

So answer ek number hoga.

Important observation:

If allowed maximum sum bada kar do → fewer partitions needed

If maximum sum chhota rakho → more partitions needed

⚡ Inverse relationship again

👉 Monotonic property present

👉 Binary search possible

🔴 1️⃣ Brute Force

💡 Idea

Har possible way try karo:

- K-1 places choose karo to split
- All combinations generate karo

Time Complexity:

```
Exponential (2^n)
```

Impossible for large n.

🟡 2️⃣ Better Approach (Try All Possible Max Sums Linearly)

Search space kya ho sakta hai?

Minimum possible largest sum:

```
max(nums)
```

Maximum possible largest sum:

```
sum(nums)
```

Ab is range me har value try karo:

For each value X:

- Check karo kya array ko ≤ k parts me split kar sakte ho
- So that each part sum ≤ X

Ye linear try karna slow hoga:

```
O(n * sum)
```

🟢 3️⃣ Optimal – Binary Search on Answer

🔥 Ye final correct approach hai.

🧠 Core Idea

Binary search on maximum allowed subarray sum.

🔹 Algorithm Steps

1. low = max(nums)
2. high = sum(nums)

While (low <= high):

- mid = possible largest sum
- Check how many partitions needed if max allowed sum = mid

If partitions <= k:

→ valid

→ try smaller (high = mid - 1)

Else:

→ too small

→ increase (low = mid + 1)

Return low

🧠 Helper Function Logic (Very Important)

Given maxAllowedSum:

```
count = 1
currentSum = 0

for each num:
    if currentSum + num <= maxAllowedSum
         add
    else
         new partition
         count++
         currentSum = num
```

Return count

✅ Java Code (Fully Commented)

```
class Solution {

    public int splitArray(int[] nums, int k) {

        int low = 0;
        int high = 0;

        // low = max element
        // high = total sum
        for (int num : nums) {
            low = Math.max(low, num);
            high += num;
        }

        // Binary Search on maximum subarray sum
        while (low <= high) {

            int mid = low + (high - low) / 2;  // candidate
largest sum

            int partitions = countPartitions(nums, mid);

            if (partitions <= k) {
                // possible to split in k or fewer parts
                // try minimizing largest sum
                high = mid - 1;
            } else {
                // too many partitions needed
                // increase allowed sum
                low = mid + 1;
            }
        }

        return low;  // minimum largest sum
    }

    // Helper to count partitions needed
    private int countPartitions(int[] nums, int maxAllowedS
```

```
um) {

        int count = 1;     // at least one partition
        int currentSum = 0;

        for (int num : nums) {

            if (currentSum + num <= maxAllowedSum) {
                currentSum += num;
            } else {
                // need new partition
                count++;
                currentSum = num;
            }
        }

        return count;
    }
}
```

🧠 Dry Run

```
nums = [7,2,5,10,8]
k = 2
```

low = 10

high = 32

Binary search:

mid = 21

Partitions needed = 2

→ valid → try smaller

Eventually answer = 18

⏱️ Time Complexity

```
O(n log(sum))
```

Space:

```
O(1)
```

🔥 Why We Use start <= end Here?

Because:

- We are searching exact minimum valid value
- Need to check every possibility
- Standard binary search on answer

This is different from shrinking-space style (`start < end`) used in lower bound.

🧠 Pattern Family

Same pattern as:

✔️ LC 1011 – Ship Packages

✔️ LC 875 – Koko

✔️ Painter Partition

✔️ Allocate Books

✔️ Aggressive Cows

All are:

```
Binary Search on Answer
+ Monotonic Check Function
```

🏆 Interview Me Kaise Bolna

> "We binary search the maximum subarray sum. Since required partitions decrease as allowed sum increases, monotonic property holds. We find the minimum allowed sum such that partitions ≤ k."

Agar tu chahe to main ek **visual graph explanation bhi bana du showing monotonic behavior**, taki ye pattern permanently brain me chipak jaye 😎