Binary Search for Greedy Problems — Coding Interview Notes (Light Theme)

General Pattern Template

```
def fn(arr):
    def check(x):
        # this function is implemented depending on the problem
        return BOOLEAN

    left = MINIMUM_POSSIBLE_ANSWER
    right = MAXIMUM_POSSIBLE_ANSWER
    while left <= right:
        mid = (left + right) // 2
        if check(mid):
            right = mid - 1
        else:
            left = mid + 1

    return left
```

Concept:
This pattern applies **Binary Search** not on array indices, but on the **answer space**. It's used when the problem asks for an optimal (minimum or maximum) value that satisfies a monotonic condition.

The check(x) function determines whether a candidate answer x is **feasible** (True) or not (False). Because the condition is monotonic — if x works, all larger (or smaller) values also work — binary search efficiently narrows the range.

Time Complexity: O(log(max - min) × check_time)

Key Ideas

1. Search over the **range of possible answers**, not indices.
2. Define a **check(x)** that returns True if x satisfies the constraint.
3. Ensure the decision function is **monotonic**: if x works, so do all greater (or smaller) values.
4. Return **left** as the smallest feasible value (or right for largest feasible).
5. Typical problems: capacity minimization, speed optimization, resource allocation, partitioning.

Example 1: Minimum Capacity to Ship Packages Within D Days

Goal: Find the smallest ship capacity to ship all packages within D days.
Approach: Binary search capacity range; check() simulates shipment with that capacity.

```
def ship_within_days(weights, D):
    def check(capacity):
        days, curr = 1, 0
        for w in weights:
            if curr + w > capacity:
                days += 1
                curr = 0
            curr += w
        return days <= D

    left, right = max(weights), sum(weights)
    while left <= right:
        mid = (left + right) // 2
        if check(mid):
            right = mid - 1
        else:
            left = mid + 1
    return left

# Example
print(ship_within_days([1,2,3,4,5,6,7,8,9,10], 5))  # Output: 15
```

Example 2: Minimum Eating Speed (Koko Eating Bananas)

Goal: Find the smallest integer Koko's eating speed (bananas/hour) to finish in H hours.
Approach: Binary search speed; check() verifies if Koko can finish on time.

```
import math

def min_eating_speed(piles, H):
    def check(speed):
        hours = 0
        for p in piles:
            hours += math.ceil(p / speed)
        return hours <= H

    left, right = 1, max(piles)
    while left <= right:
        mid = (left + right) // 2
        if check(mid):
            right = mid - 1
        else:
            left = mid + 1
    return left

# Example
print(min_eating_speed([3,6,7,11], 8))  # Output: 4
```

Example 3: Allocate Books to Minimize Maximum Pages

Goal: Split books among M students minimizing the maximum pages assigned to any student.
Approach: Binary search over max pages; check() tries distributing books greedily.

```
def allocate_books(pages, m):
    def check(limit):
        count, curr = 1, 0
        for p in pages:
            if curr + p > limit:
                count += 1
                curr = 0
            curr += p
        return count <= m

    left, right = max(pages), sum(pages)
    while left <= right:
        mid = (left + right) // 2
        if check(mid):
            right = mid - 1
        else:
            left = mid + 1
    return left

# Example
print(allocate_books([12,34,67,90], 2))  # Output: 113
```

Summary Table

ProblemRangeCheck ConditionGoalReturn Ship packages[max(weights), sum(weights)]Days ≤ DMin capacityleft Koko bananas[1, max(piles)]Hours ≤ HMin speedleft Allocate books[max(pages), sum(pages)]Students ≤ MMin max pagesleft