Median of Two Sorted Arrays — Illustrated Guide

This document demonstrates two ways to compute the median of two sorted arrays in Python — a simple merge-based approach (O(m + n)) and the optimal binary-search partition approach (O(log min(m, n))). Each includes code, explanation, and complexity notes.

Method 1 — Merge-and-Find Median (Simple O(m + n))

This approach merges both sorted lists into one sorted list, then returns the middle element(s). It's intuitive, easy to code, and fine for small inputs. Each list element is visited once.

```
def find_median_sorted_arrays_simple(a, b):
    merged = []
    i = j = 0
    while i < len(a) and j < len(b):
        if a[i] < b[j]:
            merged.append(a[i])
            i += 1
        else:
            merged.append(b[j])
            j += 1
    merged.extend(a[i:])
    merged.extend(b[j:])
    n = len(merged)
    if n % 2 == 1:
        return merged[n // 2]
    else:
        return (merged[n // 2 - 1] + merged[n // 2]) / 2

# Example
print(find_median_sorted_arrays_simple([1,3], [2]))        # 2.0
print(find_median_sorted_arrays_simple([1,2], [3,4]))      # 2.5
```

Complexity: O(m + n) time and O(m + n) space.

Method 2 — Binary Search Partition (Optimal O(log min(m, n)))

This approach uses binary search on the smaller array to find the perfect partition such that: max(leftA, leftB) ≤ min(rightA, rightB). When total length is odd, median = min(rightA, rightB); else, it's the average of max(leftA, leftB) and min(rightA, rightB).

```
def find_median_sorted_arrays(a, b):
    # Ensure a is the smaller array
    if len(a) > len(b):
        a, b = b, a
    m, n = len(a), len(b)
    total = m + n
    half = total // 2

    lo, hi = 0, m
    while lo <= hi:
        i = (lo + hi) // 2       # partition index in a
        j = half - i             # partition index in b

        leftA  = a[i-1] if i > 0 else float('-inf')
        rightA = a[i]   if i < m else float('inf')
        leftB  = b[j-1] if j > 0 else float('-inf')
```

```
        rightB = b[j]   if j < n else float('inf')

        if leftA <= rightB and leftB <= rightA:
            # Correct partition found
            if total % 2 == 1:
                return min(rightA, rightB)
            return (max(leftA, leftB) + min(rightA, rightB)) / 2
        elif leftA > rightB:
            hi = i - 1
        else:
            lo = i + 1

# Example
print(find_median_sorted_arrays([1,3], [2]))       # 2.0
print(find_median_sorted_arrays([1,2], [3,4]))     # 2.5
print(find_median_sorted_arrays([], [1]))          # 1.0
```

Complexity: O(log min(m, n)) time and O(1) space.

Comparison Summary

Aspect	Merge Method	Binary Search Method
Time	O(m + n)	O(log■min(m,■n))
Space	O(m + n)	O(1)
Use Case	Simpler, small data	Large data, interviews