

■ ■ Divide Two Integers

Bit Manipulation • Medium (Bit-Shift Division)

■ The Hook

Divide two integers without using multiplication, division, or mod operators. This problem appears in interviews because it tests whether you can implement a primitive operation from scratch. The key insight: division is repeated subtraction, and bit shifts let you do that subtraction in $O(\log^2 n)$ instead of $O(n)$.

■ Core Intuition — Bit Shift Doubling

Instead of subtracting divisor one at a time, double it using left shifts. Find the largest multiple of divisor (by doubling) that fits in dividend. Subtract that multiple, record how many divisors it represents, repeat. This is binary long division.

- Example: $43 / 3$
- $3 \ll 3 = 24$ (8×3), fits in 43. Subtract 24, quotient += 8. Remainder = 19.
- $3 \ll 2 = 12$ (4×3), fits in 19. Subtract 12, quotient += 4. Remainder = 7.
- $3 \ll 1 = 6$ (2×3), fits in 7. Subtract 6, quotient += 2. Remainder = 1.
- $3 \ll 0 = 3$ (1×3), doesn't fit in 1. Done.
- Result: $8+4+2 = 14$. Verify: $14 \times 3 = 42$, remainder 1 ✓

■ ■ Edge Cases (Critical for Interviews)

- $\text{INT_MIN} / -1$ overflows → clamp to INT_MAX
- Determine sign before working with absolute values
- Use long to avoid overflow during the doubling step

■ Java Implementation

```
public class DivideTwoIntegers {
    public static int divide(int dividend, int divisor) {
        // Edge case: overflow
        if (dividend == Integer.MIN_VALUE && divisor == -1)
            return Integer.MAX_VALUE;
    }
}
```

```

// Determine sign
int sign = (dividend > 0) == (divisor > 0) ? 1 : -1;

// Work with absolute values using long
long dvd = Math.abs((long) dividend);
long dvs = Math.abs((long) divisor);
int result = 0;

while (dvd >= dvs) {
    long temp = dvs, multiple = 1;
    // Double until temp > dvd
    while (dvd >= (temp << 1)) {
        temp <<= 1;
        multiple <<= 1;
    }
    dvd -= temp;
    result += multiple;
}

return sign == 1 ? result : -result;
}
}

```

■ ■ Complexity

Case	Time	Space	Why
Time	$O(\log^2 n)$	$O(1)$	Outer loop: $O(\log n)$ steps; inner doubling: $O(\log n)$ each
Space	$O(1)$	$O(1)$	Only a constant number of variables

■ ■ Common Mistakes

- Not casting to long before Math.abs — Integer.MIN_VALUE overflows.
- Inner loop condition: `dvd >= (temp << 1)` needs parentheses and overflow check.
- Forgetting that `temp << 1` can itself overflow if temp is already large.

■ Practice Problems

Problem	Difficulty
---------	------------

Divide Two Integers (LeetCode 29)	Medium
Pow(x, n) (LeetCode 50)	Medium
Sqrt(x) (LeetCode 69)	Easy
Super Pow (LeetCode 372)	Medium

■ Quick Revision

- Division = repeated subtraction; use bit shifts to subtract large chunks
- Double divisor using left shifts to find the largest fitting multiple
- Handle sign separately; work with long to avoid overflow
- INT_MIN / -1 is the one overflow edge case to guard against

■ Mental Model

Think of it like fitting the largest possible LEGO block that fits, then the next largest, then next — until nothing fits. Left shifts double the block size each step. Count how many blocks you used. That's your quotient.