Bit Manupilation

**Java Code:**

java

Copy code

public class DivideTwoIntegers {

public int divide(int dividend, int divisor) {

// Handle overflow

if (dividend == Integer.MIN\_VALUE && divisor == -1) {

return Integer.MAX\_VALUE;

}

// Determine the sign of the result

boolean negative = (dividend < 0) != (divisor < 0);

// Work with positive values

long absDividend = Math.abs((long) dividend);

long absDivisor = Math.abs((long) divisor);

// Initialize quotient

int quotient = 0;

// Subtract divisor from dividend using bit manipulation

for (int i = 31; i >= 0; i--) {

if ((absDividend >> i) >= absDivisor) {

quotient += 1 << i;

absDividend -= absDivisor << i;

}

}

// Apply the sign

return negative ? -quotient : quotient;

}

public static void main(String[] args) {

DivideTwoIntegers solution = new DivideTwoIntegers();

// Example usage:

System.out.println(solution.divide(10, 3)); // Output: 3

System.out.println(solution.divide(7, -3)); // Output: -2

}

}

**Explanation:**

1. **Overflow Handling:**
   * The condition if (dividend == Integer.MIN\_VALUE && divisor == -1) handles the edge case where dividing Integer.MIN\_VALUE by -1 would cause an overflow. In this case, we return Integer.MAX\_VALUE.
2. **Determine the Sign:**
   * The result should be negative if the dividend and divisor have opposite signs. The XOR operation (!=) determines this.
3. **Convert to Positive Values:**
   * We convert the dividend and divisor to their absolute values using Math.abs(). This avoids dealing with negative numbers directly in the bitwise operations.
4. **Bitwise Operations:**
   * The for loop iterates from 31 down to 0, checking if the shifted divisor can be subtracted from the dividend.
   * If (absDividend >> i) >= absDivisor, then divisor << i can be subtracted from dividend. We increment the quotient by 1 << i and subtract divisor << i from dividend.
5. **Return the Quotient:**
   * The sign of the quotient is applied by checking the negative flag.

This Java implementation efficiently handles the division of two integers, adhering to the constraints given, and ensuring the quotient is within the 32-bit signed integer range.