🔢 Numbers Interview Quick-Check Pattern

Your fast mental model and pattern recognition system for tackling number-related problems in interviews.

---

1. 🧠 When is a Number Problem Tricky?

Ask yourself:

- ❓ Is this number **too large** to store? (overflow/BigInt)
- ❓ Are we working with **digits** or the number as a **whole**?
- ❓ Are we checking for **patterns**, **divisibility**, **primes**, **squares**?
- ❓ Does the problem ask for **steps**, **count**, or **optimization**?
- ❓ Can we use **math formulas** instead of brute force?

> 💡 Number problems often hide patterns or mathematical simplifications.

---

2. 🔍 Core Patterns & Techniques

Technique	Common Use Cases
Modulo (%)	Cycle detection, divisibility, parity
Integer Division	Digit stripping, compression
Math.floor / Math.ceil	Rounding bounds
Prime Sieve	Count primes efficiently (O(n log log n))
GCD / LCM	Ratios, fractions, simplification
Bit Manipulation	Power of two, XOR for single numbers
Math Formulas	Triangular numbers, digit sum

Binary Search	Square root, integer power, guessing game

---

3. 🧪 Must-Know Templates

✅ Reverse an Integer (handle overflow)

```
function reverse(x) {
  let rev = 0;
  const sign = x < 0 ? -1 : 1;
  x = Math.abs(x);

  while (x > 0) {
    rev = rev * 10 + (x % 10);
    x = Math.floor(x / 10);
  }

  rev *= sign;
  if (rev < -(2**31) || rev > 2**31 - 1) return 0;
  return rev;
}
```

✅ Check Palindrome Number

```
function isPalindrome(x) {
  if (x < 0) return false;
  let str = x.toString();
  return str === str.split('').reverse().join('');
}
```

✅ Count Digits or Sum of Digits

```
function digitSum(n) {
```

```
  let sum = 0;
  while (n > 0) {
    sum += n % 10;
    n = Math.floor(n / 10);
  }
  return sum;
}
```

✅ Prime Number Check

```
function isPrime(n) {
  if (n < 2) return false;
  for (let i = 2; i * i <= n; i++) {
    if (n % i === 0) return false;
  }
  return true;
}
```

✅ Count Primes (Sieve of Eratosthenes)

```
function countPrimes(n) {
  const isPrime = new Array(n).fill(true);
  isPrime[0] = isPrime[1] = false;

  for (let i = 2; i * i < n; i++) {
    if (isPrime[i]) {
      for (let j = i * i; j < n; j += i) {
        isPrime[j] = false;
      }
    }
  }

  return isPrime.filter(Boolean).length;
}
```

✅ Single Number Using XOR

```
function singleNumber(nums) {
  let result = 0;
  for (let num of nums) {
    result ^= num;
  }
  return result;
}
```

---

4. 🧱 Edge Cases You Should Watch

- Negative numbers
- Large input (overflow risk!)
- 0 and 1 as special values
- Floating point precision
- Division by zero
- Integer rounding issues (`Math.floor`, `Math.ceil`)
- Infinite loop from incorrect `while` math

> 🧠 Think: What **weird value** would break this?

---

5. 🧠 Mental Model to Categorize Number Problems

All number problems fall into these 5 types:

Type	Trigger Words	Key Techniques
Digit-based	Reverse, Sum of digits, Palindrome	`%`, `/`, string conversion
Prime/Divisor	Prime check, Count primes, GCD	Sieve, GCD, Modulo
Math Trick	Triangular, Power of 2, Factorial	Math formulas, Bitwise, Log

Pattern Count	Steps, Number of ways, XOR tricks	Dynamic programming, Bit math
Transform	Remove digits, Encode, Shift	While loop, digit manipulation

---

🔁 Problem Solving Loop

1. 🔍 What are you being asked: **Check, Count, Transform, Reverse**?
2. 🔣 Work with **number as integer** or **as string**?
3. 🔢 Can you simplify using **modulo**, **division**, or **XOR**?
4. 🧪 What's the **math shortcut** here?
5. 🚨 What are the **edge cases**?

✅ Final Interview Checklist

- Is it a **digit manipulation** or **whole number** problem?
- Did I consider negative numbers and 0?
- Could this **overflow** or run into precision issues?
- Can I apply **Math tricks** (log, floor, XOR)?
- Did I miss any **special values** or **divisibility edge cases**?