

■■ Check if i-th Bit is Set

Bit Manipulation • Easy

■ The Hook

Imagine a row of 32 light switches, numbered 0 to 31. Each switch maps to one feature in a settings panel. To know if feature #5 is enabled, you don't scan all switches — you jump directly to switch 5 and check it. That is exactly what 'check the i-th bit' does. This trick powers permission flags in Linux, graphics bitmasks, and network headers.

■ Core Intuition

To isolate bit i , shift the number 1 left by i positions. This creates a mask with only bit i set. AND that mask with n . If the result is non-zero, bit i was 1 in n .

■ $i = 2, n = 13$ (1101)

■ $\text{mask} = 1 \ll 2 = 0100$

■ $13 \& 4 = 1101 \& 0100 = 0100 = 4 \neq 0 \rightarrow \text{bit 2 is SET}$

■■ How It Works — Two Approaches

Approach 1: Left-shift the mask

```
(n & (1 << i)) != 0
```

Approach 2: Right-shift n , then check bit 0

```
((n >> i) & 1) == 1
```

Both work. Approach 2 is slightly more readable because it reduces to the odd/even check after shifting.

■ Dry Run — $n=13, i=2$

■ $n = 13 = 1101$ in binary

■ $1 \ll 2 = 0100 = 4$

■ $13 \& 4 = 0100 = 4 \rightarrow \text{non-zero} \rightarrow \text{bit 2 IS set} \checkmark$

Now $i=1$:

■ $1 \ll 1 = 0010 = 2$

■ $13 \& 2 = 1101 \& 0010 = 0000 = 0 \rightarrow$ bit 1 is NOT set ✓

■ Java Implementation

```
public class BitCheck {  
    // Returns true if bit at position i (0-indexed) is 1  
    public static boolean isBitSet(int n, int i) {  
        return (n & (1 << i)) != 0;  
    }  
  
    // Alternative: shift right, check last bit  
    public static boolean isBitSet2(int n, int i) {  
        return ((n >> i) & 1) == 1;  
    }  
  
    public static void main(String[] args) {  
        int n = 13; // binary: 1101  
        for (int i = 0; i < 4; i++) {  
            System.out.println("Bit " + i + ": " + isBitSet(n, i));  
        }  
        // Bit 0: true, Bit 1: false, Bit 2: true, Bit 3: true  
    }  
}
```

■ Complexity

Case	Time	Space	Why
All cases	$O(1)$	$O(1)$	Shift + AND are single CPU instructions

■ Common Mistakes

- Shifting by $i \geq 32$ on an int causes undefined behavior — use long for large i.
- Using 1 instead of 1L for $i > 30$ on long numbers.

■ Patterns & Connections

- Setting a bit: $n | (1 \ll i)$
- Clearing a bit: $n \& \sim(1 \ll i)$
- Toggling a bit: $n \wedge (1 \ll i)$

- Basis for: counting set bits, subset enumeration, bitmask DP

■ Practice Problems

Problem	Difficulty
Check if i-th bit is set	Easy
Set / Clear / Toggle a bit	Easy
Find position of only set bit	Easy
Reverse bits of a 32-bit integer (LeetCode 190)	Medium

■ Quick Revision

- Check bit i: $(n \gg i) \& 1$
- Set bit i: $n \mid (1 \ll i)$
- Clear bit i: $n \& \sim(1 \ll i)$
- Toggle bit i: $n \wedge (1 \ll i)$

■ Mental Model

Think of $1 \ll i$ as a spotlight aimed at position i. AND-ing with n tells you: is there something at that exact spot? Shift once, AND once, answer found.