

Maximum XOR of Two Numbers

Binary Trie for XOR optimization $\oplus$

● MEDIUM

■ Hook

Find the maximum XOR of any two numbers in an array. Brute force: $O(n^2)$. With a Binary Trie (bits instead of chars): $O(32n)$. For each number, greedily pick the opposite bit at each level to maximize XOR.

■ Core Intuition — Greedy Bit Selection

■ XOR maximization = choose opposite bits

XOR is 1 when bits differ. So for each bit of a number (MSB to LSB), greedily try to find a number in the Trie that has the OPPOSITE bit. If possible $\rightarrow$ XOR bit = 1 (better). If not $\rightarrow$ take same bit, XOR bit = 0.

■ Visual — [3,10,5,25,2,8]

```
Binary Trie (32-bit, showing relevant bits for 25=11001):  
Insert all numbers in binary.  
For num=25 (11001), find max XOR partner:  
Bit 4: 25 has '1'  $\rightarrow$  want '0'  $\rightarrow$  go left if exists  
Bit 3: 25 has '1'  $\rightarrow$  want '0'  $\rightarrow$  found! XOR bit=1  
...continue for all 32 bits...  
25 XOR 5 = 11001 XOR 00101 = 11100 = 28  
★ ANSWER = 28
```

■ Code

```

public int findMaximumXOR(int[] nums) {
    int max=0, mask=0;
    // Approach 2: Greedy bit masking O(32n) without Trie
    for (int i=31; i>=0; i--) {
        mask |= (1<<i);
        Set<Integer> prefixes = new HashSet<>();
        for (int n : nums) prefixes.add(n & mask);
        int temp = max | (1<<i);
        for (int p : prefixes)
            if (prefixes.contains(temp ^ p)) { max=temp; break; }
    }
    return max;
}
// Trie approach: insert all, then query each
// Each node has children[0] and children[1] for bits

```

■ TIME

$O(32n) = O(n)$

■ SPACE

$O(32n) = O(n)$

- ✓ Binary Trie: each node has 2 children (bit 0, bit 1)
- ✓ Insert all numbers MSB to LSB
- ✓ Query: for each bit, try opposite direction for max XOR
- ✓ Greedy works because higher bits contribute more to XOR value
- ✓ Time: $O(32n) = O(n)$ | Space: $O(32n) = O(n)$

■ MENTAL MODEL — Remember This Forever

To maximize XOR, be greedy from the most significant bit. XOR 1 > XOR 0 for the same bit position. The Trie lets you instantly check: 'does any number have the opposite bit here?' Go opposite whenever possible — that's the greedy XOR maximization strategy.