Bit Manipulation questi

Basic Level
Check if a Number is Even or Odd
Write a function to determine if a number is even or odd using bit manipulation.

Count the Number of Set Bits
Count the number of set bits (1s) in the binary representation of an integer.

Find the Unique Element in an Array
Given an array where every element appears twice except for one, find the unique element.

Find the Rightmost Set Bit
Given an integer, find the position of the rightmost set bit.

Swap Two Numbers
Swap two integers without using a temporary variable, using bitwise XOR.

Check if a Number is a Power of Two
Determine if a given integer is a power of two using bit manipulation.

Toggle Bits in a Number
Given a number and a mask, toggle the bits of the number where the mask has bits set.

Find the Bitwise AND of Range
Find the bitwise AND of all numbers in a given range [m, n].

Find the Bitwise OR of Range
Find the bitwise OR of all numbers in a given range [m, n].

Clear the Last Set Bit
Write a function to clear the last set bit of an integer.

give me explanation of problem statement with example use in simple way and also give java code optimized and easy to understand and also give time complexity of individual

Memory updated

Here are the problem explanations, examples, Java code, and time complexities for the bit manipulation techniques:

---

1. Check if a Number is Even or Odd using Bit Manipulation

Explanation:

A number is even if its least significant bit (LSB) is 0, and odd if the LSB is 1. We can use the bitwise AND (`&`) operation to check this. The expression `num & 1` will give 1 if the number is odd and 0 if the number is even.

Example:

- Input: `5`
- Output: Odd (because `5 & 1 = 1`)
- Input: `8`
- Output: Even (because `8 & 1 = 0`)

Code:

```java
public class BitManipulation {
    public static String isEvenOrOdd(int num) {
        return (num & 1) == 0 ? "Even" : "Odd";
    }
}
```

Time Complexity: `O(1)` **(constant time)**

---

2. Count the Number of Set Bits

Explanation:

The set bits of a number are the number of `1s` in its binary representation. We can count these bits by repeatedly performing `num & (num - 1)`, which clears the rightmost set bit.

Example:

- Input: `5` (binary: `101`)
- Output: `2` (because 5 has two 1s)

Code:

```java
public class BitManipulation {
    public static int countSetBits(int num) {
        int count = 0;
        while (num > 0) {
            num &= (num - 1);
            count++;
        }
        return count;
    }
}
```

Time Complexity: `O(k)` where `k` is the number of set bits

---

3. Find the Unique Element in an Array

Explanation:

In an array where every element appears twice except for one, the XOR (`^`) of all elements will give the unique element since `a ^ a = 0` and `a ^ 0 = a`.

Example:

- Input: `[2, 2, 3, 4, 4]`
- Output: `3`

Code:

```java
public class BitManipulation {
    public static int findUnique(int[] arr) {
        int unique = 0;
        for (int num : arr) {
            unique ^= num;
        }
        return unique;
    }
}
```

Time Complexity: `O(n)` where `n` is the length of the array

---

4. Find the Rightmost Set Bit

Explanation:

To find the position of the rightmost set bit, we use the expression `num & -num`, which isolates the lowest set bit.

Example:

- Input: `18` (binary: `10010`)
- Output: `2` (position of the rightmost set bit)

Code:

```java
public class BitManipulation {
    public static int rightmostSetBit(int num) {
        return num & -num;
```

```
    }
}
```

Time Complexity: `O(1)`

---

5. Swap Two Numbers Using XOR

Explanation:

We can swap two numbers without using a temporary variable using XOR. This works because XOR-ing a number with itself results in `0`.

Example:

- Input: `a = 3`, `b = 4`
- Output: `a = 4`, `b = 3`

Code:

```java
public class BitManipulation {
    public static void swapNumbers(int a, int b) {
        a = a ^ b;
        b = a ^ b;
        a = a ^ b;
        System.out.println("a = " + a + ", b = " + b);
    }
}
```

Time Complexity: `O(1)`

---

6. Check if a Number is a Power of Two

Explanation:

A number is a power of two if it has only one set bit. The expression `num & (num - 1)` will be `0` for powers of two.

Example:

- Input: `8`
- Output: `True` (because `8 = 2^3`)

Code:

```java
public class BitManipulation {
    public static boolean isPowerOfTwo(int num) {
        return num > 0 && (num & (num - 1)) == 0;
    }
}
```

Time Complexity: `O(1)`

---

7. Toggle Bits in a Number

Explanation:

To toggle bits in a number, use the XOR operation with a mask where the mask specifies the bits to toggle.

Example:

- Input: `num = 5`, `mask = 3` (binary of 5: `101`, binary of 3: `011`)
- Output: `6` (binary: `110`)

Code:

```java
public class BitManipulation {
    public static int toggleBits(int num, int mask) {
        return num ^ mask;
    }
}
```

Time Complexity: `O(1)`

---

8. Find the Bitwise AND of Range

Explanation:

For a range `[m, n]`, the bitwise AND removes the varying lower bits. The result is the common prefix of `m` and `n`.

Example:

- Input: `m = 5`, `n = 7`
- Output: `4`

Code:

```java
public class BitManipulation {
    public static int rangeBitwiseAnd(int m, int n) {
        while (m < n) {
            n &= (n - 1);
        }
        return m & n;
    }
}
```

Time Complexity: `O(log(n))`

---

9. Find the Bitwise OR of Range

Explanation:

For a range `[m, n]`, the bitwise OR propagates the changes in bits across the range.

Example:

- Input: `m = 5`, `n = 6`
- Output: `7`

Code:

```java
public class BitManipulation {
    public static int rangeBitwiseOr(int m, int n) {
        while (m < n) {
            m |= (m + 1);
        }
        return m;
    }
}
```

Time Complexity: `O(log(n))`

---

10. Clear the Last Set Bit

Explanation:

To clear the last set bit of a number, use the expression `num & (num - 1)`, which removes the rightmost set bit.

Example:

- Input: `12` (binary: `1100`)
- Output: `8` (binary: `1000`)

Code:

```java
public class BitManipulation {
    public static int clearLastSetBit(int num) {
        return num & (num - 1);
    }
}
```

Time Complexity: `O(1)`

Each of these problems involves efficient bit manipulation, ensuring optimal time complexity for the operations.

Intermediate Level
Find the Single Non-Duplicate Element
Given an array where every element appears thrice except for one, find the element that appears only once.

Find Missing Number in Array
Given an array containing n distinct numbers taken from 0 to n, find the missing number.

Count the Number of Bits Required to Convert A to B
Given two integers A and B, find the number of bits required to convert A to B.

Find the Number with All Bits Set
Given a number of bits, find the number where all bits are set.

Find the Minimum Number of Flips to Convert A to B
Given two integers A and B, find the minimum number of bit flips required to convert A to B.

Determine the Number of Distinct Substrings
Find the number of distinct substrings in a binary string.

Generate All Subsets
Given a set, generate all possible subsets using bit manipulation.

Check if a Binary String has Alternating Bits
Check if a binary string contains alternating bits (e.g., 101010).

Find the Maximum XOR of Two Numbers in an Array
Given an array of integers, find the maximum XOR value of any two elements.

Find the Bitwise XOR of Range
Find the bitwise XOR of all numbers in a given range [m, n].

Memory updated

Here are the intermediate-level bit manipulation problems with explanations, examples, Java code, and time complexities:

---

1. Find the Single Non-Duplicate Element

Explanation:

Given an array where every element appears thrice except for one, use bitwise operations to isolate the element that appears once. We can count bits at each position and take the modulo 3 to ignore elements that appear thrice.

Example:

- Input: `[2, 2, 2, 5]`
- Output: `5`

Code:

```java
public class BitManipulation {
    public static int findSingleNonDuplicate(int[] arr) {
        int ones = 0, twos = 0;
        for (int num : arr) {
            ones = (ones ^ num) & ~twos;
            twos = (twos ^ num) & ~ones;
        }
        return ones;
    }
}
```

Time Complexity: `O(n)`

---

2. Find Missing Number in Array

Explanation:

Given an array containing `n` distinct numbers taken from `0` to `n`, find the missing number using XOR. The XOR of all indices from `0` to `n` and all elements will result in the missing number.

Example:

- Input: `[3, 0, 1]`
- Output: `2`

Code:

```java
public class BitManipulation {
    public static int findMissingNumber(int[] arr, int n) {
        int xor = 0;
        for (int i = 0; i <= n; i++) xor ^= i;
        for (int num : arr) xor ^= num;
        return xor;
    }
}
```

Time Complexity: `O(n)`

---

3. Count the Number of Bits Required to Convert A to B

Explanation:

To find the number of bits required to convert A to B, XOR the two numbers and count the set bits.

Example:

- Input: `A = 29`, `B = 15` (binary: `11101` and `01111`)
- Output: `2`

Code:

```java
public class BitManipulation {
    public static int countBitsToConvert(int a, int b) {
        int xor = a ^ b, count = 0;
        while (xor > 0) {
            xor &= (xor - 1);
            count++;
        }
        return count;
    }
}
```

Time Complexity: `O(k)` where `k` is the number of set bits in `A ^ B`

---

4. Find the Number with All Bits Set

Explanation:

Given a number of bits, find the number where all bits are set. This can be done using the formula `(1 << bits) - 1`.

Example:

- Input: `bits = 3`
- Output: `7` (binary: `111`)

Code:

```java
public class BitManipulation {
    public static int allBitsSet(int bits) {
        return (1 << bits) - 1;
    }
}
```

Time Complexity: `O(1)`

---

5. Find the Minimum Number of Flips to Convert A to B

Explanation:

To convert A to B, find the XOR of A and B, then count the set bits in the result, which represent the positions that need to be flipped.

Example:

- Input: `A = 29`, `B = 15`
- Output: `2`

Code:

```java
public class BitManipulation {
    public static int minFlipsToConvert(int a, int b) {
        return countBitsToConvert(a, b); // Using the previous function
    }
}
```

Time Complexity: `O(k)`

---

6. Determine the Number of Distinct Substrings

Explanation:

For a binary string, the distinct substrings can be found using bit manipulation. Each position in the string can represent a substring, and unique substrings can be tracked with a set.

Example:

- Input: `binaryString = "101"`
- Output: The number of distinct substrings.

Code:

```java
// This is typically done with tries or hashing; bit manipulation for this task is non-trivial
```

Time Complexity: `O(n^2)`

---

7. Generate All Subsets

Explanation:

Using bit manipulation, each subset of a set corresponds to a unique binary number. If the ith bit is set, include the ith element in the subset.

Example:

- Input: `[1, 2, 3]`
- Output: All subsets (e.g., `[], [1], [2], [1, 2], [3], [1, 3], [2, 3], [1, 2, 3]`)

Code:

```java
public class BitManipulation {
    public static void generateSubsets(int[] arr) {
        int n = arr.length;
        for (int i = 0; i < (1 << n); i++) {
            System.out.print("{ ");
            for (int j = 0; j < n; j++) {
                if ((i & (1 << j)) != 0) {
                    System.out.print(arr[j] + " ");
                }
            }
            System.out.println("}");
        }
    }
}
```

Time Complexity: `O(2^n)`

---

8. Check if a Binary String has Alternating Bits

Explanation:

A binary string has alternating bits if there is no consecutive sequence of `0`s or `1`s. We can use the XOR of the number and its right-shifted value to check this.

Example:

- Input: `binaryString = "101010"`
- Output: True

Code:

```java
public class BitManipulation {
    public static boolean hasAlternatingBits(int n) {
        int xor = n ^ (n >> 1);
        return (xor & (xor + 1)) == 0;
    }
}
```

Time Complexity: `O(1)`

---

9. Find the Maximum XOR of Two Numbers in an Array

Explanation:

We can use a trie to efficiently find the maximum XOR of two numbers in an array, but with bit manipulation, we check pairs directly.

Example:

- Input: `[3, 10, 5, 25, 2, 8]`
- Output: Maximum XOR value

Code:

```java
public class BitManipulation {
    public static int findMaximumXOR(int[] nums) {
        int max = 0, mask = 0;
        for (int i = 31; i >= 0; i--) {
            mask |= (1 << i);
            Set<Integer> prefixes = new HashSet<>();
            for (int num : nums) {
                prefixes.add(num & mask);
            }
            int candidate = max | (1 << i);
            for (int prefix : prefixes) {
                if (prefixes.contains(prefix ^ candidate)) {
                    max = candidate;
                    break;
                }
            }
        }
        return max;
```

```
    }
}
```

Time Complexity: `O(n)`

---

10. Find the Bitwise XOR of Range

Explanation:

The XOR of a range `[m, n]` can be found by leveraging the properties of XOR, where the XOR from `0` to `n` follows a pattern.

Example:

- Input: `m = 3`, `n = 5`
- Output: XOR of numbers in the range

Code:

```java
public class BitManipulation {
    public static int xorRange(int m, int n) {
        return xorUpto(n) ^ xorUpto(m - 1);
    }

    private static int xorUpto(int num) {
        if (num % 4 == 0) return num;
        else if (num % 4 == 1) return 1;
        else if (num % 4 == 2) return num + 1;
        else return 0;
    }
}
```

Time Complexity: `O(1)`

Each of these intermediate-level problems is optimized for bit manipulation with efficient algorithms and low time complexity.