- **How to create a mask with trailing `n` bits of all 1's:**

  - **Goal:** Create a mask that looks like `...000111` where the rightmost `n` bits are 1 and all bits to the left are 0.
  - **Approach:**
    - Start with the integer value 1. This has the bit pattern `...0001`.
    - Shift this value left `n` times. This results in the bit pattern `...1000`, where the single '1' bit is now at position `n`.
    - Subtract 1 from this result. Subtracting 1 from a number with a single '1' followed by zeros (`...1000`) flips the '1' to a '0' and all the trailing '0's to '1's (`...0111`). This produces a mask with exactly `n` trailing ones.
    - Handle the edge case where `n` is 0; the mask should be 0.

- **How to create a mask with leading `n` bits of 1's:**

  - **Goal:** Create a mask that looks like `111...1000` where the leftmost (most significant) `n` bits are 1 and all bits to the right are 0. This mask depends on the total number of bits in the integer type (e.g., 32 bits for an `int`).
  - **Approach:**
    - Create a mask that has *all* bits set to 1. This can often be done by using a bitwise NOT operation on 0 (e.g., `~0`).
    - Shift this all-ones mask left by `n` positions. This operation effectively pushes `n` zeros into the rightmost positions, while the higher bits remain 1s (or are shifted off the left end, depending on the exact value and type, but the desired mask effect is achieved for the lower bits). The result is a mask with the rightmost `n` bits being 0 and all bits to their left being 1. These higher bits represent the "leading" ones.
    - Handle the edge case where `n` is 0; the mask should be all ones. Handle the edge case where `n` is equal to the total number of bits; the mask should be 0. (Note: Shifting an all-ones mask left by the number of bits in the type typically results in 0).

- **How to create a mask with starting position `p` set `n` bits of 1:**

  - **Goal:** Create a mask with `n` consecutive 1s, where the most significant of these `n` bits is at position `p`, and the least significant is at position `p - n + 1`. All other bits are 0. The mask looks like `...011...10...0`, with ones spanning from position `p` down to `p - n + 1`.
  - **Approach:**
    - First, create a mask with `n` trailing ones (using the approach from step 1). This gives you the block of `n` ones `...000111` at the rightmost end.
    - You need to shift this block of `n` ones to the left so that its rightmost bit (which is currently at position 0) ends up at position `p - n + 1`.
    - The number of positions to shift left is calculated as `(p - n + 1)`.
    - Shift the mask with `n` trailing ones left by this calculated amount. This places the block of `n` ones exactly in the positions from `p - n + 1` up to `p`.
    - Handle the edge case where `n` is 0; the mask should be 0. Assume `p >= n - 1` for a valid range of bit positions.

- **Question:** How to clear a specific bit at position `i` in an integer `x`?
    - **Approach:** Create a mask that has a 0 at position `i` and 1s everywhere else. Perform a bitwise AND operation between `x` and this mask.
- **Question:** How to toggle (flip) a specific bit at position `i` in an integer `x`?
    - **Approach:** Create a mask that has a 1 at position `i` and 0s everywhere else. Perform a bitwise XOR operation between `x` and this mask.
- **Question:** How to check if a specific bit at position `i` is set (equal to 1) in an integer `x`?
    - **Approach:** Create a mask that has a 1 at position `i` and 0s everywhere else. Perform a bitwise AND operation between `x` and this mask. If the result is non-zero, the bit was set.
- **Question:** How to extract the value (0 or 1) of a bit at position `i` in an integer `x`?
    - **Approach:** Shift `x` right by `i` positions to move the bit in question to the rightmost position. Then, perform a bitwise AND operation with 1 to isolate this single bit's value.
- **Question:** How to set `n` bits in `x` starting at position `p` to 1?
    - **Approach:** Create a mask that has `n` consecutive 1s starting at position `p` (and 0s elsewhere). Perform a bitwise OR operation between `x` and this mask. (Hint: You can create the mask by generating `n` trailing ones and shifting them into the correct position).
- **Question:** How to clear `n` bits in `x` starting at position `p`?
    - **Approach:** Create a mask that has `n` consecutive 0s starting at position `p` (and 1s elsewhere). Perform a bitwise AND operation between `x` and this mask. (Hint: You can create this mask by inverting the mask from Question 5).
- **Question:** How to toggle `n` bits in `x` starting at position `p`?
    - **Approach:** Create a mask that has `n` consecutive 1s starting at position `p` (and 0s elsewhere). Perform a bitwise XOR operation between `x` and this mask. (Hint: Use the mask from Question 5).
- **Question:** How to extract the `n` bits from `x` starting at position `p` and return them as the rightmost bits of a new number? (e.g., if x = ...10110110 and p=5, n=3, extract 101 and return ...00000101).
    - **Approach:** Create a mask that has `n` consecutive 1s starting at position `p`. Perform a bitwise AND operation between `x` and this mask to isolate the desired bits. Then, shift the result right by `p - n + 1` positions to move the extracted bits to the rightmost positions.

- **Question:** How to rotate the bits of an integer `x` left by `k` positions? (Bits shifted off the left re-appear on the right). Assume a fixed integer size, say `SIZE`.

    - **Approach:** You need to handle the bits that "wrap around". Extract the leftmost `k` bits of `x`. Shift `x` left by `k` positions. Shift the extracted leftmost `k` bits right by `SIZE - k` positions so they are in the rightmost positions. Combine the shifted `x` and the shifted extracted bits using a bitwise OR operation. Handle edge cases where `k` is 0 or a multiple of `SIZE`.

- **Question:** How to rotate the bits of an integer `x` right by `k` positions? (Bits shifted off the right re-appear on the left). Assume a fixed integer size, say `SIZE`.

    - **Approach:** Similar to left rotation, but the other direction. Extract the rightmost `k` bits of `x`. Shift `x` right by `k` positions. Shift the extracted rightmost `k` bits left by `SIZE - k` positions so they are in the leftmost positions. Combine the shifted `x` and the shifted extracted bits using a bitwise OR operation. Handle edge cases where `k` is 0 or a multiple of `SIZE`.

- **Question:** How to count the number of set bits (bits with value 1) in an integer `x` (also known as population count or Hamming weight)?

    - **Approach:** One efficient bitwise approach (Brian Kernighan's algorithm) involves repeatedly clearing the least significant set bit of the number and incrementing a counter until the number becomes zero. Clearing the least significant set bit can be done using the operation `x = x & (x - 1)`.

- **Question:** How to check if a positive integer `x` is a power of 2 using bitwise operations?

    - **Approach:** A positive integer is a power of 2 if and only if it has exactly one bit set in its binary representation. Consider the bitwise AND of `x` and `x - 1`. If `x` is a power of 2, what does `x - 1` look like in binary, and what is the result of `x & (x - 1)`? Also, ensure `x` is greater than 0.

- **Question:** How to find the position of the least significant set bit in a non-zero integer `x`? (Return -1 if `x` is 0).

    - **Approach:** You can isolate the least significant set bit using a bitwise operation (e.g., `x & -x` for two's complement systems). Once you have a number with only the least significant bit of `x` set, you need to determine its position. This can be done by checking bit positions iteratively with shifts or by using mathematical functions like logarithm (though bitwise methods are often preferred). One way is to repeatedly right-shift the isolated bit until it becomes 1, counting the shifts.

- **Question:** How to clear the least significant set bit in a non-zero integer `x`?

    - **Approach:** Consider the bitwise AND of `x` and `x - 1`. Think about how subtracting 1 affects the least significant set bit and all the trailing zeros in a binary number.

- **Question:** How to swap two integers `a` and `b` using bitwise XOR operations without using a temporary variable? (XOR swap algorithm).

    - **Approach:** This classic trick uses three XOR operations. The first operation stores information about both `a` and `b` in `a`. The second operation uses the new value of `a` and the original value of `b` to derive the original value of `a` and store it in `b`. The third operation uses the new value of `b` (which is the original `a`) and the current value of `a` (which holds a combination) to derive the original value of `b` and store it in `a`.

- **Question:** How to check if two integers `a` and `b` have opposite signs using bitwise operations, without using comparison operators (<, >, ==)?

    - **Approach:** In two's complement representation, the most significant bit indicates the sign (0 for positive, 1 for negative). Consider the bitwise XOR of `a` and `b`. If `a` and `b` have opposite signs, what will their most significant bits be, and what will the most significant bit of their XOR result be? You need to check if this specific bit is set.

- **Question:** How to reverse the order of all bits in an integer `x`? (e.g., if x is 8 bits, bit 7 becomes bit 0, bit 6 becomes bit 1, etc.). Assume a fixed integer size, say `SIZE`.

    - **Approach:** This can be done efficiently by repeatedly swapping pairs of bits at symmetric positions from the ends, moving inwards. A more optimized approach involves a sequence of masking and shifting operations that swap groups of bits: first swap adjacent bits, then swap adjacent 2-bit groups, then adjacent 4-bit groups, and so on, until you swap the two halves of the integer. This requires `log2(SIZE)` passes.

- **Question:** How to set `n` bits in `x` starting at position `p` to the value of a single bit `b` (where `b` is either 0 or 1)?

    - **Approach:** This combines clearing and setting. First, clear the target `n` bits in `x` (using the approach from Question 6). Then, if `b` is 1, create a mask with `n` ones starting at position `p` and OR it with the cleared `x`. If `b` is 0, the bits are already cleared, so no further action is needed on the cleared `x`.

- **Question:** How to create a mask that has ones at specific, non-consecutive positions `i` and `j` (and 0s elsewhere)?

    - **Approach:** Create a mask that has only the bit at position `i` set to 1. Create another mask that has only the bit at position `j` set to 1. Combine these two masks using a bitwise OR operation.

- **Question:** Given two integers `x` and `y`, how to find the number of bit positions at which their corresponding bits are different? (This is the Hamming distance for binary strings).

    - **Approach:** The bitwise XOR operation (`x ^ y`) results in a number where a bit is set (1) if and only if the corresponding bits in `x` and `y` were different. Therefore, the problem reduces to counting the number of set bits in the result of `x ^ y`. Use the approach from Question 11 to count the set bits in `x ^ y`.

- **Question:** How to check if all bits within a specific range (defined by `n` bits starting at position `p`) are set to 1 in an integer `x`?

    - **Approach:** Create a mask that has `n` consecutive 1s starting at position `p` (and 0s elsewhere). Perform a bitwise AND operation between `x` and this mask. Compare the result of this AND operation with the mask itself. If they are equal, it means all the bits in `x` that correspond to the 1s in the mask were also 1.

- **Question:** How to check if all bits within a specific range (defined by `n` bits starting at position `p`) are set to 0 in an integer `x`?

    - **Approach:** Create a mask that has `n` consecutive 1s starting at position `p` (and 0s elsewhere). Perform a bitwise AND operation between `x` and this mask. If the result of this AND operation is 0, it means none of the bits in `x` that correspond to the 1s in the mask were set (i.e., they were all 0).

- **Question:** How to find the position of the least significant zero bit in a non-all-ones integer `x`? (Return -1 if `x` is all ones for its type).

    - **Approach:** First, invert all the bits of `x` using a bitwise NOT operation (`~x`). In the inverted number, the least significant zero bit of the original `x` will now be the least significant set bit. Use the approach from Question 13 to find the position of the least significant set bit in `~x`.

- **Question:** How to find the position of the most significant set bit in a non-zero integer `x`? (Return -1 if `x` is 0). Assume a fixed integer size, say `SIZE`.

    - **Approach:** You can iterate from the most significant bit position (`SIZE - 1`) downwards to position 0. For each position, check if the bit at that position is set using the approach from Question 3. The first position (starting from `SIZE - 1`) where the bit is set is the position of the most significant set bit.

- **Question:** How to create a mask with alternating 1s and 0s (e.g., `01010101` or `10101010`) for a given integer size `SIZE`?

    - **Approach:** You can start with a basic alternating pattern for a few bits (e.g., `01` or `10`). Then, you can replicate this pattern to fill the integer size by repeatedly shifting the current pattern left and ORing it with itself, adjusting the shift amount at each step to double the length of the pattern (`pattern = (pattern << step) | pattern; step *= 2`). You might need to handle the initial step and pattern carefully depending on whether you want the pattern starting with 01 or 10.

- **Question:** How to swap the bit sequence of length `n` starting at position `p1` with the bit sequence of length `n` starting at position `p2` in integer `x`, assuming the two ranges of bits do not overlap?

    - **Approach:** This requires extracting, clearing, and then inserting. First, extract the `n` bits starting at `p1` (using the approach from Question 8). Extract the `n` bits starting at `p2` (using the approach from Question 8). Clear the `n` bits starting at `p1` in the original

`x` (using the approach from Question 6). Clear the `n` bits starting at `p2` in the (already modified) `x`. Position the extracted bits from `p1` so they can be inserted into position `p2` (requires shifting the extracted bits). Position the extracted bits from `p2` so they can be inserted into position `p1` (requires shifting the extracted bits). Finally, OR the cleared `x` with the two positioned extracted bit sequences.

- **Question:** How to calculate the integer negation of `x` using bitwise operations, assuming a two's complement representation?
    - **Approach:** In a two's complement system, the negation of a number `x` is equivalent to performing a bitwise NOT operation on `x` (inverting all its bits) and then adding 1 to the result.

- **Question:** How to count the number of trailing zero bits in a non-zero integer `x`? (Return the count of 0s from the rightmost bit up to the first set bit).
    - **Approach:** One way is to find the position of the least significant set bit using the approach from Question 13. The position of the least significant set bit (if positions are 0-indexed from the right) is equal to the number of trailing zeros. Alternatively, you could repeatedly check the rightmost bit and right-shift the number until the rightmost bit is 1, counting the number of shifts.