Assume `int x, n, m;`  
All are 32-bit signed integers.

1. Write `(32-n)` using only `+` and bit-level operators.

Since in 2's complement, we know that `–n = ~n+1`, we have:

\[
32 - n = 32 + (-n) = 32 + (~n + 1)
\]

2. Clear the `n (n>0)` most-significant bits of `x`. Use only small constants no greater than 255, i.e. 0xFF. Note that to allow for `n=0`, you will need to isolate this case since the compiler gets confused when you try to shift an amount `>=` type size, and then use the OR operator to combine both cases.

\[
m = ~0 ((32 - n) = ~0 (32 + ~n + 1)\\
x = x & ~m
\]

3. Initialize `m` to `0x 01 01 01 01`

\[
m = (0x01 << 8) | 0x01\\
m = (m << 16) | m
\]

4. Test if `x >= 0`.

\[
!(x >> 31)
\]

Note that if `x` is zero or positive, then the resulting shift gives all 32 0's (the sign bit is 0). The negation of zero (False) is True, i.e. 1.

If `x` is negative, then the resulting shift gives all 1's (the sign bit is 1). The negation of non-zero (True) is False, i.e. 0.

5. Extend the 3-bit binary integers 100 and 011 to 5-bit.

We just sign extend them to yield 11 100, and 00 011.

And vice versa, given these 5-bit representations, we know that the numbers they represent would fit in 3-bit, since we only need to drop the extra sign bits.

6. Another fact on 2'complement: For every `x`, either `x` or its negative `–x`, is negative. The only exception is zero, since both 0 and -0 (`~0+1`) are all 0's, i.e. positive.

7. Assign `x` the value `0xFF FF FF`:

\[
x = ~1 + 1
\]