Untitled-1

```
/* howManyBits - return the minimum number of bits required to represent x in
 *             two's complement
 *  Examples: howManyBits(12) = 5
 *            howManyBits(298) = 10
 *            howManyBits(-5) = 4
 *            howManyBits(0)  = 1
 *            howManyBits(-1) = 1
 *            howManyBits(0x80000000) = 32
 *  Legal ops: ! ~ & ^ | + << >>
 *  Max ops: 90
 *  Rating: 4
 */
int howManyBits(int x) {
  // keep in mind that this asks for the minimum number of bits required
  // to represent a number in binary, not the number of bits set (equal to 1)

  // the concept behind this is simple, we first get the sign from x
  // because the MSB is always the sign

  // if the number is negative then we add one to the final result
  // otherwise we can disregard the MSB because it would be 0

  // 1) get the sign from the number
  // 2) flip all the bits if the number is negative (turn it into a
  // positive number
  // 3) starting from the middle of the binary array, check if the left side is equal
  // to 0 using !!(x >> 16)
  // 3.1) if the left side is equal to zero, that means there's no data there, so we
don't need
  // to shift x (when x = 0, !!(x >> 16) is 0, therefore 0 << 4 is still 0)
  // 3.2) if the left side is not zero, then that means all the bits to the right
contains USEFUL
  // information, hence we set pos16 to 16 and then shift x by 16 bits to the left
  // Rinse and repeat with 8 bit, 4 bit, 2 bit, and one bit shifts

  // This is like doing a binary serach for the position of the last set bit after the
MSB

  int sign = x >> 31;
  // printf("%d\n", sign);
  int pos16, pos8, pos4, pos2, pos1, pos0;
  x = x ^ sign; // flip each bit of x if x < 0

  pos16 = !!(x >> 16) << 4; // pos16 = 16 if (x >> 16) != 0
  x >>= pos16;

  pos8 = !!(x >> 8) << 3; // pos8 = 8 if (x >> 8) != 0
  x >>= pos8;

  pos4 = !!(x >> 4) << 2; // pos4 = 4 if (x >> 4) != 0
  x >>= pos4;

  pos2 = !!(x >> 2) << 1; // pos2 = 2 if (x >> 2) != 0
```

```
    x >>= pos2;

    pos1 = !!(x >> 1); // pos1 = 1 if (x >> 1) != 0
    x >>= pos1;

    pos0 = x;

    // because we disregarded the MSB earlier, add a 1 because the MSB is always required
  to
    // differentiate between positive and negative numbers
    // we always add a one because the MSB is required regardless of the sign
    return pos16 + pos8 + pos4 + pos2 + pos1 + pos0 + 1;

    // int sign = x >> 31;
    // int not_x = ~x;
    // int mask = sign | not_x;
    // int num_bits = 0;

    // int i = 0;
    // i += 2;

    // num_bits = (mask >> 16) & 16;
    // num_bits |= ((mask >> (num_bits + 8)) & 8);
    // num_bits |= ((mask >> (num_bits + 4)) & 4);
    // num_bits |= ((mask >> (num_bits + 2)) & 2);
    // num_bits |= ((mask >> (num_bits + 1)) & 1);

    // return num_bits + 1;
  }
```