Untitled-1

```
//1
/*
 * bitXor - x^y using only ~ and &
 *   Example: bitXor(4, 5) = 1
 *   Legal ops: ~ &
 *   Max ops: 14
 *   Rating: 1
 */
int bitXor(int x, int y) {
  // compute the bitwise complement of x & y, equivalent to the bitwise XOR of x and y
(all bits flipped)
  int a = ~(x & y);

  // compute the bitwise complement of x & y, which is the bitwise OR of x and y (all
bits flipped)
  int b = ~(~x & ~y);

  // the AND of these two returns the XOR of x and y
  return a & b;
}
/*
 * tmin - return minimum two's complement integer
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 4
 *   Rating: 1
 */
int tmin(void) {
  // -1 is 1111...1111, all the bits after the MSB are flipped,
  // so the largest negative number would be 1000...0000
  // hence, we shift 1 to the left by 31 bits

  return (0x01 << 31);
}
//2
/*
 * isTmax - returns 1 if x is the maximum, two's complement number,
 *     and 0 otherwise
 *   Legal ops: ! ~ & ^ | +
 *   Max ops: 10
 *   Rating: 1
 */
int isTmax(int x) {
  // if x is the the maximum int, then adding 1 to it would make it the smallest int
  // 0111 .... 1111 + 1 => 1000 .... 0000

  int x1 = x + 1;
  // XOR x1 with x would give all 1s because their bits must be all different
  int XOR = x^x1;

  // flip all the bits in XOR, if x is the max then it would be all 0s, and the logical !
would turn it into true
  /// we also need to make sure x1 is not -1
  return !(~XOR | (!x1));
```

```
    // return !((~(x+1)^x)|(!(x+1)));
}
```