1. A. Write a function `printBinary(int n)` that takes a 32 bit signed integer and prints it's binary representation. **[1]**

Input	Output
10 -10	0000 0000 0000 0000 0000 0000 0000 1010 : 2 1111 1111 1111 1111 1111 1111 1111 0110 : 30

2. Write a function `int setBit(int n, int b)` that will set the b^{th} bit of an integer n. **[1]**

Input	Output
10 2 10 3	14 10

3. Write a function `int resetBit(int n, int b)` that will reset the b^{th} bit of an integer n. **[2]**

Input	Output
14 2 10 2	10 10

4. Write a function `int justLarger(int n),` Takes an integer n, returns the integer just larger than n but has same number of set bit in binary representation. **[6]**

Input	Output
412 423	419 427

Explanation:
(0000 0000 0000 0000 0000 0001 10**01** 1100)$_2$ = (412)$_{10}$
(0000 0000 0000 0000 0000 0001 10**10** 0011)$_2$ = (419)$_{10}$
(0000 0000 0000 0000 0000 0001 1010 0111)$_2$ = (423)$_{10}$
(0000 0000 0000 0000 0000 0001 1010 1011)$_2$ = (427)$_{10}$

Algorithm:

1. Find the first occurrence of “01” from right. Say you encountered p 0 and q 1 before that.
2. Swap “01” to “10”.
3. Reset next p bit [use `resetBit(int n, int b)`]
4. Set next q bit[use `setBit(int n, int b)`]

0000 0000 0000 0000 0000 0001 10**01** 1100
Here p=2, q=2.
0000 0000 0000 0000 0000 0001 10**10** 1100 [swapping done]
0000 0000 0000 0000 0000 0001 1010 **00 11** [p zeros then q ones]

5. Write a function `int justSmaller(int n),` Takes an integer n, returns the integer just smaller than n but has same number of set bit in binary representation. **[Bonus]**

Input	Output
419 427	412 423