Bonus 4.5 41247001S 盧昱安

把程式碼跑了一下
發現問題是：
輸入正整數時最後一個 bit 是錯誤的
輸入負整數時整個都是錯誤的（永遠 32 個 1）

於是我先做了一個測試。
先印出 bit 的值。是 -2147483648，沒有問題。
因為題目的 code 是將其變成
10000000 00000000 00000000 0000000

然而我接下來將 bit 右移一個位元 (>>= 1)
馬上就發現了這題問題的原因
右移了一位並沒有變成
01000000 00000000 00000000 00000000

而是變成了
11000000 00000000 00000000 00000000

馬上就找出問題了
我發現若這樣設定，不斷右 shift 位元只會一直移動 1
但最左邊也會繼續補 1 而非 0
所以到後面會變成
11111111 11111111 11111111 11111111
而非原本所要的
00000000 00000000 00000000 00000001

這也是為甚麼前面輸入負數永遠都是輸出 32 個 1
而輸入正數會出現問題也是因為如此
因為只要第一個 1 開始對到之後，後面全部都會變成符合條件
（因為是 int32_t，負數第一個就開始配到 1 了，所以 & operator 馬上有 value，導致瘋狂的 printf 1 出來。）

而為甚麼我們 right shift
會橫空一直補 1 出來我上網找到了一個
stackoverflow 的答案
https://stackoverflow.com/questions/141525/what-are-bitwise-shift-bit-shift-operators-and-how-do-they-work

原因是因為 C / C++ 的 >> 運算子是 (arithmetic right shift) 而非 (logical right shift)。因此最左邊會補上 most significant bit (int32_t 的形況下就變成判斷正負的那個 bit 了)。所以才會出現範例程式他一直補 1 的情況。

我想到簡單的解決方法有幾種

第一種：剛開始改成左移 30 格就好，不要 31 格。因為這樣就不會把 1 放到第一個判斷正負的地方。然後前面再多寫一個 if else 判斷正負就好。

第二種：迴圈裡面右移的時候把該 bit 前面全部的 bit 都改成 0。可以用 (& 原本自己的方式去掉所有的 0)