Bit Manipulation

Problem Statement
Find the largest integer that can be created from a pair of integers and the following bit manipulation operators: <<, >>, and |. Once the | operator is applied, the two inputs are consumed and cannot be referred to again. So you can shift the two original integers as much as you want, then apply the | operator to yield a result.

Input Format
The first line and second line contain two integers n1 and n2.

Output Format
Print out the largest integer that can be created using n1 and n2 and the bit manipulation operators <<, >>, and |.

Constraints
-2147483647<=n1<=2147483647
-2147483647<=n2<=2147483647

Sample Input
5
6

Sample Output
1946157056

```
import java.util.*;

public class Solution {
    public static void main(String args[]) throws Exception {
        Scanner myscanner = new Scanner(System.in);
        int num1 = myscanner.nextInt();
        int num2 = myscanner.nextInt();
        int record=Integer.MIN_VALUE;
        for(int i = 0;i<=32;i++){
            for(int j=0;j<=32;j++){
                if(((num1<<i)|(num2<<j))>record){
                    record=((num1<<i)|(num2<<j));
                }
                if(((num1>>i)|(num2<<j))>record){
                    record=((num1>>i)|(num2<<j));
                }
                if(((num1<<i)|(num2>>j))>record){
                    record=((num1<<i)|(num2>>j));
                }
```

```
                if(((num1>>i)|(num2>>j))>record){
                    record=((num1>>i)|(num2>>j));
                }
            }
        }
        System.out.println(record);

//This is just brute force - try everything, no thinking required

    }
}
```

Little Endian to Big Endian

Problem Statement
The goal is to convert a 32-bit integer from Little Endian (Java, iPhone, Xbox) to Big Endian (Intel, TCP, IPv6) encoding, and also to other possible encodings as well. This is a very common task in software engineering. In Little Endian encoding the first byte contains the most significant bits, and bytes 2, 3, and 4 decrease in significance. In Big Endian encoding the first byte contains the least significant bits, and bytes 2, 3 and 4 increase in significance. Thus, to move from Little Endian to Big Endian the byte shuffling is as follows:

Byte 1 -> Byte 4
Byte 2 -> Byte 3
Byte 3 -> Byte 2
Byte 4 -> Byte 1

We could of course shuffle the four bytes into a different order (e.g. Byte 3, Byte 4, Byte 1, Byte 2). The goal is to take in an integer encoded in Little Endian, and shuffle its bytes using bit manipulation operators and put them into the specified order.

Input Format
An integer which represents n in 32-bit Little Endian format.
This is followed by a 4-digit String, where each digit specifies the new ordering of each byte. For example "4321" would specify Big Endian ordering of the bytes (e.g. Byte 4 first, followed by Byte 3, Byte 2, Byte 1). This 4-digit String tells you how to shuffle up the bytes of n.

Output Format
An integer which represents n with its bytes arranged into the specified format.

Sample Input
3
4321

Sample Output
50331648

```
import java.util.*;

public class Solution {
    public static void main(String args[]){
        Scanner myscanner = new Scanner(System.in);
        int num = Integer.parseInt(myscanner.nextLine());
        String digits = myscanner.nextLine();

        int byte1 = ((num>>>0)<<24)>>>digits.indexOf('4')*8;
        int byte2 = ((num>>>8)<<24)>>>digits.indexOf('3')*8;
        int byte3 = ((num>>>16)<<24)>>>digits.indexOf('2')*8;
        int byte4 = ((num>>>24)<<24)>>>digits.indexOf('1')*8;
        System.out.println(byte1|byte2|byte3|byte4);

// The idea here is you kind of zig-zag the bits left and right to blank
out the ones you don't like. And you move them into position. Then you
combine all the 4 separate chunks with an OR - because the bits are all in
different places, they don't conflict with each other.
    }
}
```