1. Find a collision in each of the hash functions below:

a. H(x) = x mod 7^{12}, where x can be any integer

Collision: x = 1 and y = 7^{12} + 1

b. H(x) = number of 1-bits in x, where x can be any bit string

Collision: x = 00001000 and y = 1000000

c. H(x) = the three least significant bits of x, where x can be any bit string

Collision: x = 00000001 and y = 11110001

2. Prove the statement: In a class of 500 students, there must be two students with the same birthday.

Proof:

1. Assumed that there are 366 possibilities for birthdays in one year (including February 29th).
2. There are 500 students in the class.
3. Using the pigeon-hole principle, there must be at least two students with the same birthday.
    a. Pigeonholes: Number of possible birth date within a year [1-366].
    b. Pigeons: Students [1-500]
    c. Collision: There must be two students "mapped" to a specific date of birthday.

3. Find an x such that H (x ◦ id) ∈ Y where

a. H = SHA-256

b. id = 0xED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77

c. Y is the set of all 256 bit values that have some byte with the value 0x1D.

Assume SHA-256 is puzzle-friendly. Your answer for x must be in hexadecimal. You may provide your code for partial credit, if your x value is incorrect. You may use the accompanying CryptoReference1.java file to help you with this question. Submit both your code and your value of x.

Notes:

- The notation "x ◦ id" means the byte array x concatenated with the byte array id. For example, "11110000 ◦ 10101010" is the byte array "1111000010101010".

- The following two code segments are **not** equivalent:

Segment 1

```
String val =
"0x4E4135E7746419FEB0";
if (val.contains("1D"))
return true;
```

Segment 2

```
String val =
"0x4E4135E7746419FEB0";
byte[] arr =
val.getBytes(); for (byte
b : arr) if (b == 0x1D)
return true;
```

The second code segment above is the correct way to check whether 0x1D is a byte in 0x4E4135E7746419FEB0. Remember that hex format is only a way to represent a byte sequence in a human readable format. You should not be performing operations directly on hex-string representations. Instead, you should first convert hex-strings into byte arrays, then perform operations on the byte arrays directly, and then convert the final byte array into a hex format when giving your answer. Performing operations directly on the hex strings is incorrect.

Ans:
x could be
0x46EA26E0F28DAE6DAF6CC0163073E9C2143CC99175B7BF939034441BE321A04E

Code:

```
/**
 *
 * @author Shaowei
 */
import javax.xml.bind.DatatypeConverter;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Random;

public class FindX {

    public static void main(String[] args) throws NoSuchAlgorithmException, IOException {

        boolean found = false;
        int t = 1;

        while (found != true) {
            System.out.println("------- This is " + t + " times----------");
            //Convert idHex into a byte array. API requires omitting the leading "0x".
            String idHex =
"ED00AF5F774E4135E7746419FEB65DE8AE17D6950C95CEC3891070FBB5B03C77";
            byte[] id = DatatypeConverter.parseHexBinary(idHex);
            System.out.println("# id: " + DatatypeConverter.printHexBinary(id));
            System.out.println("# hex digits in id: " + idHex.length());

            //Generate a pseudo-random 256-bit x.
            byte[] x = new byte[32]; //256 bit array
            new Random().nextBytes(x); //pseudo-random
            String xHex = DatatypeConverter.printHexBinary(x);
            System.out.println();
            System.out.println("x: " + xHex);
            System.out.println("# hex digits in x: " + xHex.length());
            System.out.println("# bits in x: " + x.length * 8);

            //Concatentate two byte arrays
            ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
            outputStream.write(x);
            outputStream.write(id);
            byte concat[] = outputStream.toByteArray();
            String concatHex = DatatypeConverter.printHexBinary(concat);
            System.out.println();
            System.out.println("x concatenate id: " + concatHex);

            //Apply SHA-256 to hash x||id
            MessageDigest digest = MessageDigest.getInstance("SHA-256");
            byte[] hash = digest.digest(concat); //SHA256 hash
            String hashHex = DatatypeConverter.printHexBinary(hash);
            System.out.println();
            System.out.println("Hash: " + hashHex);
            System.out.println("# hex digits in hash: " + hashHex.length());
            System.out.println("# bits in hash: " + hash.length * 8);

            //Check whether H (x ◦ id) contains Y or not
            String val = DatatypeConverter.printHexBinary(hash);
            System.out.println(val);
            byte[] b = DatatypeConverter.parseHexBinary(val);
            System.out.println(DatatypeConverter.printHexBinary(b));
            for (byte a : b) {
                if (a == 0x1D) {
                    found = true;
                    System.out.println("Found!");
                }
            }
            t++;
            System.out.println("x in HEX will be " + xHex);
        }
    }
}
```

4. Alice and Bob want to play a game over SMS text where Alice chooses a number between 1 and 10 in her head, and then Bob tries to guess that number. If Bob guesses correctly, he wins. Otherwise, Alice wins. However, Bob complains that the game isn't fair, because even if Bob guessed correctly, Alice could lie and claim that she chose a different number than what she initially chose. What can Alice do to prove that she didn't change the number she initially chose? Devise a mechanism to address Bob's concern. Provide a detailed explanation of the mechanism and why it works. An answer with insufficient detail will not receive credit.

Step I. Alice chooses a number between 1 and 10. Then she takes this number and a secret random value (called 'nonce') to do a certain type of hash function.

Step II. Alice then sends the result of hash function H (number, nonce) to Bob. Upon receiving the text of hash function result, Bob now can guess the number Alice chosen and he might send the text of his guessing to Alice.

Step III. After receiving Bob's guessing text, Alice can send a text to tell Bob whether he is correct or not. Also, she must send the answer (the original number she chose) and nonce she used (the secret random number) to Bob for verification as well.

Step IV. If Bob has concern with Alice about cheating, he can apply the answer and the nonce to the hash function. If the result he got does not meet the one Alice sent him on Step II, he could accuse Alice of lying. Otherwise, Alice is proven that she is a fair player and did not cheat on the game.