Pham Hoang Long Do
NetID: 2995060

Homework 1

CSE 3400
Ghada Almashaqbeh

Problem 1

Part a

Python 1: XORing with 8003

```
import time

def main():
    L = [2**(10),2**(20),2**(30)]
    for i in L:
        start = time.time()
        for counter in range(0,i+1,1):
            xor = counter ^ 8003
        end = time.time()
    print(end-start)
```

`CPU: I7-1165G7 (4 cores - 8 threads - 2.8GHz Clock Speed)`
`RAM: 16GB`
`Language: Python`

Use the for the CPU time formula:

$$\text{CPU time} = \text{Intructions count} \times CPI \times \text{Clock Cycle}$$

Since we use the same code for every counter $2^{10}, 2^{20}, 2^{30}, 2^{330}$ so the CPI will be the same. Lets consider the two cases 2^{30} and 2^{330} which is CPU Time$_A$ and CPU Time$_B$ respectively. There is only one intruction (XORing) in the body of the code. Therefore, the total instructions count for CPU Time$_A$ is 2^{30} instructions and for CPU Time$_B$ is 2^{330} instructions. Clock cycle will always be the same. In addition, this algorithm has the 0(n) run time.

$$38.3 = 2^{30} \times CPI \times \text{Clock Cyle}$$
$$\text{CPU Time}_B = 2^{330} \times CPI \times \text{Clock Cyle}$$

Divide both equations, we have:

$$\frac{38.3}{\text{CPU Time}_B} = \frac{2^{30}}{2^{330}}$$

Therefore

$$\text{CPU Time}_B = \frac{2^{330} \times 38.3}{2^{30}} = 7.8 \times 10^{91} seconds = 2.47 \times 10^{84} years$$

Part b

Python 2: Dividing with 4009

```
import time

def main():
    L = [2**(10),2**(20),2**(30)]
    for i in L:
        start = time.time()
        for counter in range(0,i+1,1):
            xor = counter / 4009
        end = time.time()
        print(end-start)
```

Use the same logic that we used to solve part a since this question, we have:

$$\frac{31.3 \times 2^{330}}{2^{30}} = 6.374 \times 10^{91} \text{ seconds} = 2.02 \times 10^{84} years$$

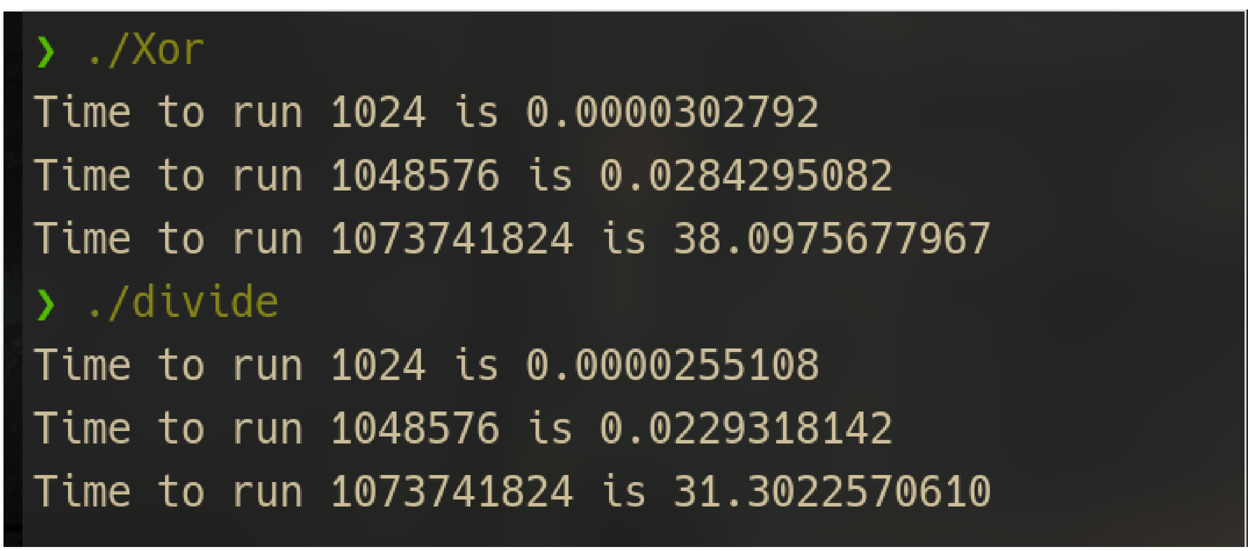

Figure 1: Result when XORing and Dividing

Problem 2

1. $G_2(s \| t \| z) = G_1(s) \| (t \oplus z)$
   G_2 is not a secure PRG for the following reasons:
   (a) $G_1(s)$ is a secure PRG
   (b) $t \oplus z$ is not random since z is random but t is not necessary random; therefore, half of the string could be not necessarly random.

   Therefore, the attacker could ignore the $G_1(s)$ and go after the $t \oplus z$ part. Hence, $G_2(s)$ is ***Not A Secure PRG***

2. $G_3(s) = G_0(\bar{s}) \oplus 1^{2n}$
   (a) $G_0(\bar{s})$ is a secure PRG since $G_0(s)$ is the PRG and $\bar{s}$ will not change the distribution of $0, 1$
   (b) $1^{2n} = 1^n$ which is a string of 1 that has length n. This is a fixed pattern.

   We basically flip the bit of $G_0(\bar{s})$. In orther words, $G_3(s) = \tilde{\ } G_0(\bar{s})$. We use arguement from 1 to say that $G_0(\bar{s})$ is a secure PRG. Therefore, $G_3(s)$ is a ***Secure PRG***
3. $G_4(s \parallel z) = ((s \parallel z) \oplus G_0(s)) \parallel G_0(s)$
   (a) $G_0(s)$ is a secure PRG
   (b) $s \oplus z$ will produce another random string since both s and z are random.
   (c) $((s \parallel z) \oplus G_0(s))$ is a secure PRG due to the above properties.
   (d) $((s \parallel z) \oplus G_0(s)) \parallel G_0(s)$ is a secure PRG since we concatenate property c with property a.

   Therefore, $G_4(s)$ is ***A Secure PRG***.
4. $G_5(s) = \mathsf{msb}(G_0(s)) \parallel G_0(s) \parallel G_1(z)$, where msb is the most significant bit.
   (a) $G_0(s)$ and $G_1(s)$ are a secure PRG
   (b) The attacker knows the algorithm that constructed G_0 but he doesn't know the seed. However, he still has a non-negligible chance of guessing $msb(G_0(s)$ which in this case is 50%

   Basically, the problem becomes,

   50% {0,1} || PRG || PRG

   $G_5(s)$ is ***A Secure PRG*** since he only has 50% change of guessing the correct MSB of $G_5(s)$, and the rest of $G_5(s)$ are still random.

Problem 3