Problem 1 :

1 : Output : Running Task 1...

Time taken for 1024 iterations: 0.000000 seconds

Time taken for 1048576 iterations: 0.005000 seconds

Time taken for 1073741824 iterations: 2.938000 seconds

```c
#define VALUE 1254632

void measure_time_for_counter(int64_t num_iterations) {
    clock_t start_time = clock();

    for (int64_t i = 0; i < num_iterations; i++) {
        int result = (i ^ XOR_VALUE) + (i / DIVISOR);
    }

    clock_t end_time = clock();
    double elapsed_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("Time taken for %lld iterations: %f seconds\n", num_iterations,
elapsed_time);
}

int main() {
    printf("Exercice 1 Question 1...\n");

    // Test cases
    measure_time_for_counter(1 << 10);
    measure_time_for_counter(1 << 20);
    measure_time_for_counter(1 << 30);

    return 0;
}


// I have as a result for 2 ^ 30 a time of 2.938000, so to estimate the time
of a counter incremented 2^480 times,
//we can use the fact that 2^480 = 2.938000 * 2^450. The number of seconds in
a year is 3,154e+7. So the number of years is :
//( 2.938000 * 2^450) / (3,154e+7)
//2^450 is 2.907355e+135, so the number of years is  : (2.938000 *
2.907355e+135) / (3,154e+7) = 1.32 x 10e128 years
```

2 : Output : Running Task 2...

Time taken for 1024 iterations: 0.000000 seconds

Time taken for 1048576 iterations: 0.008000 seconds

Time taken for 1073741824 iterations: 3.832000 seconds

```c
#define XOR_VALUE 990
#define DIVISOR 3456

void measure_time_for_counter(int64_t num_iterations) {
    clock_t start_time = clock();

    for (int64_t i = 0; i < num_iterations; i++) {
        int result = (i ^ XOR_VALUE) + (i / DIVISOR);
    }

    clock_t end_time = clock();
    double elapsed_time = (double)(end_time - start_time) / CLOCKS_PER_SEC;
    printf("Time taken for %lld iterations: %f seconds\n", num_iterations,
elapsed_time);
}

int main() {
    printf("Running Task 2...\n");

    // Test cases
    measure_time_for_counter(1 << 10);
    measure_time_for_counter(1 << 20);
    measure_time_for_counter(1 << 30);

    return 0;
}


// I have as a result for 2 ^ 30 a time of 3.832000 seconds, so to estimate
the time of a counter incremented 2^480 times,
//we can use the fact that 2^480 = 3.832000 * 2^450. The number of seconds in
a year is 3,154e+7. So the number of years is :
//( 3.832000 * 2^450) / (3,154e+7)
//2^450 is 2.907355e+135, so the number of years is  : (3.832000 *
2.907355e+135) / (3,154e+7) = 3.53 x 10e128 years
```

[redacted]

Problem 2 :

1:

 Go is a PRG, so its output will be undistinguishable from random. But t is XORed with 1^n, so if we know t, or t is not random, then G2 (s||t) = G0(s) || (t XOR 1^n) wont be random. Then G2 is not a PRG.

A way to attack G2 could be by knowing t, after that, we would be awared of the value of G0(s) because we would know the value of t XOR 1^n

Then the G0(s) would be known

2:

We have s € {0, 1}^n and z € {0,1}^n.

G3 is applying Go, a PRG, on the XOR of s and z, random seeds, so G3 is undistinguishable from a random number, G3 is a PRG.

3:

LH(G0(s)) is the left half of G0(s), which is undistinguishable from random.

In the same way, the right half of a randomseed, applied to G1 is indistinguishable from random. So the XOR between those two is a PRG too.

4:

G5(s) = (G0(s) mod 2) || G0(s)

G0(s) mod 2 extract the less significant bit of the PRG output. Then it is concatenated with G0(s) which is a PRG.

So it leaks informations about the output, which is exploitable by an adversary.

For example, the last bit of G0 is leaked, so the attacker can reduce the entropy of the output, which is a leak.
So G5(0) is not secure.

Problem 3 :

1 : In the monoalphabetic substitution ciper, we can try to decrypt the ciphertext by analysing the frequency f each letter in the cipher, and relate thosethose frequencies to the frequencies of the language of the sentence. So, the bigger the ciphertext is, the more we will have accurate frequencies and the easier it will be to decrypt.

2: The answer will be different considering a CPA attacker, because it can directly reveal parts of the key (the substitution permutation) without caring of the frequencies. So, short cyphertext would be enough in a CPA attack to have the key. Just imaging choosing the 26 letters of the alphabet as a plaintext, we would know the key directly.

Part 2 :

1 : Y1 and Y2 depends on x. So if we know x we will know P1 and P2.

Nevertheless, we cannot know x with a CTO attack because we cannot reverse the XOR without x.

2 : The probleme here is that Eve uses p and p+1. P iis random but p+1 is now predictable. Which can be used as a weakness in this case of CTO attack.

If the attacker knows C1 and C2 he can do :
C1 XOR C2 = (p XOR m1) XOR ((p+1) XOR m2)

Which means :

(p XOR (p+1)) XOR (m1 XOR m2)

It gives us informations about the 2 messages as p XOR (p+1) is a fixed value of ones.