Homework 2

> ## Homework Maths 2
>
> 1. Modular arithmetic - you just need to find examples, you don't need to prove anything.
>   1. Is it true that all odd squares are ≡ 1 (mod 8) ?
>   2. what about even squares (mod 8) ?
> 2. Try out the vanity bitcoin address example at asecurity or the Ethereum version
> 3. What do you understand by
>   1. O(n)
>   2. O(1)
>   3. O(log n)
>
> For a proof size, which of these would you want ?

1) 1.1 In modular arithmetic, all odd squares are indeed congruent to 1 modulo 8. Here are some examples:

- (1^2) mod 8 = 1 (1 is an odd square, and it is congruent to 1 mod 8).
- (3^2) mod 8 = 1 (9 is an odd square, and it is congruent to 1 mod 8).
- (5^2) mod 8 = 1 (25 is an odd square, and it is congruent to 1 mod 8).
- (7^2) mod 8 = 1 (49 is an odd square, and it is congruent to 1 mod 8).
- (9^2) mod 8 = 1 (81 is an odd square, and it is congruent to 1 mod 8).
- (11^2) mod 8 = 1 (121 is an odd square, and it is congruent to 1 mod 8).

So, for any odd integer "n," the square of "n" (n^2) is always congruent to 1 modulo 8. This is a consistent pattern in modular arithmetic for odd squares.

1.2 For even squares, the congruence modulo 8 varies. Even squares can be congruent to different values modulo 8, and they follow a different pattern than odd squares. Here are some examples:

- (2^2) mod 8 = 0 (4 is an even square, and it is congruent to 0 mod 8).
- (4^2) mod 8 = 0 (16 is an even square, and it is congruent to 0 mod 8).
- (6^2) mod 8 = 4 (36 is an even square, and it is congruent to 4 mod 8).
- (8^2) mod 8 = 0 (64 is an even square, and it is congruent to 0 mod 8).

As you can see, even squares have different congruences modulo 8, and they can be congruent to 0, 4, or other values depending on the specific even integer being squared.

In summary, for even squares, the congruence modulo 8 is not fixed at 1, as it is for odd squares. Even squares follow a pattern modulo 8 based on their value, and they can be congruent to various values, including 0 and 4.

3)

O(n) (Linear):The running time of an algorithm grows proportionally with the input size "n." If "n" doubles, the running time roughly doubles.

O(1) (Constant): The running time of an algorithm remains constant and doesn't depend on the input size "n."

O(log n) (Logarithmic): The running time of an algorithm grows slowly as the input size "n" increases. It's often faster than linear and is characterized by dividing the problem in each step.

For the reasons mentioned above, in our system we will prefer O(1).