Write code to implement the following algorithms. You can use any language, but you can not use built-in mechanisms to achieve the essential task (for example MATLAB might have a routine for checking Primality; you cannot use it if the exercise task is to implement the Primality test.)

Discuss at recitation, if there is time. Demo to a TA for Extra Credit.

**EC 1.** Implement Euclid’s algorithm recursively. Take as input two integers and output their GCD.

**EC 2 (difficulty ★).** Implement Euclid’s algorithm non-recursively using a stack.

**EC 3 (difficulty ★).** Implement Euclid’s Extended Algorithm recursively. Take as input two integers \(a, b\) and output their GCD \(d = \gcd(a, b)\) together with gcd-coefficients \(k, h\) such that \(ak + bh = d\)

**EC 4.** Implement a linear cipher. Given integers \(n, a, b\)
- Use Euclid-Extended to verify that \(\gcd(n, a) = 1\), and to get the gcd-coefficients for \((n, a)\). If \(n, a\) are not coprime print an error and terminate.
- implement \(\text{encode}(x, n, a, b)\) to map input \(x\) into cipher \(y = ax + b \mod n\)
- implement \(\text{decode}(y, n, a, b)\) to map cipher \(y\) into \(x = a^{-1}(y - b) \mod n\)

**EC 5 (difficulty ★).** Implement the Primality Test based on Fermat’s Little Theorem. Given an integer \(n\), output 0 when the test fails, and 1 when it does not fail (some numbers will pass the test without being prime - verify this on few Carmichael numbers)

**EC 6 (difficulty ★).** Implement RSA. Given integers \(p, q\)
- test primality for \(p, q\). If they pass compute \(n = pq\).
- find a public key \(e\) coprime with \((p - 1)(q - 1)\). Output this key together with \(n\)
- compute private (secret key) \(d = e^{-1} \mod (p - 1)(q - 1)\)
- implement \(\text{encode}(x, n, e)\) to map input \(x\) into cipher \(y = x^e \mod n\)
- implement \(\text{decode}(y, n, d)\) to map cipher \(y\) into \(x = y^d \mod n\)