Section 1.1

1. $t \leftarrow a, a \leftarrow b, b \leftarrow c, c \leftarrow d, a \leftarrow t$

2. If m is less than n, then on the first execution of **E3**, m and n will be switched. Otherwise, n is always greater than r by definition.

3. **F1.** [Find remainder.] Divide m by n and let r be the remainder. (We will have $0 \leq r < n$.)

   F2. [Is it zero?] If $r = 0$, the algorithm terminates; n is the answer.

   F3. [Find remainder 2.] Divide n by r and let m be the remainder. (We will have $0 \leq m < r$.)

   F4. [Is it zero?] If $m = 0$, the algorithm terminates; r is the answer.

   F5. [Find remainder 3.] Divide r by m and let n be the remainder.

   F6. [Is it zero?] If $n = 0$, the algorithm terminates; m is the answer, otherwise go to step F1.

4. $r = 1767$, 399, 171, 57, 0. The answer is 57.

5. The algorithm is not finite, its steps are not definite and there is no output. Regarding formatting, the algorithm has no name, its steps are not prefixed by a letter and there is no solid bar at the end to indicate the algorithm is complete.

6. If $m = 1$, E1 is executed 2 times, for $m = 2$ 3 times, for $m = 3$ 4 times, for $m = 4$ 3 times and for $m = 5$ once. The average is $\frac{(2+3+4+3+1)}{5} = \frac{13}{5}$.

7. When $n > m$, after the first iteration of E1, m and n will be swapped. Since, there are infinitely more cases where $n > m$, we can treat average number of steps using T_m where m is fixed. Therefore, $U_m = T_m + 1$ assuming that average is *strongly* average, such that we can ignore how many times E1 is executed when $m \geq n$.

8. The problem comes down to repeatedly transforming a string in the form of $a^m b^n$ into $a^{min(m,n)} b^{|m-n|}$ until only as remains. At that point, the length of the string is $gcd(m,n)$.

   The trick is to perform a series of match and replaces. The computation method allows us to find strings and replace them with other strings. To find $|m - n|$, the difference between m and n, we continually search for the string ab and replace it with an empty string until there are no more matches. At that point, we are left with only as or bs. We can then search for a and replace it with c and b and replace that with c. The end result is a string of cs who's length is the difference.

   Finding $min(m,n)$ requires an additional trick. We need to search for ab as we did for computing the difference, but we also need to store how many matches we have seen thus far. This requires a 'loop' with two steps. The first matches ab and if successful, replace it with the empty string and continues to step two. The second step unconditionally adds a c to the front and goes back to step two. This is the really tricky part. Eventually, we are left with $c^{min(m,n)}(a|b)^{|m-n|}$. That is we have $min(m,n)$ number of cs followed by $|m - n|$ number of a or b. Finally, we transform c into a and a into b to get $a^{min(m,n)} b^{|m-n|}$. When there are no more bs, the algorithm terminates with $a^{gcd(m,n)}$.

j	θ_j	ϕ_j	a_j	b_j	*Comment*
0	'ab'	''	2	1	Replace 'ab' with an empty string
1	''	'c'	0	0	Append a 'c' in the front
2	'a'	'b'	3	2	Replace 'a' with 'b'
3	'c'	'a'	4	3	Replace 'c' with 'a'
4	'b'	'b'	5	0	Check if there are any 'b's. If so, go to step 0.