Running time of Euclid’s Algorithm:

**Lemma 5.1:** For positive integers \( p \) and \( q \) with \( p > q \), \( p \mod q < p/2 \).

**Proof (idea):** Consider two cases:
(a) \( q \leq p/2 \) and (b) \( q > p/2 \).

**Usefulness of Lemma 5.1:**
\[
gcd(m, n) \\
\downarrow \\
gcd(n, r) \quad \text{where } r = m \mod n \\
\downarrow \\
gcd(r, r') \quad \text{where } r' = n \mod r
\]

- In the last step above, \( r' < n/2 \) by the lemma; i.e., after two iterations, the second argument reduces by a factor of \( 1/2 \).
- So, the number of iterations before the second argument becomes 0 is \( O(\log n) \).
- No. of arithmetic operations per iteration = \( O(1) \).
- Total no. of arithmetic operations = \( O(\log n) \).

Input size and running time:

- For sorting, input size is \( n \), the number of values to be sorted. Insertion Sort, Merge Sort, etc., are efficient algorithms (i.e., they run in time that is bounded by a polynomial in the input size).

- For binary search, the input size is \( n \), the size of array. Running time \( O(\log n) \) does not include the time needed to read in the array.

- For the GCD problem:
  - Input size = \( O(\log m + \log n) \).
  - Time to read input = \( O(\log m + \log n) \).
  - No. of arithmetic operations = \( O(\log n) \).
  - Time for each operation = \( O((\log m + \log n)^2) \).
  - Running time = \( O(\log n (\log m + \log n)^2) \).

So, Euclid’s Algorithm is efficient.
Recursive version:

Euclid(m, n) /* m > n */
  if (n = 0)
    return m
  else
    return Euclid(n, m mod n)

Notation: For all \(i \geq 1\), \(F_i\) denotes the \(i^{th}\) Fibonacci number.

Lemma 5.2: If \(m > n \geq 0\) and Euclid \((m, n)\) performs \(k\) recursive calls, then \(m \geq F_{k+2}\) and \(n \geq F_{k+1}\).

Proof: Induction on \(k\). (Details in class.)

Lamé’s theorem: For any integer \(k \geq 1\), if \(m > n \geq 0\) and \(n < F_{k+1}\), then Euclid \((m, n)\) performs at most \(k - 1\) recursive calls.

Exercise: Show that for all \(k \geq 2\), Euclid \((F_{k+1}, F_k)\) leads to exactly \(k - 1\) recursive calls.

Ref: Chapter 4 of text (omit Section 4.4).

Assumptions:
- Usual goal: To obtain an asymptotic (big-O) estimate.
- For small values of \(n\), function values are constants.

Substitution method:
- Uses induction to prove the solution.
- Common to use second form of induction.

Example 1: Prove by the substitution method that the solution to the recurrence

\[ T(n) = 2T\left(\left\lfloor n/2 \right\rfloor\right) + n \text{ for } n \geq 4 \]

with \(T(2) = 4\) and \(T(3) = 5\) is

\[ T(n) = O(n \log n). \]
Notes:
- We prove by induction that for some constant $c > 0$, $T(n) \leq cn \log_2 n$. (Details in class.)
- The value of constant $c$ is identified after going through the proof.

Example 2: Prove by the substitution method that the solution to the recurrence

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1 \text{ for } n \geq 2$$

with $T(1) = c_1$ is: $T(n) = O(n)$.

Notes:
- If we try to prove that $T(n) \leq cn$ for some constant $c > 0$, the attempt will fail. (Details in class.)
- Remedy: Strengthen the inductive hypothesis. We prove that for some constants $c$ and $b$, $T(n) \leq cn - b$. (Details in class.)
- The values of constants $c$ and $b$ are identified after going through the proof.

A pitfall to avoid: For the recurrence

$$T(n) = 2T(\lfloor n/2 \rfloor) + n \text{ for } n \geq 4$$

consider the following “proof” that $T(n) = O(n)$:

Verify basis as usual; assume that $T(k) \leq ck$ for $3 \leq k < n$. Now, consider $k = n$.

$$T(n) = 2(c\lfloor n/2 \rfloor) + n \text{ (by ind. hyp.)}$$
$$= cn + n$$
$$= O(n) \quad \text{(defn. of big-O)}$$

- The above “proof” is INCORRECT. What is assumed as the hypothesis for $k < n$, must be proven for $k = n$ as well.
- That is, we must prove that $T(n) \leq cn$ for $k = n$. We can’t use big-O at that point.
Changing variables: May help in converting an unfamiliar recurrence into a familiar one.

Example 3: Solve the recurrence

\[ T(n) = 2T(\sqrt{n}) + \log_2 n \]

Let \( m = \log_2 n \); that is, \( n = 2^m \). The recurrence becomes

\[ T(2^m) = 2T(2^{m/2}) + m. \]

Let \( S(m) = T(2^m) \). Now, the recurrence becomes familiar:

\[ S(m) = 2S(m/2) + m. \]

Solution: \( S(m) = O(m \log m) \).

Solution in terms of \( T(n) \):

\[ T(n) = O(\log n \log (\log n)). \]

Master Theorem: (MT)

Let \( a \geq 1 \) and \( b > 1 \) be constants, let \( f(n) \) be a function and let \( T(n) \) be defined by

\[ T(n) = aT(n/b) + f(n) \]

where \( n/b \) may be either \( \lfloor n/b \rfloor \) or \( \lceil n/b \rceil \).

Part 1: If \( f(n) = O(n^{\log_b a - \epsilon}) \) for some \( \epsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \).

Part 2: If \( f(n) = \Theta(n^{\log_b a}) \) then \( T(n) = \Theta(n^{\log_b a} \log n) \).

Part 3: If \( f(n) = \Omega(n^{\log_b a + \epsilon}) \) for some \( \epsilon > 0 \) and \( a f(n/b) \leq cf(n) \) for some \( c < 1 \) and for sufficiently large \( n \), then \( T(n) = \Theta(f(n)) \).

Notes:

- In all parts, \( f(n) \) is compared with \( n^{\log_b a} \). The “larger” function determines the solution.
- In Part 1, \( f(n) \) must be polynomially smaller than \( n^{\log_b a} \).
• In Part 3:
  - $f(n)$ must be \textit{polynomially} larger than $n^{\log_b a}$.
  - $f(n)$ must satisfy the \textit{regularity} condition
    $a f(n/b) \leq c f(n)$ as stated in MT.

• MT does \textit{not} cover all possible cases.

\underline{Example 1:} $T(n) = 4T(n/2) + n$.
  Here $a = 4$, $b = 2$. So, $\log_b a = 2$.
  $f(n) = n = O(n^{2-\epsilon})$ with $\epsilon = 1$.
  So, Part 1 of MT applies and $T(n) = \Theta(n^2)$.

\underline{Example 2:} $T(n) = 3T(n/2) + n$.
  Here $a = 3$, $b = 2$. So, $\log_b a = \log_2 3$.
  $f(n) = n = O(n^{\log_2 3-\epsilon})$ with $\epsilon = \log_2 3 - 1 \approx 0.59$.
  So, Part 1 of MT applies and $T(n) = \Theta(n^{\log_2 3})$.

\underline{Example 3:} $T(n) = 2T(n/2) + n$.
  Here $a = 2$, $b = 2$. So, $\log_b a = 1$.
  $f(n) = n = \Theta(n)$.
  So, Part 2 of MT applies and $T(n) = \Theta(n \log n)$.

\underline{Example 4:} $T(n) = T(2n/3) + 1$.
  Here $a = 1$, $b = 3/2$. So, $\log_b a = 0$.
  $f(n) = 1 = \Theta(1)$.
  So, Part 2 of MT applies and $T(n) = \Theta(\log n)$.

\underline{Example 5:} $T(n) = 2T(n/2) + n^2$.
  Here $a = 2$, $b = 2$. So, $\log_b a = 1$.
  $f(n) = n^2 = \Omega(n^{1+\epsilon})$ with $\epsilon = 1$.
  So, Part 3 of MT can be used if regularity condition holds.
  $a f(n/b) = 2f(n/2) = 2(n/2)^2 = \frac{1}{2}n^2$. So, regularity condition holds with $c = 1/2$.
  So, Part 3 of MT applies and $T(n) = \Theta(n^2)$.

\underline{Example 6:} $T(n) = 3T(n/4) + n \log_2 n$.
  Here $a = 3$, $b = 4$. So, $\log_b a = \log_4 3 < 1$.
  $f(n) = n \log_2 n = \Omega(n^{\log_4 3+\epsilon})$ with $\epsilon = 1 - \log_4 3$.
  So, Part 3 of MT can be used if regularity condition holds.
\[ a \frac{f(n)}{b} = 3f(n/4) = 3(n/4) \log_2 (n/4) < \frac{3n \log_2 n}{4} \]. So, regularity condition holds with \( c = 3/4 \).

So, Part 3 of MT applies and \( T(n) = \Theta(n \log n) \).

**A case where MT does not apply:**

**Example 7:** \( T(n) = 2T(n/2) + n \log_2 n \).

Here \( a = 2 \), \( b = 2 \). So, \( \log_b a = 1 \).

\( f(n) = n \log_2 n \).

Thus, \( f(n) \neq O(n^{\log_b a - \epsilon}) \) for any \( \epsilon > 0 \). So, Part 1 of MT does not apply.

Similarly, Part 2 also does not apply.

Also, \( f(n) \neq \Omega(n^{1+\epsilon}) \) for any \( \epsilon > 0 \) (since \( \log_2 n = o(n^\epsilon) \) for every \( \epsilon > 0 \)). So, Part 3 also does not apply.

**Exercise:** Use the iteration or the substitution method to show that for the recurrence of Example 7, the solution is \( T(n) = O(n \log^2 n) \).