Euclid's GCD algorithm

**Def**: Let $a \in \mathbb{Z}$, $b \in \mathbb{Z}$. Then $d$ is a common divisor of $a$ and $b$, if $d \mid a$ and $d \mid b$.

**Def**: Let $a \in \mathbb{Z}$, $b \in \mathbb{Z}$. Then $d$ is the greatest common divisor (gcd) of $a$ and $b$, if
- $d$ is a common divisor of $a$ and $b$
- if $e$ is a common divisor of $a$ and $b$ then $e \leq d$.

$\text{gcd}(a, b)$
for \( k \leftarrow 1 \) to \( \min(a, b) \) do
  if \( k \mid a \) and \( k \mid b \) then
    ans \leftarrow k
  return ans

# iterations : \( \min(a, b) \).

Lemma (Euclid)

Let \( a \in \mathbb{Z}^+ \), \( b \in \mathbb{Z}^+ \), let \( c = a \mod b \).
That is, \( c \) is the remainder when \( a \) is divided by \( b \). Then

\[
gcd(a, b) = gcd(b, c).
\]

Proof: Let \( d = gcd(a, b) \) & let \( e = gcd(b, c) \).
We want to prove that \( d = e \).
\[ a = b \cdot q + c, \text{ for some } m \in \mathbb{Z} \ldots \Rightarrow c = a - b \cdot q. \]
\[ d \leq e \]
- \( d \mid a \), \( d \mid b \)
- \( d \mid c \) (since \( c = a - b \cdot q \))

\[ e \leq d \]
- \( e \mid b \), \( e \mid c \)
- \( e \mid a \)

- \( d \) is a common divisor of \( b \) & \( c \)
- \( e \) is a common divisor of \( a \) & \( b \)

\[ d \leq e \]
\[ e \leq d \]
\[ d = \gcd(a, b) \]
\[ e = \gcd(c) \]

\[ \therefore d \leq e \]

---

**Example.**

\[ \gcd(846, 315) \]
\[ = \gcd(315, 216) \]
\[ = \gcd(216, 99) \]
\[ = \gcd(99, 18) \]

\[ = 846 \mod 315. \]
\[\text{gcd}(18, 9) = 9\]

\[\text{gcd}(a, b)\]

1. \(c \leftarrow a \mod b\)

2. \text{if } c = 0 \text{ then}

3. \quad \text{return } b

4. \text{else}

\quad \text{return } \text{gcd}(b, c).

\underline{Theorem:} Euclid's alg. yields the correct answer for every pair \(a, b\) of integers.

\underline{Proof:} Assume for contradiction that Euclid fails. Among all pairs \(a, b\) integers
that the alg. fails, let \((a, b)\) be the pair s.t. \(a + b\) is the smallest.

Recall that \(c = a \mod b\).

When \(c = 0\), Euclid's alg. returns \(b\), which is the correct answer & hence, since the alg. fails, \(c > 0\).

We have

\[
\begin{align*}
    c &< b \\
    b &\leq a \\
    \hline
    b + c &< a + b
\end{align*}
\]

The alg. returns \(\gcd(b, c)\), which by Euclid's lemma is the right answer. But
Since the alg. fails on \((a, b)\), & we know \(\text{gcd}(a, b) = \text{gcd}(b, c)\), the alg. fails on \((b, c)\) too. This is a contradiction as \(b + c < a + b\) & by our assumption, the alg. must return the correct answer for \((b, c)\).

\[\text{Case I: } b > a\]

\[\text{gcd}(a, b) = \text{gcd}(b, a)\]

\[\text{prev. case.}\]

Lemma: Let \(a \in \mathbb{Z}^+, b \in \mathbb{Z}^+\). Let \(a > b\).

Let \(c = a \mod b\). Then \(c < a/2\).

Proof: \(\text{Case I: } b \leq a/2\)
We know $c < b$ & hence done.

\[ \text{Case II: } \ b > c/2 \]

\[ 10 \bigg) \ 16 \]

when $a$ is divided by $b$, the quotient is 1, remainder $c = a - b < a - c/2 = c/2$.

\[
\begin{align*}
\begin{array}{c}
(a, b) \\
\xrightarrow{\text{2 calls}} \\
(\frac{a}{2}, \frac{b}{2}) \\
\xrightarrow{\text{2 calls}} \\
(\frac{a}{4}, \frac{b}{4}) \cdots \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\frac{b}{2^k} < 1
\end{align*}
\]
\[
\frac{k}{2^k} = \frac{k}{2^k} = \frac{1}{2^{k-1}}
\]

\[
\therefore k > \log_2 b
\]

We must have our answer within \(2 \log_2 b\) calls to Euclid's alg.
Suppose \(b = 2^{100}\).

Euclid's alg: 100 steps.

**Sorting**

**Input:** Array \(A\) of \(n\) integers.

**Objective:** Sort array \(A\) in non-decreasing order.

Initially sort \([A[1..n]]\)

\[\text{for } j \leftarrow 2 \text{ to } n \text{ do } \leftarrow \leq n \text{ times}\]

\[\text{key} \leftarrow A[j] \leq \text{Const}\]
// Insert A[i] into the sorted array A[1..j-1]

i ← j - 1

while (i > 0) and (A[i] > key) do


   i ← i - 1

end while

A[i+1] ← key

Example:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>j</td>
<td>13</td>
<td>4</td>
<td>19</td>
<td>11</td>
</tr>
</tbody>
</table>

j ← 2
key ← 4
i ← 1

<table>
<thead>
<tr>
<th># times</th>
<th>cost</th>
</tr>
</thead>
</table>
Insertion Sort \( A[1...n] \)

\[
\text{for } j = 2 \text{ to } n \text{ do}
\]

\[
\text{key} \leftarrow A[j]
\]

\[
i \leftarrow j - 1
\]

\[
\text{while } (i > 0) \text{ and } (A[i] > \text{key}) \text{ do}
\]

\[
A[i+1] \leftarrow A[i]
\]

\[
i \leftarrow i - 1
\]

\[
A[i+1] \leftarrow \text{key}
\]

Total running time =

\[
c_1 \cdot n + (c_2 + c_3 + c_4)(n-1) + c_4 \cdot \sum_{j=2}^{n} t_j +
\]

\[
(c_5 + c_6) \sum_{j=2}^{n} (t_j - 1).
\]
**Best Case:** Array A is sorted.

In this case, \( t_j = 1 \), \( t_{j-1} = 0 \).

- Total running time

\[
= (c_1 + c_2 + c_3 + c_7) n - (c_2 + c_3 + c_7) \\
+ c_4 \sum_{j=2}^{n} 1^{n-1}
\]

\[
= (c_1 + c_2 + c_3 + c_4 + c_7) n - (c_2 + c_3 + c_4 + c_7)
\]

\[
= a n + b, \text{ for some constants } a, b.
\]

**Worst Case:** A is in decreasing order.

\[
t_j = j
\]
Running time:

\[ C_1 \cdot n + (C_2 + (3 + C_2)) \cdot (n-1) + C_4 \cdot \sum_{j=2}^{n} j \]

\[ + \left( C_5 + C_6 \right) \sum_{j=2}^{n} (j-1) \]

\[ = \left( C_1 + C_2 + (3 + C_2) \right) \cdot n - (C_2 + (3 + C_2)) \]

\[ + C_4 \cdot \left( \frac{m(n+1)}{2} - 1 \right) + (C_5 + C_6) \left( \frac{(n-1) \cdot n}{2} \right) \]

\[ = an^2 + bn + c, \quad \text{for constants } a, b, c. \]

\[ \text{order of growth.} \]
For balance the alg, we get the total \# itv \leq n^2.

Is this analysis tight?

Yes. In the worst case, the total \# iter of the while loop

\[ = 1 + 2 + 3 + \cdots + n-1 \leq n^2. \]

Why worst-case?

1. Allows us to give a guarantee on any input.

2. In many applications, worst case happens more often than expected.
3. Average case analysis.

\[ t_j = \frac{1}{2} \]

\[ \approx n^2 \]

Compute the running time:

- Implement the alg & run it on various inputs.

(i) Need to implement the alg.

(ii) The i/p s that we miss may be indicative of the worst case running time of the alg.

(iii) To compare two alg s, we have to implement them using the same
software & hardware environment.

We will analyze algo analytically.

**RAM model of computation.**

- Each high-level operation takes **const time (unit time)**
  - arithmetic operation, comparison, assignment, 
    - call, return

- no memory hierarchy
  - all memory access take const time.