Euclid's GCD algorithm

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

**Def**: Let $a \in \mathbb{Z}$ and $b \in \mathbb{Z}$. Then $d$ is the **greatest common divisor** 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 $\text{min}(a, b)$ do
  if $k | a$ and $k | b$ then
ans \leq k

\text{return ans}

The above alg will take \text{min} (n, b) iterations in the worst case.

\textbf{Lemma (Euclid)} Let a \& b be positive integers.
Let c = a \mod b (remainder when a is divided by b). Then
\[ \gcd(a, b) = \gcd(b, c) \]

\textbf{Proof:} Let \( d = \gcd(a, b), \ e = \gcd(b, c) \).
Want to show that \( d = e \).
\[ d \leq e \]
\[ -d \mid a \text{ and } d \mid b \]
\[ -d \mid c \quad (\because c = a - bq) \]
\[ -e \mid b \text{ and } e \mid c \]
\[ -e \mid a \quad (a = bq + c, \text{ for some } q) \]
\[ d \mid b, d \mid c. \]

- \[ e = \gcd(b, c) \]

- \[ d \leq e \]

- \[ e \text{ is a common divisor of } a \& b \]

- \[ d = \gcd(a, b) \]

- \[ e \leq d. \]

\[
\gcd(846, 315) = \gcd(315, 216) = \gcd(216, 99) = \gcd(99, 18) = \gcd(18, 9) = 9.\
\]

\[ \gcd(a, b) \]

1. \[ c = a \mod b \]
2. if $c = 0$ then

3. return $b$

4. else

    return $\gcd(b, c)$.

**Theorem**: Euclid's GCD algorithm returns the correct answer.

**Proof**: Assume for contradiction that Euclid's algorithm fails. Among all pairs of integers on which Euclid fails, let $(a, b)$ be the pair s.t. $a + b$ is the smallest.

Can A: $a \geq b$

c = a \mod b.

Can I: $c = 0$

The algorithm returns $b$, which is the correct answer. Since we know the algorithm fails on $(a, b)$, $c > 0$. 

\[\text{Can II: } c > 0.\]
\[c < b \quad \underline{+} \quad b \leq a\]
\[\underline{\quad b + c < a + b}\]

Our alg. returns \(\gcd(b, c)\), which is correct (by Euclid's lemma). Since our algorithm gives an incorrect answer for \(\gcd(a, b)\), it must give an incorrect answer for \(\gcd(b, c)\). This is a contradiction as by our assumption, \(\gcd(b, c)\) must be correct.

\[\text{Can B: } a < b\]
In this case, \( \gcd(a, b) = \gcd(b, c) \)

Now we are back to Case A.

Lemma: Let \( a \in \mathbb{Z}^+ \) and \( b \in \mathbb{Z}^+ \) and let \( a > b \). Let \( c = a \mod b \). Then \( c < a/2 \).

Proof:

Case I: \( b \leq a/2 \)

Since \( c < b \), we are done.

Case II: \( b > a/2 \)

In this case, \( c = a - b \leq a - a/2 = a/2 \).

\[ 9 \left\lfloor \frac{1}{16} \right\rfloor \]
\[ \frac{a}{2}, \frac{b}{2} \]

\[ (a, b) \xrightarrow{2 \text{ calls}} (\frac{a}{2}, \frac{b}{2}) \xrightarrow{2 \text{ calls}} (\frac{a}{4}, \frac{b}{4}) \cdots \]

No more calls to Euclid's alg when

\[ \frac{b}{2^k} \leq 1. \]

\[ \Rightarrow k > \log_2 b \]

\[ \therefore \text{max \# recursive function calls before we get the ans } \leq 2 \cdot \log_2 b. \]
Sorty.

Input: array A of n integers.

Objective: Sort the array A.

Insertionsort (A[1..n])

for j ← 2 to n do
    key ← A[j]
    // insert A[j] into the sorted array
    // A[1..j-1]
    i ← j - 1
    while (i > 0) and (A[i] > key) do
        → A[i+1] ← A[i]
        i ← i - 1
\( i \leftarrow i-1 \)

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

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
13 & 14 & 19 & 11
\end{array}
\]

\( j = 2 \to 3 \)

\( \text{key} = 4 \to 19 \)

\( i = 1 \to 2 \)

\( \boxed{\text{sorted}} \)

\[
\begin{array}{cccc}
1 & 4 & 11 & 13 & 19
\end{array}
\]

**Insertion Sort \([A[1..n]]\)**

\[
\text{for } j \leftarrow 2 \text{ to } n \text{ do} \\
\text{key} \leftarrow A[j] \\
i \leftarrow j - 1
\]

\[
\begin{array}{c|c|c|c|c}
\# \text{times} & \text{cost} \\
\hline
n & c_1 \\
n-1 & c_2 \\
n-1 & c_3 \\
n & \checkmark
\end{array}
\]
while ($i > 0$) and ($A[i] > key$) do
\[
A[i+1] \leftarrow A[i],
\]
\[
i \leftarrow i - 1
\]
\[
A[i+1] \leftarrow key
\]

Total running time:
\[
C_1 \cdot n + (C_2 + C_3 + C_7) \cdot n - 1 + C_4 \cdot \sum_{j=2}^{n} t_j + C_5 \cdot \sum_{j=2}^{n} (t_j - 1) + C_6 \cdot \sum_{j=2}^{n} (t_j - 1)
\]

Best Case: Input array $A$ is sorted.
\[
t_j = 1, \forall j.
\]

\[
\therefore \text{Running time} = (C_1 + (C_2 + C_3 + C_7) \cdot n - (C_2 + C_3 + C_7) + C_4 \cdot \sum_{j=2}^{n} 1
\]
\[(C_1 + C_2 + C_3 + C_4 + C_5)n - (C_2 + C_3 + C_4)\cdot n = a \cdot n + b, \text{ for some constants } a \& b.\]

\underline{Worst Case: A is in decreasing order.}

\[t_j = ?\]

\[t_j = j.\]

**Running Time:**

\[= (C_1 + C_2 + C_3 + C_4) \cdot n - (C_2 + C_3 + C_4)\cdot n + C_4 \cdot \sum_{j=2}^{n} j + (C_5 + C_6) \sum_{j=2}^{n} (j-1)\]

\[= (C_1 + C_2 + C_3 + C_2) \cdot n - (C_2 + C_3 + C_4)\cdot n + C_4 \cdot \left(\frac{n(n+1)}{2} - 1\right) + (C_5 + C_6) \cdot \left(\frac{(n-1) \cdot n}{2}\right)\]
\[ \sum (c_1 + c_2 + c_3 + c_4) \cdot n - \left( c_2 + c_3 + c_4 \right) \\
+ c_4 \cdot \left( \frac{n^2}{2} + \frac{n}{2} - 1 \right) + (c_5 + c_6) \cdot \left( \frac{n^2}{2} - \frac{n}{2} \right) \]

\[ \equiv \quad a n^2 + b n + c , \quad \text{const} \ a, b, c. \]

- Running time as a function of \( n \), the size of the i/p.
- Best case running time \( IS \approx n \).
- Worst case \( IS \approx n^2 \).

Eye-balling the algo, we get that the running time taken \( \leq n^2 \) steps.
Is this upper-bound tight?
Yes
1 + 2 + 3 + \cdots + n-1 \approx \frac{n^2}{2}.

We will consider worst-case analysis of algorithms, unless specified otherwise.

- Guaranteed upper-bound on the running time of the alg.
- Worst-case happens more often than we may expect.
- The performance of the alg. in any case is as bad as in the worst case.

In InsertionSort, on avg. \( t_i = i/2 \).

\approx \frac{n^2}{2}. 
Running time:

- implement the alg. & run it on various inputs.
- actually have to implement every alg. to analyze it.
- we can only check the alg. on some inputs. We may leave out inputs that may be indicative of the worst case running time of the alg.
- to compare two algs., we need to implement them using the same software & hardware environment.

For the rest of the course, we will
analyze algo analytically.

**RAM model of computation.**

- Each **high-level operation** takes a unit step (constant time).

- High-level operation: assignment, fn. call, return stmt, comparison, arithmetic...

- No memory hierarchy.

  - All memory accesses take constant time.