Analysis of algorithms -- the framework

I. Establish problem size, and classify the inputs by problem size.
Example 1:
  greatest common divisor of non-negative integer numbers $a$ and $b$
  size: $a + b$, or, $\min(a, b)$, or, the number of bits of $a$ etc.
  Usual practice for arithmetic problems: the number of bits
Example 2:
  find $k^{th}$ largest object in unsorted array of $n$ objects of a comparable type
  size: the length of the array? for large size objects, the total number of bits?

II. Establish/assume the cost for the prevalent operations
  In example 1: finding remainder, in example 2: comparing objects

III. Choose the type of analysis: worst case for a size class, best case, average case. Typical is worst case and average case.
  Average case analysis is rather rare. Some further distinctions we will see later.
Analysis of algorithms -- simple counting

We may have simple loops and nested loops. Analysis reduces to multiplications and summations.

Example A:

```c
for (i = 0; i < n1; i++)
    for (j = 0; i < n3; i++)
        for (C[i][j] = k = 0; i < n2; i++)
            C[i][j] += A[i][k] * B[k][j];
```

This is matrix multiplication code:

- A is an $n1 \times n2$ matrix
- B is an $n2 \times n3$ matrix
- C = A×B is an $n1 \times n3$ matrix

we use $O(n1 \times n2 \times n3)$ operations
Example B:

```c
all_different = 1;
for (i = 1; i < n; i++)
    for (j = 0; j < i; i++)
        if (A[i] == A[j])
            all_different = 0,
            i = j = n;
```

This code checks uniqueness of elements in array A[n].

We use \( \sum_{i=1}^{n} i = \Theta(n^2) \) operations.

This is a “brute force” algorithm, we check all \( C(n, 2) \) pairs.

A brute force algorithm is perhaps “stupid”, but it may be a part of a clever algorithm: we may “chop” a problem instance into small pieces, and solve the pieces by brute force. We will see several applications of that.
We can use a “clock” to estimate the number of iteration that a loop performs, each time the loop is executed the clock ticks, which means that it changes value by 1, or at least 1.

Example C:
```c
for (i = 0; i < n; i++)
    for (j = i; j > 0; j /= 2)
        small task X;
```

the “clock” of inner loop: \(\log_2 j\),
it ticks from \(\log_2 i\) down to 0, discrepancy below 1

we use \(\Theta(n + \sum_{i=2}^{n} \log_2 i)\) operations

We need an estimate for \(\sum_{i=2}^{n} \log_2 i\);

from above, \(\sum_{i=2}^{n} \log_2 n = n \log_2 n\),

from below, \(\sum_{i=n/2}^{n} \log_2 n/2 = (n \log_2 n - 1)/2\).

thus \(\Theta(n \log n)\).
Example D:

```plaintext
for (i = 0; i < n; i++)
    for (j = i; j < n; j *= 2)
        small task Y;
```

the “clock” of inner loop: \( \log_2 j \) ticks from \( \log_2 i \) up to \( \log_2 n \)

we use \( \Theta(n + \sum_{i=2}^{n} (\log_2 n - \log_2 i)) \) operations

We need an estimate for \( T(n) = \sum_{i=2}^{n} (\log_2 n - \log_2 i) \).

\[
T(n) = \sum_{i=1}^{n} (\log_2 n - \log_2 i) =
\]

\[
\sum_{i=1}^{n/2} (\log_2 n - \log_2 n/2 + \log_2 n/2 - \log_2 i) + \sum_{i=n/2}^{n} (\log_2 n - \log_2 i) =
\]

\[
\sum_{i=1}^{n/2} (\log_2 n - \log_2 n/2 + \log_2 n/2 - \log_2 i) + \sum_{i=n/2}^{n} (\log_2 n - \log_2 i) =
\]

\[
cn + T(n/2) = 2cn
\]

Here we used a recurrence relation, more on that in the next lecture.
Example E: Euclid algorithm

```java
while (b != 0)
    c = a % b,
    b = a,
    a = b;
```

The body of the loop changes the pair \((a, b)\) from \((b_i, b_{i-1})\) into \((b_{i-1}, b_{i-2})\) where \(b_{i-2} = b_i \mod b_{i-1}\).

Because \(b_{i-2} = b_i \mod b_{i-1}\), we have \(b_i = m \cdot b_{i-1} + b_{i-2}\) for some \(m \geq 1\), so

\[
\begin{align*}
    b_i &\geq b_{i-1} + b_{i-2}, \text{ thus also } \\
    b_{i-1} &\geq b_{i-2} + b_{i-3}, \text{ add together and simplify } \\
    b_i &\geq 2 \cdot b_{i-2} + b_{i-3}.
\end{align*}
\]

This shows that \(b_i > 2b_{i-2}\), hence \(\log_2 b\) is a clock that ticks at least once for every two iterations of the loop. This shows that the number of iterations is at most \(2 \log_2 b\), so the running time is \(O(n)\) if \(n\), the input size, is the number of bits of \(b\).
Moreover, the relationship we got is Fibonacci recurrence if $m = 1$:
$f_0 = 0, \quad f_1 = 1, \quad f_i = f_{i-1} + f_{i-2}$ for $i > 1$.

One can see that if we start with $(a, b) = (f_n, f_{n-1})$ then Euclid algorithm performs exactly $n$ iterations, so for this special input we have an exact clock.

Because we can show that $f_{n-1}$ has at most $n$ bits, we got a lower bound $\Theta(n)$ for the worst case for inputs with $n$ bits.