COL100: Introduction to Computer Science

6.1: Efficiency analysis

Correctness and efficiency

Correctness: Does the algorithm get to the right answer?

Efficiency: How much work does it take to get there?

- How much *time*?
- How much *space*?

$$
\begin{array}{r}
8\,2\,4\,7 \\
\times\ 5\,8\,3\,1 \\
\hline
8\,2\,4\,7 \\
2\,4\,7\,4\,1 \\
6\,5\,9\,7\,6 \\
+\ 4\,1\,2\,3\,5 \\
\hline
4\,8\,0\,8\,8\,2\,5\,7
\end{array}
$$

Example: *factorial*

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

$$\begin{aligned} & factorial(3) \\ &= factorial(2) \times 3 \\ &= (factorial(1) \times 2) \times 3 \\ &= ((factorial(0) \times 1) \times 2) \times 3 \\ &= ((1 \times 1) \times 2) \times 3 \\ &= (1 \times 2) \times 3 \\ &= 2 \times 3 \\ &= 6 \end{aligned}$$

Suppose each multiplication takes the same amount of time.
(True when multiplying `int`s!)

Total time
$\propto$ number of multiplications
= ?

$$\mathit{factorial}(n) = \begin{cases} 1 & \text{if } n = 0, \\ \mathit{factorial}(n-1) \times n & \text{otherwise} \end{cases}$$

Let $T(n)$ = #multiplications for computing *factorial*(n).

$$T(n) = \begin{cases} 0 & \text{if } n = 0, \\ T(n-1) + 1 & \text{otherwise.} \end{cases}$$

By induction, $T(n) = n$.

We call this the *time complexity* of this algorithm.

$$factorial(n) = \begin{cases} 1 & \text{if } n = 0, \\ factorial(n-1) \times n & \text{otherwise} \end{cases}$$

We also need space to:

- keep track of deferred operations
- or, stack up frames for function calls

Similarly, show that #frames = n + 1: *space complexity*

$$\begin{aligned} & factorial(3) \\ &= factorial(2) \times 3 \\ &= (factorial(1) \times 2) \times 3 \\ &= ((factorial(0) \times 1) \times 2) \times 3 \\ &= ((1 \times 1) \times 2) \times 3 \\ &= (1 \times 2) \times 3 \\ &= 2 \times 3 \\ &= 6 \end{aligned}$$

$\boxed{factorial(3)}$

$\boxed{factorial(2)}$

$\boxed{factorial(1)}$

$\boxed{factorial(0)}$

Algorithms and complexity

Time and space complexity depend on the *algorithm*, not the *problem*.

$$power(x, n) = \begin{cases} 1 & \text{if } n = 0, \\ x \cdot power(x, n-1) & \text{otherwise.} \end{cases}$$

$$fastPower(x, n) = \begin{cases} 1 & \text{if } n = 0, \\ fastPower(x^2, \lfloor n/2 \rfloor) & \text{if } n \text{ is even,} \\ x \cdot fastPower(x^2, \lfloor n/2 \rfloor) & \text{if } n \text{ is odd.} \end{cases}$$

power is similar to *factorial*: *power*(x, n) requires n multiplications.

How many multiplications does *fastPower*(x, n) require?

$$fastPower(x, n) = \begin{cases} 1 & \text{if } n = 0, \\ fastPower(x^2, \lfloor n/2 \rfloor) & \text{if } n \text{ is even}, \\ x \cdot fastPower(x^2, \lfloor n/2 \rfloor) & \text{if } n \text{ is odd}. \end{cases}$$

Clearly, $T(0) = 0$, and $T(1) = 2 + T(0) = 2$.

For $n > 1$, consider simplest case: $n = 2^k$.

$$\begin{aligned} T(2^k) &= 1 + T(2^{k-1}) \\ &= 2 + T(2^{k-2}) \\ &\vdots \\ &= k + T(2^0) \\ &= k + 2. \end{aligned}$$

So $T(n) = \log_2 n + 2$ when $n = 2^k$.

Bigger differences

Consider the problem of finding the determinant of an $n \times n$ matrix.
(No SML implementation for now!)

Schoolbook algorithm:

$$\det\left(\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}\right) = a_{11}\det\left(\begin{bmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix}\right) - a_{12}\det\left(\begin{bmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{bmatrix}\right) + a_{13}\det\left(\begin{bmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix}\right)$$

Bigger differences

Consider the problem of finding the determinant of an $n \times n$ matrix.
(No SML implementation for now!)

Schoolbook algorithm:

$$\det([\ a\ ]) = a,$$
$$\det(A) = a_{11} \det(A_{11}) - a_{12} \det(A_{12}) + \cdots \pm a_{1n} \det(A_{1n}).$$

Here the A_{ij} are $(n-1) \times (n-1)$ matrices obtained by deleting a row and column.

What is the time complexity?

$$\det([\,a\,]) = a,$$
$$\det(A) = a_{11}\det(A_{11}) - a_{12}\det(A_{12}) + \cdots \pm a_{1n}\det(A_{1n}).$$

By induction, for $n \geq 2$ this algorithm requires at least $n!$ multiplications.

Base case: $n = 2$. Then $\det(A) = a_{11}\det([\,a_{22}\,]) - a_{12}\det([\,a_{21}\,])$ which requires 2 multiplications.

Induction hypothesis: $T(n-1) \geq (n-1)!$

Induction step: $T(n) = n\,T(n-1) + n \geq n\,(n-1)! + n \geq n!$

$$\det([\,a\,]) = a,$$
$$\det(A) = a_{11}\det(A_{11}) - a_{12}\det(A_{12}) + \cdots \pm a_{1n}\det(A_{1n}).$$

This algorithm requires at least $n!$ multiplications (and lots of additions and subtractions as well).

At 10^9 multiplications per second,

- $n = 10$ requires 3.6 ms,
- $n = 15$ requires 22 minutes,
- $n = 20$ requires 77 years!

On a supercomputer (10^{12} ops/sec): n = 20 → 28 days, n = 21 → 1.6 years

Gaussian elimination* can compute the determinant in less than $n^3 + 2n^2$ arithmetic operations.

Now even on a 30-year-old computer (10^6 ops/sec),

- n = 20 → 9 ms,
- n = 100 → 1 sec,
- n = 30,000 → < 1 year!

Moral: Constants (10^6, 10^9, 10^{12}) don't matter as much as the rate of growth of computational complexity.

* We may not cover how Gaussian elimination works in this course.

Afterwards

- Read Sections 3.6.2 and 3.6.3 of the notes.
- Let n be the largest size of matrices whose determinant you are able to compute in a given amount of time (say 24 hours). If you buy a 1000× faster machine, approximately what size of matrices can you process in the same amount of time? Give an answer for both algorithms, assuming $T(n) = n!$ for one and $T(n) = n^3$ for the other.
- Show that for any n, *fastPower*(x, n) requires at most $2 \lceil \log_2 n \rceil + c$ multiplications for some constant c.
- Evaluate the number of function calls and the space complexity of *fastPower*(x, n).