Analyzing Algorithms

Definitions: An algorithm is a finite answer to an infinite question.
- Problem: specification of infinite set of inputs
  - specification of corresponding outputs
  [Note: can be difficult in practice to distinguish
  infinite from large-finite, e.g. chess/Rubik's cube —
  finite but large enough to be very hard & interesting]

Algorithm: well defined computational procedure
to go from any input to corresponding output.
For our purposes — described in pseudo-code.

Analyze an Algorithm — measure time and space
used by the algorithm as a function of input size
— measured not by running the program, but by using
an abstract model of computing.

Models of Computation
- specify the elementary computations out of which
  algorithms are built.
- specify measure of time, space, input size.
Bottom line: model should reflect (but simplify) reality.
Models of Computing

General purpose model [can do anything a real computer can]

I. pseudo-code

(a) each line takes 1 time step.

caution:

- some pseudo-code lines are too powerful
  - e.g. initializing array is 1 line but
    should cost \( n \) time steps (\( n = \text{length of array} \))
- numbers can grow too large

example:

```python
function Fibonacci (n)
    i = 0, j = 1
    for k = 1 to n
        j = i + j
        i = j - i (old value of j)
    \} 2n steps
    return j
```

0, 1, 1, 2, 3, 5, 8, ...

But the numbers grow so quickly that \( 2n \) steps does not reflect reality: \( n = 47 \) causes overflow on 32 bit words.
(b) count bit cost

\[ \text{size of number } a = \# \text{ bits in binary representation} \approx \log_2 a \llbracket \log a + 1 \rrbracket \]

\[ a \times b \text{ takes } O((\log a)(\log b)^{0.59}) \text{ using efficient multiplication} \]

(we'll see the alg. later on)

school method takes \( O((\log a)(\log b)) \)

---

More formal than pseudo code:

II. \( \text{RAM} = \text{Random Access Machine} \) abstracts assembly lang.

- "random access" means we can access memory location \( i \) instantly \( \text{(not like tape or Turing machine)} \)

how to charge for "size" of a memory location:

unit cost (like (a) above) \( \text{— too powerful} \)

bit cost (like (b) above) \( \text{— too weak} \)

good compromise?

word \( \text{RAM} \) — each memory location holds one word

and assume \( \# \text{ bits in word} = \Theta(\log n) \)

\( n \) = input size

- a bit weird if hardware changes to fit input!

- but, really, if input is array \( A[i..n] \) we want an index \( i \) to fit in a word.

Note: sorting can be faster than \( O(n \log n) \) on this model.
other general purpose models

III Circuit family - abstracts hardware circuitry

IV Turing machine - abstracts human computer working with pencil and paper

Note: time to access memory location $i$ is proportional to $i$.

Special purpose or “structured” models of computing

* comparison-based model for sorting $\Omega(n \log n)$ lower bound.

* arithmetic model

Our model of Computing

word RAM

=pseudo-code and try to be realistic about what are elementary operations

Challenge: If you have a RAM model and allow unlimited # bits for each memory location and charge 1 for arithmetic and shifts, how can you “cheat” and sort in $O(n)$ steps?

Hint: Do all $n^2$ pairwise comparisons in 1 subtraction
Running Time of an Algorithm

- We think of this as more accurately representing running time, which depends on input.
- \( T_A(I) \) - running time of algorithm \( A \) on input \( I \)
- We expect running time to increase as size of input increases.
- Simplify by expressing run time as a function of input size.
- [Note: model of computing must say how to count input size]
- For given size \( n \), there are various inputs (a finite number).
- How do we combine running times to one number?

Worst case running time

[the standard unless otherwise specified]
\[ T_a(n) = \max \{ T_a(I) : I \text{ an input of size } n \} \]
[leave off \( A \) if it's understood]

Why worst-case?
- We want an absolute guarantee.
- Alternative of average case is hard to analyze and depends on assumptions about input distribution (uniform?)
This might come up later in the course (but ignore for now)

When an algorithm uses random numbers we use expected run-time

\[ T_{\text{E}}(I) = \text{expected run-time on input } I \]
\[ \text{(depends on random nos. used by } A \text{)} \]

Still use worst case over inputs

\[ T_{\text{E}}(n) = \max_{\text{inputs of size } n} T_{\text{E}}(I) \]

[we might discuss this later on]
Asymptotic Analysis of Algorithms

\( T(n) \) = worst case run time of algorithm as a function of input size

We want \( T(n) \) to be

- simple to express, e.g. \( n^2 \)
- machine independent, thus ignore multiplicative factors (one machine might be twice as fast), thus ignore lower order terms (\( n+5 \leq 2n, \ n \geq 5 \))

**Definition** Big Oh notation

Let \( f(n) \), \( g(n) \) be functions from \( \mathbb{N} \) to \( \mathbb{R}^+ \)

\( f(n) \) is \( O(g(n)) \) “order \( g(n) \)” “big oh of \( g(n) \)”

if \( \exists \) constants \( c > 0 \) and \( n_0 \) s.t.

\( f(n) \leq c \cdot g(n) \) for all \( n > n_0 \).

(we say \( f \) is bounded by a constant times \( g \) for \( n \)

sufficiently large — this is what asymptotic means)

**Notation** \( f(n) \leq O(g(n)) \) [CLRS writes \( f(n) = O(g(n)) \)]

Big oh gives an upper bound. Examples:

- \( T(n) = 5n^2 + 3n + 25 \) is \( O(n^2) \)
- \( 10^{100} \cdot n \) is \( O(n) \)
- \( \log n \) is \( O(n) \) but \( n \) is not \( O(\log n) \)
- \( 2^{n+1} \) is \( O(2^n) \)
- \( ? \) \( (n+1)! \) is \( O(n!) \) ? \( \text{No.} \)