Solving Interesting Problems

1. Create algorithms & data structures to solve problems.
   - Alg. efficiency depends on data structure. Two parts of the same idea.
   - Buggy algorithms are worthless!

2. Prove algorithms work.
   - Running time, space needed, simplicity, …

3. Examine properties of algorithms.
   - Abstract vs. Concrete Data Structures

Abstract vs. Concrete Data Structures

**Abstract:**
- What can it do?
  - I.e., its interface – what operations?

**Concrete:**
- How is it implemented?
  - How efficient?

Familiar idea, but... people & algorithm textbooks often don’t distinguish.

A Quick Example of Interesting Results

Roughly how much time for each?
1. Find shortest path between two points in graph.
2. Find shortest path between each pair of points in graph.
3. Find longest path between two points in graph.

? ?

1. O(V \times E)
2. O(V \times E)
3. No known polynomial-time algorithm.

Mutability

**Mutable:**
- Operations can change data structure.
  - enqueue(elt, Q) modifies Q to have new elt.

**Immutable:**
- Operations can’t change data structure.
  - enqueue(elt, Q) creates & returns Q’ to have new elt. Q unchanged.

Most algorithm textbooks & this course: mostly mutable.
Outline of Semester

• Finish course overview – extended example
• Math background – review & beyond
• Algorithms & data structures
  Techniques, as needed:
  • Randomization
  • Probabilistic analysis
  • Amortized analysis
  • Dynamic programming
• Really hard problems

Extended Example: Multiplication

Illustrates some of course’s interesting ideas.
Just an overview – not too much detail yet.

How to multiply two integers: x•y.
Common & familiar. Simple?

Suggested algorithms & their efficiency?

Multiplication Algorithm 1

Single basic machine operation, O(1) time.

Problem with this answer?

Only applies to bounded-sized integers.
Instead, explicitly assume unbounded-sized integers.

Multiplication Algorithm 2

x • y = x+…+x = y+…+y

y copies  x copies

How long for each addition?

• Grade-school algorithm takes time \( \propto \) result length.
  For simplicity, assume x,y have same length.
  Length n = log_2 x (choice of base 2 is arbitrary)
  O(n)

Back to multiplication:
• O(n \times x) = O(n \times 2^n)

Multiplication Algorithm 3

Grade-school “long-hand” algorithm.

O(n \times n + n \times n) = O(n^2)

Much better!

Multiplication Algorithm 4: Part 1

Karatsuba, 1962

Break the bit strings of x,y into halves.

x = a \times 2^{n/2} + b = a << n/2 + b
y = c \times 2^{n/2} + d = c << n/2 + d

xy = ac << n + (ad+bc) << n/2 + bd

Compute 4 subproducts recursively.
Divide-and-conquer.
Multiplication Algorithm 4: Part 1

How long does this take?

Form recurrence equation:

\[
T(n) = \begin{cases} 
  k & n = 1 \\
  4 \times T\left(\frac{n}{2}\right) + k \times n & n > 1 
\end{cases}
\]

Solve recurrence equation.

- How? Discussed next week.
- \(T(n) = O(n^2)\) — No better than previous.

\(4\) subproducts + additions & shifts

\[k = \text{arbitrary constant to fill in for the details}\]

Multiplication Algorithm 4: Part 2

How long does this take?

Regroup (very non-obvious step!):

\[
\begin{align*}
  u &= (a+b) \times (c+d) \\
  v &= ac \\
  w &= bd \\
  xy &= v << n + (u - v - w) << n/2 + w
\end{align*}
\]

Only 3 subproducts! But more additions & shifts.

Previous:

\[
\begin{align*}
  xy &= ac << n + (ad + bc) << n/2 + bd
\end{align*}
\]

Multiplication Algorithms 5—8

**Toom-Cook.** 1963, 1966: \(O(n^{1+\epsilon})\)
- Generalizes both long-hand & Karatsuba
- Based upon polynomial multiplication & interpolation.

**FFT-based:**
Karp: \(O(n \log^2 n)\)

**Schoenhage & Strassen.** 1971: \(O(n \log n \log \log n)\)

Fürer, 2007: \(O(n \log n 2^{O(\log^* n)})\)

Approaching the conjectured \(\Omega(n \log n)\) lower bound.

Multiplication Algorithms 9, ...

Use parallelism.

Even serial processors use some bit-level parallelism. Divide-&-conquer & FFT algorithms easily parallelized.
Summary

Asymptotically-faster
often \(\rightarrow\)
more complicated, higher constant factors, & slower for typical input sizes.

Good ideas lead to more good ideas
Karatsuba generalizes to Strassen’s matrix multiplication [CLRS] 4.2 & Toom-Cook.
Toom-Cook & FFT generalize to polynomial multiplication.

Algorithm Analysis Summary

2. Characterize the input size.
   - Often difficult, of course
4. Prove algorithm correctness.
   - Necessary!
5. Determine its resource usage.
   - Often via recurrence equations
   - Compare algorithms
   - Decide which algorithm suitable for given application
   - Guide the search for better algorithms

We almost missed an important detail that would have produced incorrect analysis.