CS 473 Algorithms: Lecture 1 (2022-01-18)

**Logistics:**
- PSER 0 on Friday, Spring, due 2 weeks after
- Signup: pizza
- Gradescope

**Today:**
- Introduction
- Divide and conquer
  - Integer Multiplication

**Lecture:** TR 14-15:15, Silva 1404

**Staff:**
- Instructor: Prof. Michael A. Forbes (mforbes)
- TAs:
  - Christiaan Hanraad (Hanraad28)
  - Payal Kulkarni (payjark2)

**Review:**
- Canvas, Canvas.illinois.edu/cs 473/sp 2021
- Full course details
- Lecture notes
- Textbooks
- Lecture notes: Suggested reader list
- Exam

**Outline:**
- Course care
- Pre-discussion
- Can handle soft
  - Course work submission/return/registry
  - Gradescope
  - Course materials:
    - Lecture: in-person exam at first week
    - Lecture notes
    - Slides/handouts
    - Videos
    - Textbooks: suggested reader list
      - Each home
      - 90% free kindle
- TAs:
  - "M" for Math
  - "A" for Algorithms
- psrs (25%) -
  - 12 psrs, 3 per bar each
  - on the first
  - [no] late psrs

- GPS
  - [True] base psrs
  - [False] base psrs

- peer group
  - peer done individually

- peer N, N ≥ 1 can be
close in ages ≤ 3

- intensity

- exam (45%)
  - 2 x 22.5%
  - non cumulative

- final (30%)
  - cumulative

prep: final:
- CS 173 (discrete maths)
  - CS 225 (data structures)
  - CS 374 (algo, models of computation)

informed:
- final proofs
- basic algo
  - data structures
  - graph algo
  - probability
  - models of computation
Q: Why does Google work so well?
A: Because of algorithms.

Q: How does Google do it?
A: Algorithms!

Q: Can algorithms do everything?
A: No.
divide and conquer

idea:

reasoning:

...
Given multipliers of two n-bit numbers, \( a = a_1 \cdot 2^{n_1} + a_0 \) and \( b = b_1 \cdot 2^{n_2} + b_0 \), we can compute \( a \cdot b \) as follows:

\[
a \cdot b = (a_1 \cdot 2^{n_2} + a_0)(b_1 \cdot 2^{n_2} + b_0)
\]

This multiplication can be done in \( O(n) \) time by shifting and multiplying, then

\[
a \cdot b = a_1 b_1 \cdot 2^{n_1+n_2} + (a_0 b_1 + a_1 b_0) \cdot 2^{n_1+n_2} + a_0 b_0
\]

If the multipliers are n/2-bit numbers, divide \( 4 \cdot T(n/2) + O(n) \) for \( T(n) \) = worst case time of any n-bit mul.
Theorem 5.4.2

Let

\[
\begin{align*}
\alpha &= a_1 \cdot 2^{n/2} + a_0 \\
\beta &= b_1 \cdot 2^{n/2} + b_0 \\
\gamma &= c_1 \cdot 2^{n/2} + c_0 \\
\alpha \cdot \beta &= a_1 b_1 \cdot 2^n + (a_0 b_1 + b_0 a_1) \cdot 2^{n/2} + a_0 b_0
\end{align*}
\]

**Claim:** It takes 4 recursive calls to compute 3 numbers \(a_1 b_1, a_0 b_1 + b_0 a_1, a_0 b_0\).

**Idea:** Use 3

\[
\begin{align*}
\alpha &= (a_1 - a_0) (b_1 - b_0) = a_1 b_1 + a_0 b_0 - (a_0 b_1 + a_1 b_0)
\end{align*}
\]

Thus, this is a \(\gamma\)-bit multiplication (after adjusting sign).

**Also:**

1. Recursively compute \(a_1 b_1\).
2. Compute \(a_0 b_1 + a_1 b_0\).
3. Compute \(a_1 b_0\).

Correctness: Clear.

**Complexity:**

\[
T(n) \leq 3 \cdot T(\frac{n}{2}) + O(n) \leq O(n \log_2 3)
\]
- $O(n^2)$ conjectured necessary by
  Kolmogorov 68
- Kacser 68 disprove this
- Todd 63/Coakley 66:
  split n-bit number into $k \geq 2$ parts
  \[ \text{multiplication in } n \cdot 1 + O(1/k) \]
  \[ \text{for } k \leq O(1) \]
- Gauss 1801, Cooley-Tukey 65, Schönhage-Strassen 71:
  multiplication via Fast Fourier transform
  \[ \text{in } O(n \log n \log \log n) \text{ steps} \]
- Föuer 07: $O(n \log n \cdot 1 + O(\log n))$
- Harvey-Van Der Hoeven 19: $O(n \log n)$