Lecture 22: Future Algorithms Topics

More Where 6.006 Came From

- Natural followups/extensions to 6.006:
  - 6.046: Design and Analysis of Algorithms
  - 6.851: Advanced Data Structures
  - 6.854: Advanced Algorithms
  - 6.855: Network Optimization (algorithms on graphs beyond shortest paths)
- Compare to 6.006:
  - Much more to say about sets and sequences, graphs, DP
  - More design, less reduction: greedy, divide and conquer, etc.
  - More tools for analysis, e.g., amortized runtimes
  - More with randomness: how it can help; tools to analyze expected or with-high-probability runtimes
- Example: Beating AVL Trees!
  - Ordered set interface: insert, delete, find, predecessor, successor
  - Word-size w, possible keys are $\{0, 1, \ldots, 2^w - 1 = u - 1\}$
  - AVL gets $O(\log n)$ for each op
  - (Without ordered queries, hash tables get $O(1)$ expected)
  - Try higher branching factor! Takes longer to find correct child to step to, so it doesn't help on its own. How to compensate?
  - Idea: Exploit w-bit word-size with bit manipulation tricks!
    * Fusion Trees can handle $w^{1/5}$ branching factor and still find the correct child in $O(1)$ time, giving $O(\log n / \log \log u)$ time.
  - Different idea: Branching factor so high ($\sqrt{u} = 2^{w/2}$), we need a data structure just to deal with children!
    * van Emde Boas trees get $O(\log \log u)$ per op
  - Min of these: $O(\sqrt{\log n / \log \log n})$ per op

Parallel Algorithms

- 6.816: Multicore Programming. Multiple processors/cores working on shared memory
- 6.852: Distributed Algorithms. Interconnected processors communicating via messages, e.g., cluster computing (what about missed messages? processor failures? malicious errors?)
- Very rare that k-fold parallelism leads to perfect k-fold speedups.
- New challenges! Race conditions, deadlocks, ...
- Harder to reason about!
- Dining Philosopher's Problem
- Multicore example: Batcher Odd-Even Mergesort, $O(\log^2 n)$ time with $\Theta(n)$ processors
  - If $A[:n/2]$ and $A[n/2:n]$ are sorted, here's a merge algorithm:
    - Recursively merge $A[::2]$ and $A[1::2]$ in-place
    - For $1 \leq i \leq n/2 - 1$, swap $A[2i-1]$ with $A[2i]$ if necessary.
    - Key observation: this last step can be done in $O(1)$ time by $n/2 - 1$ processors!
- Distributed example: Collaboratively compute a BFS tree
  - Root sends "1" to its neighbors. When a node receives new lowest value k, it sends $k+1$ to its neighbors. Just parallel Bellman-Ford!
  - Uh oh, exponential example can still be exponential... (there are good fixes)

Randomized Algorithms

- 6.856: Randomized Algorithms
- Algorithm makes random choices during execution, usually *independent* of input data
- Often using random samples, random walks, random permutation of data ...
- Often faster or simpler than deterministic counterparts (if there even is a deterministic counterpart!), e.g., hashing, quicksort
- New meaning of "answer"
  - Las Vegas algorithm: correct answer, expected runtime (may take longer if unlucky)
  - Monte Carlo algorithm: worst-case runtime, correct most of the time
- Example: Primality testing!

- Miller-Rabin: If n is *not* prime, there's an easily-testable property that at least $3/4$ of the numbers $\{1, \ldots, n-1\}$ have that *prove* n is not prime
- Like finding a needle in a stack of needles
- Run 100 times, to be safe. Chance of error is $1/2^{200} < 10^{-60}$. About the same chance that a thoroughly shuffled deck of cards ends up fully sorted. Much less than the probability of hardware error.[citation needed] I can live with those odds.
- Important: probability is based on random choices during the algorithm, *independent* of the input
- Takes $O(n^3)$ time for an n-bit number
- Best known deterministic algorithm takes $O(n^6)$ (Agrawal, Kayal, Saxena)

Cryptography

- 6.857: Computer and Network Security
- Computing or communicating in the presence of adversaries—eavesdroppers, malicious participants, etc.
- Practical example: SSL Certificates. By releasing some information ("public key") and keeping some hidden ("private key"), sites can identify themselves in a way that, *ideally*,
  - can only be done by someone who knows the private key, and
  - does not reveal enough information for anyone else to reconstruct the private key.
- Your browser does this check for every https site you visit
- E.g., RSA: private key is two primes p, q; public key is $n = p \cdot q$. (many details omitted)
- Mathematically, n determines p and q
- Theoretically, factoring is in NP! Lucky alg could *guess* p and q
- Realistically, we hope there aren't fast enough algorithms / hardware to factor n, if numbers are big enough, e.g., $n \sim 2^{2048}$
- Relies on an *unproven assumption*: factoring is slow

More algorithms classes!

- 6.047: Computational Biology (analyzing and synthesizing genomic data, . . .)
- 6.849: Geometric Folding Algorithms (origami, robot arms, . . .)
- 6.850: Geometric Computing (lines, polygons, meshes)
- 6.853: Algorithmic Game Theory (Nash equilibria, auction mechanism design, . . .)

More theory classes!

- 6.045: Automata, Computability, and Complexity
- 6.440: Essential Coding Theory
- 6.441: Information Theory
- 6.840: Theory of Computing
- 6.841: Advanced Complexity Theory
- 6.842: Randomness and Computation
- 6.845: Quantum Complexity Theory
- 6.890: Algorithmic Lower Bounds: Fun with Hardness Proofs