Lecture 19b

Friday, November 9, 2018 10:48 AM

SIAM Top 10 algorithms in 20-th century

1962: Tony Hoare of Elliott Brothers, Ltd., London, presents **Quicksort**.

Putting N things in numerical or alphabetical order is mind-numbingly mundane. The intellectual challenge lies in devising ways of doing so quickly. Hoare's algorithm uses the age-old recursive strategy of divide and conquer to solve the problem: Pick one element as a "pivot," separate the rest into piles of "big" and "small" elements (as compared with the pivot), and then repeat this procedure on each pile. Although it's possible to get stuck doing all $N(N-1)/2$ comparisons (especially if you use as your pivot the first item on a list that's already sorted!), Quicksort runs on average with $O(N \log N)$ efficiency. Its elegant simplicity has made Quicksort the pos-terchild of computational complexity.

James Cooley

1965: James Cooley of the IBM T.J. Watson Research Center and John Tukey of Princeton University and AT&T Bell Laboratories unveil the **fast Fourier transform**.

Easily the most far-reaching algo-rithm in applied mathematics, the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of asteroids), but it was the Cooley–Tukey paper that made it clear how easily Fourier transforms can be computed. Like Quicksort, the FFT relies on a divide-and-conquer strategy to reduce an ostensibly $O(N^2)$ chore to an $O(N \log N)$ frolic. But unlike Quick- sort, the implementation is (at first sight) nonintuitive and less than straightforward. This in itself gave computer science an impetus to investigate the inherent complexity of computational problems and algorithms.

John Tukey

1977: Helaman Ferguson and Rodney Forcade of Brigham Young University advance an **integer relation detection algorithm**.

The problem is an old one: Given a bunch of real numbers, say $x_1, x_2, \ldots, x_n$, are there integers $a_1, a_2, \ldots, a_n$ (not all 0) for which $a_1x_1 + a_2x_2 + \ldots + a_nx_n = 0$? For $n = 2$, the venerable Euclidean algorithm does the job, computing terms in the continued-fraction expansion of x_1/x_2. If x_1/x_2 is rational, the expansion terminates and, with proper unraveling, gives the "smallest" integers a_1 and a_2. If the Euclidean algorithm doesn't terminate—or if you simply get tired of computing it—then the unraveling procedure at least provides lower bounds on the size of the smallest integer relation. Ferguson and Forcade's generalization, although much more difficult to implement (and to understand), is also more powerful. Their detection algorithm, for example, has been used to find the precise coefficients of the polynomials satisfied by the third and fourth bifurcation points, $B_3 = 3.544090$ and $B_4 = 3.564407$, of the logistic map. (The latter polynomial is of degree 120; its largest coefficient is 257^{30}.) It has also proved useful in simplifying calculations with Feynman diagrams in quantum field theory.

1987: Leslie Greengard and Vladimir Rokhlin of Yale University invent the **fast multipole algorithm**.

This algorithm overcomes one of the biggest headaches of N-body simulations: the fact that accurate calculations of the motions of N particles interacting via gravitational or electrostatic forces (think stars in a galaxy, or atoms in a protein) would seem to require $O(N^2)$ computations—one for each pair of particles. The fast multipole algorithm gets by with $O(N)$ computations. It does so by using multipole expansions (net charge or mass, dipole moment, quadrupole, and so forth) to approximate the effects of a distant group of particles on a local group. A hierarchical decomposition of space is used to define ever-larger groups as distances increase. One of the distinct advantages of the fast multipole algorithm is that it comes equipped with rigorous error estimates, a feature that many methods lack.

divide and conquer.

Consider the n-body problem of gravity

x_1 $\quad$ x_2

x $\qquad$ $\text{force} = \sum_{i=1}^{n} m_i \frac{x_i - x}{\|x_i - x\|^3}$

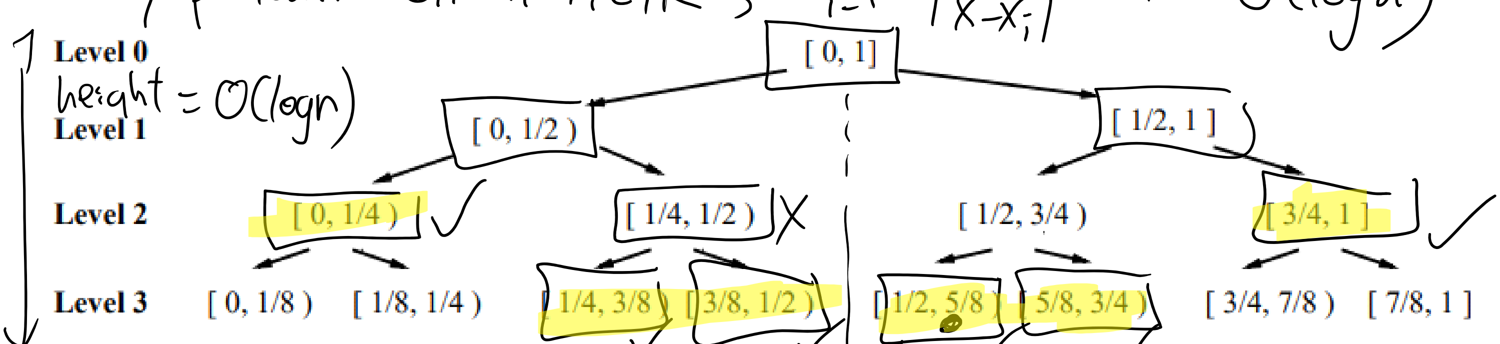
$x_3 \in \mathbb{R}^3$

Naively: $O(n)$ time

Question: With processing, $O(\log n)$ time.

$O(n \log^{O(1)} n)$

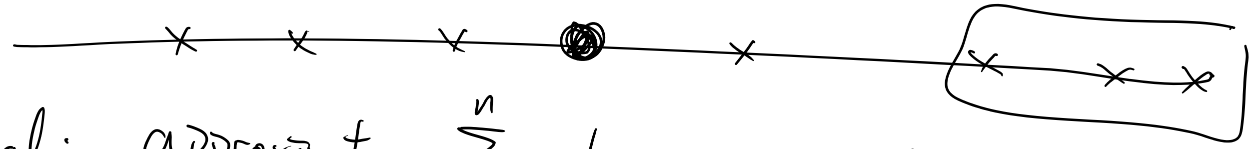
Toy problem: Given $x_i \in \mathbb{R}$, $\sum_{i=1}^{n} \frac{1}{|x - x_i|}$ in $O(\log n)$



- each leaf of tree contains at most 1 ...

Level 3 [0, 1/8) [1/8, 1/4) [1/4, 3/8) [3/8, 1/2) [1/2, 5/8) [5/8, 3/4) [3/4, 7/8) [7/8, 1]

- each leaf of tree contains at most 4 star.

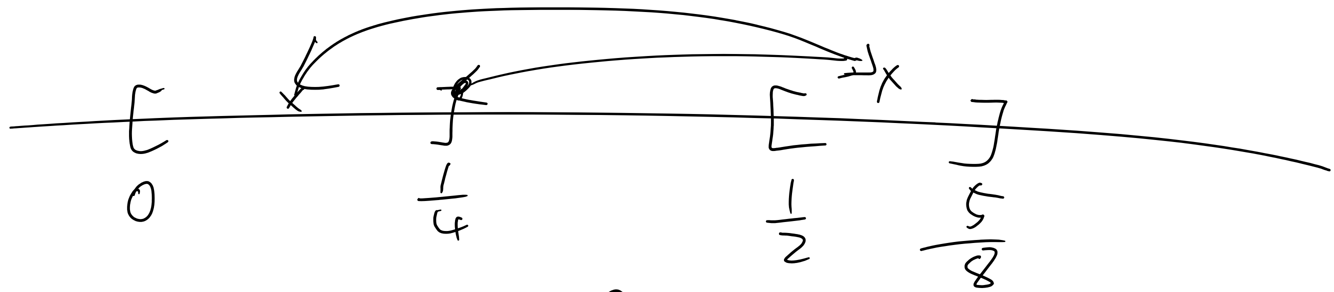


Goal: approximate $\sum_{i=1}^{n} \frac{1}{|x-x_i|}$ $O(\log n)$

- each node record

$$C_{[\alpha,\beta)} = |\{ x_i \text{ st } x_i \in [\alpha,\beta) \}|$$

- define $f_{[\alpha,\beta)}(x) = \sum_{x_i \in [\alpha,\beta)} \frac{1}{|x-x_i|}$



$$f_{[0,\frac{1}{4}]}(x) \simeq \frac{C_{[0,\frac{1}{4}]}}{|x-\frac{1}{4}|}$$

We call $[\alpha,\beta)$ is far from x

if $\forall t \in [\alpha,\beta)$, $\frac{1}{4}|t-x| \le |\beta - x| \le 4|t-x|$

$$\frac{1}{4|t-x|} \le \frac{1}{|\beta-x|} \le \frac{4}{|t-x|}$$

ALG eval$(T_{[\alpha,\beta)}, x)$

ALG eval($T_{[\alpha,\beta)}$, x)

{

If x is ~~far~~ far from $[\alpha, \beta)$

return $\frac{C_{[\alpha,\beta)}}{|\beta - x|}$

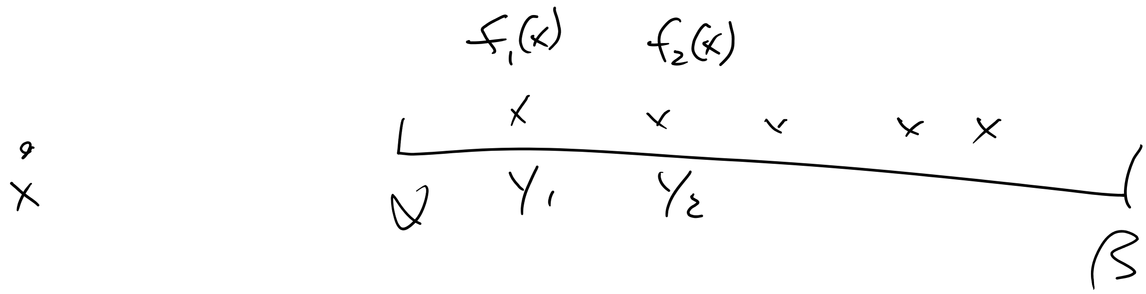
else T is leaf

return $f_{[\alpha,\beta)}(x)$

else

return eval(T→child$_0$, x) + eval(T→child$_1$, X)

}



$$f_1(x) = \underbrace{O + Ox + Ox^2 + \dots}_{} \quad \log(\tfrac{1}{\varepsilon}) \text{ deg}$$