SIAM Top 10 algorithms in 20-th century


Pivoting offers a natural and alphabetical order in mind-shattering triangle. The intuitive 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 the idea of all \( n \log n \) comparisons (especially if one uses as pivot the first or last item on list that's already sorted), Quick sort runs on average with \( O(n \log n) \) efficiency. This insight enabled Hoare's algorithm to be the most successful of all.

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 algorithm in applied mathematics, the FFT revolutionized signal processing. The underlying idea goes back to Gauss (who needed to calculate orbits of comets), but it was the Cooley-Tukey paper that made it clear how easily FFT transforms can be computed. Like Quick sort, the FFT relies on a divide-and-conquer strategy to reduce an otherwise \( O(N^2) \) (where \( N \) is the length of the input) brute-force DFT sort, the implementation is not very straightforward, and there's no littering around. The insight that gave computer science an impetus to investigate the inherent complexity of computational problems and algorithms.

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

The problem is an old one: Given a set of numbers \( x_1, x_2, \ldots, x_n \), find the smallest integers \( a_i \), \( a_2, \ldots, a_n \) (not all 0) for which

\[
a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = 0
\]

is minimal, the expansion terminates, and, with proper scaling, gives the "smallest" integers \( a_i \) and \( a_n \). If the Euclidean algorithm doesn't terminate—or if you simply get tired of computing it—then the preceding procedure at least provides lower bounds on the size of the smallest integer relation. Ferguson and Forcade's algorithm, although much more efficient in implementation and execution, is also more powerful. Their detection algorithm, in fact, has been used to find the sparsest sets of coefficients that satisfy the third and fourth lattice points, \( B_3 = 3.544001 \) and \( B_4 = 3.564476 \), of the binary code (the latter polynomial is of degree 130). Its greatest integer relation is \( 237^2 \). 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 electromagnetic forces (by stars in a galaxy, or atoms in a crystal) would seem to require \( O(N^2) \) computations—since for each particle of particles. The fast multipole algorithm lets you do this by using multiple expansions of charge and dipole moments, quadrupole, and so forth, to approximate the effects of distant groups of particles as a local group. A hierarchical decomposition of space is used to define these 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.

Consider the \( n \)-body problem of gravity.

Naively: \( O(n^2) \) time.

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

Toy problem: Given \( x_i \in \mathbb{R} \), \( x_i \geq 1 \), \( \frac{1}{1 - x_i} \in O(\log n) \)

### Level 0
- Height = \( \mathcal{O}(\log n) \)

### Level 1
- \( [0, 1] \)
- \( [0, 1/2] \)
- \( [1/2, 1] \)

### Level 2
- \( [0, 1/4] \)
- \( [1/4, 1/2] \)
- \( [1/2, 3/4] \)
- \( [3/4, 1] \)

### 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 level of tree contains a root
* each leaf of tree contains at most 7 stars.

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

* each node record

\[ C_{[\alpha, \beta)} = \left| \{ x_i \text{ st } x_i \in [\alpha, \beta) \} \right| \]

* define \( \frac{C_{[0, \frac{1}{4}]}(x)}{X-\frac{1}{4}} \)

We call \( [\alpha, \beta) \) is for \( \text{from } x \)

\( \frac{1}{4} |t-x| \leq |\beta-x| \leq 4 |t-x| \)

\( \frac{1}{4 |t-x|} \leq \frac{1}{|\beta-x|} \leq 4 \frac{1}{|t-x|} \)

\[ \text{ALG eva}(T_{[\alpha, \beta)}, x) \]
\text{ALG} \quad \text{eval}(L_{\beta_0}, x)
\{
\text{If x is far from } L_{\beta_0} \text{,}
\text{return } \frac{C_{\beta_0}}{|\beta - x|}
\text{else T is leaf}
\text{return } f_{\beta_0}(x)
\text{else}
\text{return } \text{eval}(T \rightarrow \text{child}_0, x) + \text{eval}(T \rightarrow \text{child}_1, x)
\}

\begin{align*}
\hat{f}_1(x) &= f_1(x) + f_2(x) \\
&= \frac{x_{\nu} \cdot x_{\gamma}}{x_{\nu} \cdot x_{\gamma}}
\end{align*}

\hat{f}_1(x) = O + O x + O x^2 + \ldots \quad (\log(\frac{1}{\epsilon})) \text{ deg}

\text{\underline{K}}