CS 312: Algorithms

More Divide and Conquer

Dan Sheldon

Mount Holyoke College

Last Compiled: October 29, 2018

Master Theorem

Consider the general recurrence:

$$T(n) \le aT\left(\frac{n}{b}\right) + cn^d$$

This solves to:

$$T(n) = \begin{cases} \Theta(n^d) & \text{if } \log_b a < d \\ \Theta(n^d \log n) & \text{if } \log_b a = d \\ \Theta(n^{\log_b a}) & \text{if } \log_b a > d \end{cases}$$

Intuition: work at each level of the recursion tree is (1) decreasing exponentially, (2) staying the same, (3) increasing exponentially.

Integer Multiplication

Motivation: multiply two 30-digit integers?

```
  153819617987625488624070712657
x 925421863832406144537293648227
  ------------------------------
```

- Multiply two 300-digit integers?
- Cannot do this in Java with built-in data types
- 64-bit unsigned integer can only represent integers up to ~20 digits ($2^{64} \approx 10^{20}$)

Warm-Up: Addition

Input: two n-digit binary integers x and y
Goal: compute $x + y$

Let's do everything in base-10 instead of binary to make examples more familiar.

Grade-school algorithm:

```
    1854
  + 3242
 -------
    5096
```

Running time? $\Theta(n)$

Integer Multiplication Problem

Input: two n-digit base-10 integers x and y
Goal: compute xy

Can anyone think of an algorithm?

Grade-School Algorithm (Long Multiplication)

Example: $n = 3$

```
    287
  x 132
 ------
    574
   861
  287
--------
  37884
```

$$287 \times 132 = (2 \times 287) + 10 \cdot (3 \times 287) + 100 \cdot (1 \times 287)$$

Running time? $\Theta(n^2)$
But xy has at most $2n$ digits. Can we do better?

Divide and Conquer: First Try

Idea: split x and y in half (assume n is a power of 2)

$$x = \underbrace{3380}_{x_1}\underbrace{2367}_{x_0}$$
$$y = \underbrace{4508}_{y_1}\underbrace{1854}_{y_0}$$

Then use distributive law

$$\begin{aligned} xy &= (10^{n/2}x_1 + x_0) \times (10^{n/2}y_1 + y_0) \\ &= 10^n x_1y_1 + 10^{n/2}(x_1y_0 + x_0y_1) + x_0y_0 \end{aligned}$$

Have reduced the problem to multiplications of $n/2$-digit integers and additions of n-digit numbers

Divide and Conquer: First Try

Recursive algorithm:

$$xy = 10^n x_1y_1 + 10^{n/2}(x_1y_0 + x_0y_1) + x_0y_0$$

Running time? Four multiplications of $n/2$ digit numbers plus three additions of at most n-digit numbers

$$\begin{aligned} T(n) &\leq 4T\left(\frac{n}{2}\right) + cn \\ &= O(n^{\log_2 4}) \\ &= O(n^2) \end{aligned}$$

We did not beat the grade-school algorithm. :(

Better Divide and Conquer

Same starting point:

$$xy = 10^n x_1y_1 + 10^{n/2}(x_1y_0 + x_0y_1) + x_0y_0$$

Trick: use three multiplications to compute the following:

$$\begin{aligned} A &= (x_1 + x_0)(y_1 + y_0) = x_1y_1 + x_1y_0 + x_0y_1 + x_0y_0 \\ B &= x_1y_1 \\ C &= x_0y_0 \end{aligned}$$

Then

$$xy = 10^n B + 10^{n/2}(A - B - C) + C$$

Total: three multiplications of $n/2$-digit integers, six additions

Better Divide and Conquer

Total: three multiplications of $n/2$-digit integers, six additions of at most n-digit integers

$$\begin{aligned} T(n) &\leq 3T\left(\frac{n}{2}\right) + cn \\ &= O(n^{\log_2 3}) \\ &\approx O(n^{1.59}) \end{aligned}$$

We beat long multiplication!

Idea can be generalized to be even faster (split x and y into k parts instead of two)

Fastest known integer multiplication algorithm is $O(n \log n)$ — also by divide-and-conquer (Fast Fourier transform),

Closest Pair of Points

Another beautiful divide and conquer algorithm

Closest Pair of Points

- **Problem 1**: Given n points on a line $p_1, p_2, \ldots, p_n \in \mathbb{R}$, find the closest pair: $\min_{i \neq j} |p_i - p_j|$.
  - Compare all pairs $O(n^2)$
  - Better algorithm? Sort the points and compare adjacent pairs. $O(n \log n)$
- **Problem 2:** Now what if the points are in $\mathbb{R}^2$?
  - Compare all pairs $O(n^2)$
  - Sort? Points can be close in x-coordinate and far in y, and vice-versa
  - We'll do it in $O(n \log n)$ steps using divide-and-conquer.

Problem Formulation

- **Input**: set of points $P = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, y_i)$
- **Assumption**: we can iterate over points in order of x- or y-coordinate in $O(n)$ time. Pre-generate data structures to support this in $O(n \log n)$ time.

Recursive Algorithm

1. Find vertical line L to divide points into sets P_L and P_R of size $n/2$. $O(n)$
2. Recursively find minimum distance in P_L and P_R.
   - δ_L = minimum distance between $p, q \in P_L, p \neq q$. $T(n/2)$
   - δ_R = same for P_R. $T(n/2)$
3. δ_M = minimum distance between $p \in P_L, q \in P_R$. ??
4. Return $\min(\delta_L, \delta_R, \delta_M)$.

Naive Step 3 takes $\Omega(n^2)$ time. But if we do it in $O(n)$ time we get

$$T(n) \leq 2T(n/2) + O(n) \Longrightarrow T(n) = O(n \log n)$$

Making Step 3 Efficient

- **Goal**: given δ_L, δ_R, compute $\min(\delta_L, \delta_R, \delta_M)$
- Observation: Let $\delta = \min(\delta_L, \delta_R)$. If $p \in P_L, q \in P_R$ differ by at least δ in either the x- or y-coordinate, they cannot be the overall closest pair, so we can ignore the pair (p, q).
- Let S be the set of points within distance δ from L. We only need to consider pairs that are both in S.
- For a given point $p \in S$, how many points q are within δ units of p in the y coordinate?
  - **Claim**: at most 12
  - Algorithm: iterate through points $p \in S$ in order of y coordinate and compare p to 12 adjacent points in this order. $O(n)$.
- Intuition: the set S is "nearly one-dimensional". Points cannot be packed in tightly, because two points on the same side of L are at least distance δ apart. Proof sketch on board.