COMPSCI 311: Introduction to Algorithms

Lecture 10: Divide and Conquer

Marius Minea

University of Massachusetts Amherst

slides credit: Dan Sheldon

Review: Solving Recurrences

Useful general recurrence and its solutions:

$$T(n) \le q \cdot T(n/2) + cn$$

1. $q = 1$: $T(n) = O(n)$ — more work at top level of tree
2. $q = 2$: $T(n) = O(n \log n)$ — equal contributions
3. $q > 2$: $T(n) = O(n^{\log_2 q})$ — more work towards base

Clicker Question 1

Which of the following is *not* true ?

A) $n \log n = O(n^2)$

B) $n \log n = O(n^{1.1})$

C) There exists a large enough k with $n \log n = \Theta(n^k)$

D) $n \log n = \Omega(n \log \log n)$

More general: Master Theorem

Let $T(n) = aT(n/b) + f(n)$, with $a \ge 1$, $b > 1$. Then:

1. $T(n) = \Theta(n^{\log_b a})$ when $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$
   $f(n)$ grows polynomially slower than $n^{\log_b a}$ pause
2. $T(n) = \Theta(n^{\log_b a} \log n)$ when $f(n) = \Theta(n^{\log_b a})$ (border case)
   $T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$ when $f(n) = \Theta(n^{\log_b a} \log^k n)$
3. $T(n) = \Theta(f(n))$ when $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ *and*
   $af(n/b) < cf(n)$ for some $c < 1$ when n sufficiently large
   $f(n)$ grows polynomially faster than $n^{\log_b a}$

Does not cover everything: gaps between 1 and 2, and 2 and 3

Guess and prove by induction for other cases

Clicker Question 2

Recall the Master theorem for $T(n) = aT(n/b) + f(n)$:
1. $T(n) = \Theta(n^{\log_b a})$ when $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$
2. $T(n) = \Theta(n^{\log_b a} \log n)$ when $f(n) = \Theta(n^{\log_b a})$
3. $T(n) = \Theta(f(n))$ when $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$ *and* $af(n/b) < cf(n)$ for some $c < 1$ when n sufficiently large

If $T(n) = 9T(n/3) + f(n)$ solves to $T(n) = \Theta(n^2)$, what can $f(n)$ be? Choose the best answer.

A) $f(n) = O(n)$

B) $f(n) = O(n \log n)$

C) $f(n) = O(n \log^2 n)$

D) $f(n) = O(n^2)$

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 ($2^{64} \approx 10^{20}$)

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

Algorithm?

Warm-Up: Addition

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

We'll do it in base-10 instead of binary (perhaps more familiar).

Grade-school algorithm:

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

Running time? $\Theta(n)$

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: An Example

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: Analysis

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

$$T(n) \le 4T\left(\frac{n}{2}\right) + cn$$

Does this fit in our general formulas?

$$\begin{aligned} &= O(n^{\log_2 4}) \\ &= O(n^2) \end{aligned}$$

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

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

We beat long multiplication!

Can be done even faster (split x and y into k parts instead of two)

Finding Minimum Distance between 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 $O(n^2)$
  - Sort the points and compare adjacent pairs $O(n \log n)$
  - Can you directly do divide-and-conquer? Need median
- **Problem 2:** Now what if the points are in $\mathbb{R}^2$?
  - Compare all pairs $O(n^2)$
  - Sort? Points can be close in one coordinate and far in the other
  - We'll do it in $O(n \log n)$ steps using divide-and-conquer.
- **Input**: set of points $P = \{p_1, \ldots, p_n\}$ where $p_i = (x_i, y_i)$

Minimum Distance: Recursive Algorithm

- **Assumption**: we can iterate over points in order of x- or y-coordinate in $O(n)$ time.
  Pre-sort in $O(n \log n)$ time along each axis (two arrays).

1. Find vertical line L to split points into sets P_L, 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

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

Making Step 3 Efficient

- **Goal**: given δ_L, δ_R, compute $\min(\delta_L, \delta_R, \delta_M)$
- Let $\delta = \min(\delta_L, \delta_R)$. If $p \in P_L, q \in P_R$ are at least δ apart, they cannot be a closer pair, so we can ignore pair (p, q).
- Let S be the set of points within distance δ from L (vertical strip centered on line 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?

How to find closest pair with one point in each side?

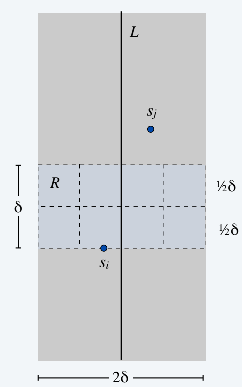
Def. Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate.

Claim. If $|j - i| > 7$, then the distance between s_i and s_j is at least δ.

Pf.
- Consider the 2δ-by-δ rectangle R in strip whose min y-coordinate is y-coordinate of s_i.
- Distance between s_i and any point s_j above R is ≥ δ.
- Subdivide R into 8 squares.
- At most 1 point per square.
- At most 7 other points can be in R. ▪

diameter is $\delta / \sqrt{2} < \delta$

constant can be improved with more refined geometric packing argument



73

slide credit: Kevin Wayne / Pearson

Clicker Question 3

Based on the split into squares in the figure, it suffices to compare each point in the vertical strip to

A) 7 points

B) 14 points

C) 4 points

Concluding the Merge Step

- Compute sorted lists S_L and S_R of close points left and right of the line L select in $O(n)$
- Advance in both lists by increasing y coordinate (merge-like) $O(n)$ iterations
- Compare to at most 4 following points in *other* list $O(1)$ work in loop
- Minimum distance across halves in $O(n)$
- Overall recursion gives $O(n \log n)$