EE360C: Algorithms

Divide and Conquer

Summer 2019

Department of Electrical and Computer Engineering
University of Texas at Austin

Multiplication

Consider the traditional iterative "ripple carry" algorithm for addition. What is its running time for adding two n-digit numbers?
Assuming one-digit addition is a constant time operation, $O(n)$.

What is our common algorithm for multiplying two n-digit numbers?
We use n one-digit multiplications and n n-digit additions. What is the running time of this algorithm?

It's fairly straightforward to write as a pair of nested for loops, each done $O(n)$ times, so it's $O(n^2)$ overall.

Can we do better?

Recursive Multiplication I

We start recursive multiplication by observing:

$$(10^m a + b)(10^m c + d) = 10^{2m} ac + 10^m (bc + ad) + bd$$

MULTIPLY(x, y, n)

```
if n = 1
return x × y
   else
m ← ⌈n/2⌉
a ← ⌊x/10^m⌋; b ← x mod 10^m
d ← ⌊y/10^m⌋; c ← y mod 10^m
e ← MULTIPLY(a, c, m)
f ← MULTIPLY(b, d, m)
g ← MULTIPLY(b, c, m)
h ← MULTIPLY(a, d, m)
return 10^(2m) e + 10^m (g + h) + f
```

Can you write the recurrence? $T(n) = 4T(\lceil n/2 \rceil) + O(n)$

Can you solve the recurrence? Use the master method (revisited) where $a = 4$, $b = 2$, $l = 1$, and $k = 0$. $\log_b a = \log_2 4 = 2$. So $l < \log_b a$, so use Case 1. Then $T(n) = \Theta(n^{\log_b a}) = \Theta(n^2)$

Recursive Multiplication II

We can replace two multiplications with two we're already doing anyway and one additional one.

$$ac + bd - (a - b)(c - d) = bc + ad$$

$\textsc{FastMultiply}(x, y, n)$

if $n = 1$
return $x \times y$
    else
$m \leftarrow \lceil n/2 \rceil$
$a \leftarrow \lfloor x/10^m \rfloor$; $b \leftarrow x \bmod 10^m$
$d \leftarrow \lfloor y/10^m \rfloor$; $c \leftarrow y \bmod 10^m$
$e \leftarrow \textsc{Multiply}(a, c, m)$
$f \leftarrow \textsc{Multiply}(b, d, m)$
$g \leftarrow \textsc{Multiply}(a - b, c - d, m)$
return $10^{2m}e + 10^m(e + f - g) + f$

What's the recurrence for this one? $T(n) = 3T(\lceil n/2 \rceil) + O(n)$

Which solves to? $\Theta(n^{\lg 3}) = \Theta(n^{1.585})$.

Matrix Multiplication

Matrix Multiplication

Matrix Multiplication

Given two n-by-n matrices, A and B, compute $C = AB$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

Brute Force

$\Theta(n^3)$ arithmetic operations

Fundamental Question

Can we improve upon brute force?

Matrix Multiplication Warmup

Divide and Conquer

- Divide: partition A and B into $\frac{1}{2}n$ by $\frac{1}{2}n$ blocks
- Conquer: multiply 8 $\frac{1}{2}n$ by $\frac{1}{2}n$ recursively
- Combine: add appropriate products using 4 matrix additions

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= (A_{11} \times B_{11}) + (A_{12} \times B_{21}) \\ C_{12} &= (A_{11} \times B_{12}) + (A_{12} \times B_{22}) \\ C_{21} &= (A_{21} \times B_{11}) + (A_{22} \times B_{21}) \\ C_{22} &= (A_{21} \times B_{12}) + (A_{22} \times B_{22}) \end{aligned}$$

$$\mathrm{T}(n) = \underbrace{8T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, form submatrices}} \Rightarrow \mathrm{T}(n) = \Theta(n^3)$$

Matrix Multiplication: Key Idea

Key Idea

Multiply 2-by-2 block matrices with only 7 multiplications (7 multiplications and 18 additions/subtractions)

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= P_5 + P_4 - P_2 + P_6 \\ C_{12} &= P_1 + P_2 \\ C_{21} &= P_3 + P_4 \\ C_{22} &= P_5 + P_1 - P_3 - P_7 \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \times (B_{12} - B_{22}) \\ P_2 &= (A_{11} + A_{12}) \times B_{22} \\ P_3 &= (A_{21} + A_{22}) \times B_{11} \\ P_4 &= A_{22} \times (B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{22}) \times (B_{11} + B_{22}) \\ P_6 &= (A_{12} - A_{22}) \times (B_{21} + B_{22}) \\ P_7 &= (A_{11} - A_{21}) \times (B_{11} + B_{12}) \end{aligned}$$

Fast Matrix Multiplication

Fast matrix multiplication (Strassen 1969)

- Divide: partition A and B into $\frac{1}{2}n$ by $\frac{1}{2}n$ blocks
- Compute: 14 $\frac{1}{2}n$ by $\frac{1}{2}n$ matrices via 10 matrix additions
- Conquer: multiply 7 $\frac{1}{2}n$ by $\frac{1}{2}n$ matrices recursively
- Combine: 7 products into 4 terms using 8 matrix additions

Analysis

- Assume n is a power of 2
- $T(n)$ is the number of arithmetic operations

$$\mathrm{T}(n) = \underbrace{7T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n^2)}_{\text{add, subtract}} \Rightarrow \mathrm{T}(n) = \Theta(n^{\log_2 7}) = O(n^{2.81})$$

Recursion Trees

Another Master Method Example

Can $T(n) = 2T(n/2) + \Theta(n/\lg n)$ be solved using the master method?

- $a = 2$, $b = 2$, $l = 1$, and $k = -1$. Whoops!

As an exercise, use the original master method formulation and show that this formula doesn't fit there either. (It doesn't.)

So how do we solve generic recurrences?

Recursion Trees

In a **recursion tree**, each node represents the cost of a single subproblem somewhere in the set of recursive function invocations.

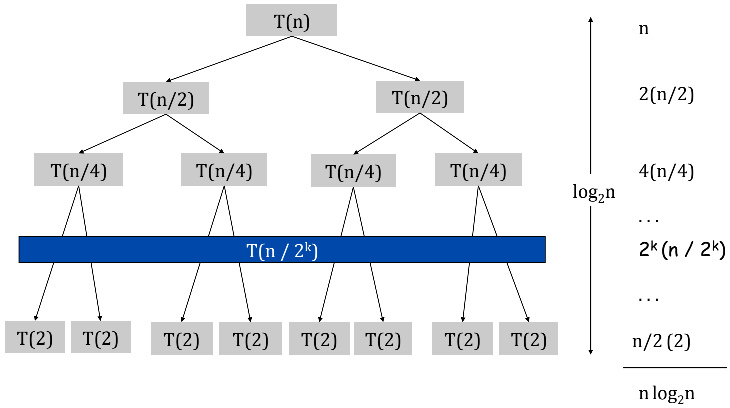
- Sum the nodes in each level to get a per-level cost
- Sum all of the levels to get a total cost

Recursion trees are particularly useful for divide and conquer problems.

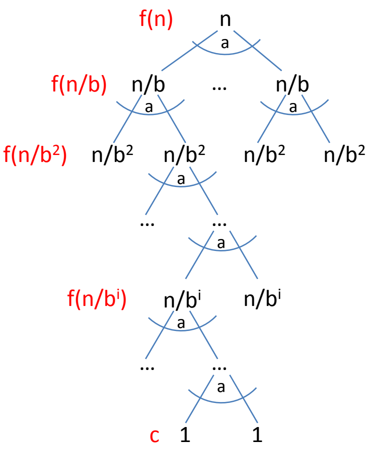
Recursion Trees (cont.)

A simple example:

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ \underbrace{2T(n/2)}_{\text{sorting both halves}} + \underbrace{n}_{\text{merging}} & \text{otherwise} \end{cases}$$



Recursion Trees (cont.)



level	num. of nodes	time per node
0	1	$f(n)$
1	a	$f(n/b)$
2	a^2	$f(n/b^2)$
i	a^i	$f(n/b^i)$
$\log_b n$	$a^{\log_b n}$	c

Recursion Tree Summation

Before we get to the full summation, we need the following fact:

$$a^{\log_b n} = n^{\log_b a}$$

You can verify this by taking the $\log_b$ of both sides.

So we can get the running time of the entire recurrence by summing up all of the levels:

$$T(n) = \left[\sum_{i=0}^{(\log_b n)-1} a^i \times f(n/b^i) \right] + n^{\log_b a} \times c$$

Recursion Tree Summation (cont.)

$$T(n) = \left[\sum_{i=0}^{(\log_b n)-1} a^i \times f(n/b^i) \right] + n^{\log_b a} \times c$$

The term $f(n/b^i)$ represents the running time of a single subproblem at level i of the recursion tree. This is the second term of our general recurrence statement ($T(n) = aT(n/b) + f(n)$). From the master method, we know that it is useful to write $f(n)$ in terms of l and k:

$$f(n) = \Theta(n^l \, (\lg n)^k)$$

Substituting n/b^i for n, our sum becomes:

$$T(n) = \left[\sum_{i=0}^{(\log_b n)-1} a^i \times \Theta((n/b^i)^l \, (\lg (n/b^i))^k) \right] + n^{\log_b a} \times c$$

Recursion Tree Exercise

Use this summation to formulate (and solve) the merge sort recurrence:

$$T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$$

Here's that summation again:

$$T(n) = \left[\sum_{i=0}^{(\log_b n)-1} a^i \times \Theta((n/b^i)^l \, (\lg\,(n/b^i))^k) \right] + n^{\log_b a} \times c$$

Recursion Tree Example

Let's use a recursion tree to solve:

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

Substituting into the summation, we have:

$$T(n) = \left[\sum_{i=0}^{(\log_4 n)-1} 3^i \times f(n/4^i) \right] + \Theta(n^{\log_4 3})$$

Restating using the Θ expression from master method:

$$T(n) = \left[\sum_{i=0}^{(\log_4 n)-1} 3^i \times \Theta((n/4^i)^l \, (\lg (n/4^i))^k) \right] + \Theta(n^{\log_4 3})$$

Since $l = 2$ and $k = 0$, this reduces to:

$$T(n) = \left[\sum_{i=0}^{(\log_4 n)-1} 3^i \times \Theta(n^2/16^i) \right] + \Theta(n^{\log_4 3})$$

Recursion Tree Example (cont.)

So how do we know what the running time is, given:

$$T(n) = \left[\sum_{i=0}^{(\log_4 n)-1} 3^i \times \Theta((n/4^i)^2) \right] + \Theta(n^{\log_4 3})$$

We have to solve the sum. First let's rewrite it a little bit:

$$T(n) = cn^2 \left[\sum_{i=0}^{(\log_4 n)-1} (3/16)^i \right] + \Theta(n^{\log_4 3})$$

Then we can apply the geometric series rule (Appendix A, A.5) and get:

$$cn^2 \left[\frac{(3/16)^{\log_4 n} - 1}{3/16 - 1} \right] + \Theta(n^{\log_4 3})$$

Recursion Tree Example (cont.)

Wow. That's not helpful. What does this mean?

$$cn^2 \left[\frac{(3/16)^{\log_4 n} - 1}{3/16 - 1} \right] + \Theta(n^{\log_4 3})$$

But we can take a step back and say that this finite geometric series must be less than the geometric series that is the same other than being infinite. That is:

$$T(n) < cn^2 \left[\sum_{i=0}^{\infty} (3/16)^i \right] + \Theta(n^{\log_4 3})$$

Which is a simpler summation to solve:

$$\begin{aligned} T(n) &< \frac{16}{13} cn^2 + \Theta(n^{\log_4 3}) \\ &= O(n^2) \end{aligned}$$

Verifying Recurrence Solutions

Let's say we wanted to verify that this was the correct answer. How would we do it? Induction!

Our recurrence is: $T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$; our guess is $O(n^2)$.

Our base case is taken care of by the fact that we assume $T(n) = \Theta(1)$ when $n = 1$.

Inductive Step

We need to prove that $T(n) \leq cn^2$ for an some c. We assume that the claim holds for $\lfloor n/4 \rfloor$, i.e., $T(\lfloor n/4 \rfloor) \leq c(\lfloor n/4 \rfloor)^2$. Then

$$\begin{aligned} T(n) &\leq 3(c(\lfloor n/4 \rfloor)^2) + \Theta(n^2) \\ &\leq 3cn^2/16 + dn^2 \\ &\leq cn^2 \end{aligned}$$

which is true for values $c \geq (16/3)d$

Recurrences and Induction Exercise

Solve the merge sort recurrence by induction:

$$T(n) = 2T(\lfloor n/2 \rfloor) + \Theta(n)$$

Summations

Solving recurrences like this requires copious use of summations like the one above. Look these up.

The most common:

Arithmetic Series

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1)$$

Geometric Series

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1}$$

Infinite Geometric Series

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1-x}$$

Closest Pairs

Another Example: Closest Pairs

Closest Pair

Given n points in the plane, find a pair with smallest Euclidean distance between them.

A Fundamental geometric primitive

Used in graphics, computer vision, geographic information systems molecular modeling, air traffic control.

Brute Force

Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

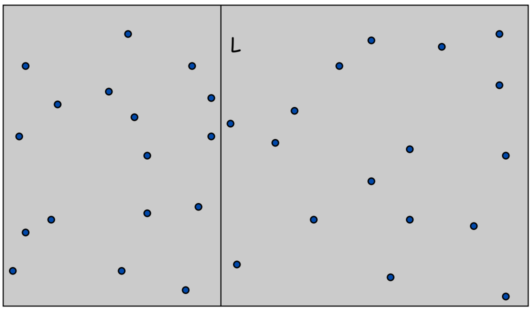
Can we do better?

- To make the presentation clearer, let's assume no two points have the same x coordinate.

Closest Pair of Points

Algorithm: Divide

Draw a vertical line L so that roughly $n/2$ points are on each side of L.



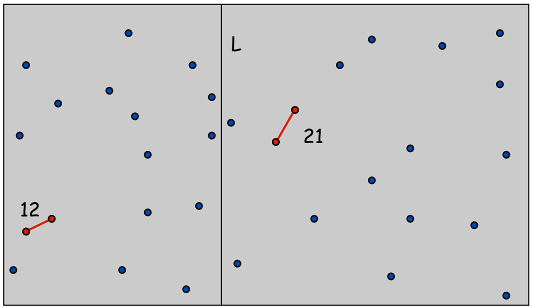
Closest Pair of Points

Algorithm: Divide

Draw a vertical line L so that roughly $n/2$ points are on each side of L.

Algorithm: Conquer

Find the closest pair in each side recursively.



Closest Pair of Points

Divide

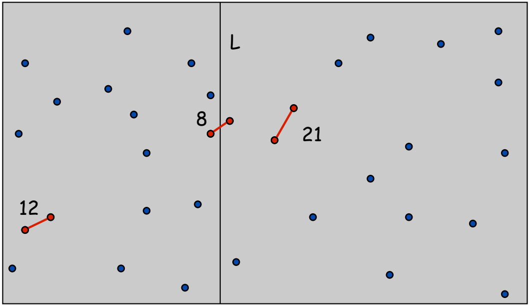
Draw a vertical line L so that roughly $n/2$ points are on each side of L.

Conquer

Find the closest pair in each side recursively.

Combine

Find the closest pair with one point in each side. Return the best of the three solutions.



Closest Pair: Combine Step

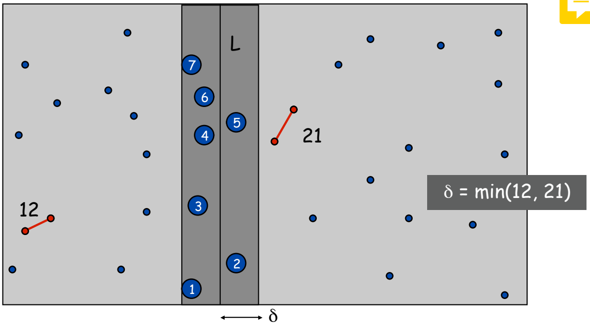
Find the closest pair with one point on each side, assuming that *distance* $< \delta$ (where δ is the minimum of the closest pair

Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < δ.

- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!

ne



31

Closest Pair: Combine Step (II)

Definition

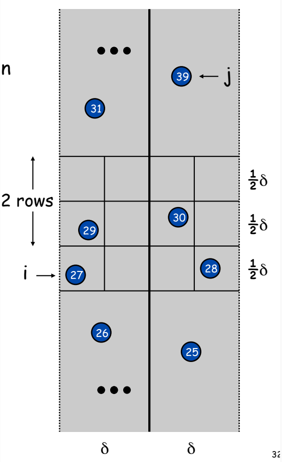
Let s_i be the point in the 2δ-strip with the i^{th} smallest y coordinate.

Claim

If $|i - j| \geq 12$ then the distance between s_i and s_j is at least δ.

- No two points lie in the same $\frac{1}{2}\delta$ by $\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$.

This is also true if you replace 12 with 7.



Closest Pair Algorithm

$\text{ClosestPair}(p_1, p_2, \ldots, p_n)$

Compute L s.t. half the points are on each side of L
$\delta_1 = \text{ClosestPair}(p_1, \ldots, p_L)$
$\delta_2 = \text{ClosestPair}(p_{L+1}, \ldots, p_n)$
$\delta = \min \delta_1, \delta_2$
Delete all points further than δ from L
Sort remaining points by y-coordinate
Scan points in y order and compare distance
between each point and next 11 neighbors.
If any of these distances is less than δ, update δ.
return δ

Closest Pair Analysis

- Time to sort original points: $O(n \log n)$
- Divide: $O(n)$
- Conquer: 2 subproblems of size $n/2$
- Combine: $O(n \log n)$

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

Which solves, by the Master Method, to $O(n \log^2 n)$

Which we can reduce to $O(n \log n)$ by pre-sorting the y coordinates before we start.

Questions?