Announcements

- Proj. #1 due Today
- Monday: MLK Holiday
- Screencast Lecture #7 & Quiz #2
  - Due by end of day Tuesday
- HW #4 due Wednesday at beginning of class
- We’ll work on HW #5 Wednesday in class together
  - The other half of the Lecture #7 "flip"

Objectives

- Introduce Divide and Conquer
- Apply to Multiplication
- Analyze by defining a recurrence relation
- Introduce the Convex Hull problem, time permitting

Thought

"Nothing is particularly hard if you divide it into small jobs."

-- Henry Ford

Divide and Conquer

Solve the smaller pieces.

Been there

- Which divide and conquer algorithms have we already encountered?
Recall Efficiency of Multiplication Algorithms

- Problem: Multiplication
- Algorithms:
  - American
  - British
  - a la Francaise / Russe
  - Arabic
  - ...
- Efficiency so far: $O(n^2)$

\[
\begin{array}{c}
05 \times 02 \\
5001
\end{array}
\]

Divide and Conquer

Is it correct?

\[
\begin{array}{c}
05 \times 50 \text{ shift } 4 = 2500000 \\
05 \times 01 \text{ shift } 2 = 500 \\
02 \times 50 \text{ shift } 2 = 10000 \\
02 \times 01 \text{ shift } 0 = 2 \\
\end{array}
\]

\[
\begin{array}{c}
5001 \\
10002 \\
0 \\
2500500 \\
2510502 \\
2510502
\end{array}
\]

How long does it take on an input of size $n$? $O(n^2)$

Why isn’t it faster?

- Yes.
- We shall see how ...

Divide and Conquer

\[
\begin{array}{c}
05 \times 50 \text{ shift } 4 = 2500000 \\
05 \times 01 \text{ shift } 2 = 500 \\
02 \times 50 \text{ shift } 2 = 10000 \\
02 \times 01 \text{ shift } 0 = 2 \\
\end{array}
\]

\[
\begin{array}{c}
2500500 \\
2500500 \\
2510502 \\
2510502
\end{array}
\]

Product = $wy_10^4 + (wz + xy)10^2 + xz$

Can we do better?

- Yes.
### Divide and Conquer

Let \( r = (w + x)(y + z) = wy + wz + xy + xz \)

Then \( wz + xy = r - wy - xz \)

\[
\text{product} = wy10^4 + (r - wy - xz)10^2 + xz
\]

From four sub-instances to three!

### Recursive application

- **Result for 2 digits, base 10:**
  \[
  \text{product} = wy \cdot 10^4 + \{(w+x)(y+z) - wy - xz\} \cdot 10^2 + xz
  \]

- **Result for \( n \) digits, base 10:**
  \[
  \text{product} = wy \cdot 10^n + \{(w+x)(y+z) - wy - xz\} \cdot 10^n + xz
  \]

- **Result for \( n \) digits, base 2:**
  \[
  \text{product} = 2^r \cdot \{(w+x)(y+z) - wy - xz\} \cdot 2^r + xz
  \]

### Algorithm

```python
function multiply(x, y):
    Input: Positive integers \( x \) and \( y \), in binary
    Output: Their product
    n = max(size of \( x \), size of \( y \))
    if n = 1: return xy
    \( x_l, x_r = \) leftmost \( \lfloor n/2 \rfloor \), rightmost \( \lfloor n/2 \rfloor \) bits of \( x \)
    \( y_l, y_r = \) leftmost \( \lfloor n/2 \rfloor \), rightmost \( \lfloor n/2 \rfloor \) bits of \( y \)
    \( f = multiply(x_l, y_l) \)
    \( f = multiply(x_l, y_r) \)
    \( f = multiply(x_r, y_l) \)
    \( r = multiply(x_r, y_r) \)
    return \( f \cdot 2^n + \{f - r - \epsilon\} \cdot 2^n + xz \)
```

### Algorithm

```python
function multiply(x, y):
    Input: Positive integers \( x \) and \( y \), in binary
    Output: Their product
    n = max(size of \( x \), size of \( y \))
    if n = 1: return xy
    \( x_l, x_r = \) leftmost \( \lfloor n/2 \rfloor \), rightmost \( \lfloor n/2 \rfloor \) bits of \( x \)
    \( y_l, y_r = \) leftmost \( \lfloor n/2 \rfloor \), rightmost \( \lfloor n/2 \rfloor \) bits of \( y \)
    \( f = multiply(x_l, y_l) \)
    \( f = multiply(x_l, y_r) \)
    \( f = multiply(x_r, y_l) \)
    \( r = multiply(x_r, y_r) \)
    return \( f \cdot 2^n + \{f - r - \epsilon\} \cdot 2^n + xz \)
```
How long does it take?

- Characterize running time using a recurrence relation.
- Let \( t(n) \) be the amount of time to compute an answer on an input of size \( n \).
- Then: \( t(n) = 3t\left(\frac{n}{2}\right) + g(n) \)
  - where \( g(n) \in O(n) \), the cost of dividing and reassembling sub-results
- Answering a question with a question!
- Want: a closed-form answer

Master Theorem: Analysis of Divide & Conquer

Define:
- \( a \) = number of sub-instances that must be solved
- \( n \) = original instance size (variable)
- \( n/b \) = size of sub-instances
- \( d \) = polynomial order of \( g(n) \), where \( g(n) \) = cost of dividing and recombining

Then:
- \( t(n) = a \cdot t(n/b) + g(n) \) for \( g(n) \in O(n^d) \)

We’ll prove this! * Assume that \( n \) is a power of \( b \).

Analysis

- DC multiplication with Karatsuba’s insight:
  - \( a = 3 \): there are 3 instances of multiplication
  - \( b = 2 \): each is 1/2 the size of the original multiplication
  - \( d = 1 \): the divide and the reassemble has linear complexity

- Thus, \( t(n) = 3 \cdot t(n/2) + g(n) \) where: \( g(n) \in O(n^2) \)
- Notice: \( a = 3 > 2^1 = b^d \)
- Recall: \( t(n) = \begin{cases} 
  O(n^a) & \text{if } a < b^d \Rightarrow d > \log_b a \\
  O(n^d \log n) & \text{if } a = b^d \Rightarrow d = \log_b a \\
  O(n^{d \cdot \log_b a}) & \text{if } a > b^d \Rightarrow d < \log_b a 
\end{cases} \)

- Thus, \( t(n) \in O(n^{ab^d}) = O(n^{1.585}) \)

Another Logarithm Identity

\[
\log_m(n) = \frac{\log_p n}{\log_p m}
\]

Consequence:
- \( o(k^{\log_a b}) = o \left( \frac{\log_{k^{\log_a n}}}{\log_{k^{\log_a n}} n} \right) = o \left( \log_{k^{\log_a n}} \right) \approx 0 \left( \log_{k^{\log_a n}} \right) \leq 0 \left( \log_k n \right) \)

Essentials of Divide and Conquer

- Given problem instance of size \( n \)
- Divide into \( a \) sub-instances of size \( n/b \)
- Recursively solve these sub-instances
- Appropriately combine their answers
- Analyze with the Master Theorem
- In project #2, your job is to apply this pattern to the Convex Hull problem.

General: Divide and Conquer

function DC(x) : answer
if length(x) < threshold then return adhoc(x)
decompose x into sub-instances \( x_1, x_2, ..., x_a \) of size \( n/b \)
for i \( \in \) 1 to a do \( y_i \leftarrow DC(x_i) \)
recombine the \( y_i \)’s to get a solution \( y \) for \( x \)
return \( y \)

Where:
- adhoc(x) = is the basic algorithm for small instances
- \( a \) = the number of divisions at each level
- \( n/b \) = the fraction of the whole for a sub-instance
Choosing a Threshold

- Threshold = t means:
  "stop dividing when problem size = t"

- Threshold depends on
  - Algorithm
  - and implementation
  - and platform

- If t is too big?
- If t is too small?

Recursive vs. Iterative Algorithms

- Tail Recursion can be written iteratively easily
  - When results of recursive call aren't used in an assignment

- In fact, all recursive algorithms can be written as iteration. How?
  - By maintaining your own stack

- Is the iterative implementation better?
  - Saves on stack allocation and function call

- Is the recursive implementation better?
  - More "elegant" to some tastes:
    - more natural, more readable, more maintainable.

- Same algorithmic complexity

Next

- Due by end of day Tuesday:
  - Screencast #7: Solving Homogeneous Recurrence Relations directly
  - Quiz #2 covering screencast #7

- Due Wednesday: HW #4

- Working up to a proof of the master theorem