CSC236 fall 2012
more complexity: mergesort

Danny Heap
heap@cs.toronto.edu
BA4270 (behind elevators)
http://www.cdf.toronto.edu/~heap/236/F12/
416-978-5899

Using *Introduction to the Theory of Computation*,
Chapter 3
Outline

divide and conquer (recombine)

using the Master Theorem

Notes
General case
revisit...

Class of algorithms: partition problem into $b$ roughly equal subproblems, solve, and recombine:

$$T(n) = \begin{cases} k & \text{if } n \leq B \\ a_1 \, T(\lfloor n/b \rfloor) + a_2 \, T(\lceil n/b \rceil) + f(n) & \text{if } n > B \end{cases}$$

where $B, k > 0, a_1, a_2 \geq 0$, and $a_1 + a_2 > 0$. $f(n)$ is the cost of splitting and recombining.

$$a = 2 \quad \quad \frac{b}{2} \quad \quad \Theta(n^d)$$
Master Theorem
(for divide-and-conquer recurrences)

If $f$ from the previous slide has $f \in \theta(n^d)$, then

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

$$\Theta(n^{\log_2 4}) = \Theta(n^2)$$
Proof sketch

1. Unwind the recurrence, and prove a result for $n = b^k$

2. Prove that $T$ is non-decreasing

3. Extend to all $n$, similar to MergeSort
multiply lots of bits
what if they don’t fit into a machine instruction?

Multiplication of $n$-bit numbers.

```
  1101
× 1011
```

- try this in Java, another language.

```
  1101
1101
1101
1101
1101
```

- $n$ copies of $n$-bit numbers

```
  100000
```

- $n$ sums of $n$-bit numbers

```
10001111
```

$\Theta(n^2)$
divide and recombine
recursively...

\[ x = \overline{x_1 \times 2^2} = 1100 \]

\[ x_0 = 11 \]
\[ x_1 = 01 \]
\[ \times 10 \]
\[ \overline{x_0} = 11 \]
\[ x_1 \]
\[ y_0 \]
\[ y_1 \]

\[ xy = 2^n x_1 y_1 + 2^{n/2} (x_1 y_0 + y_1 x_0) + x_0 y_0 \]

\[ = 2^n (10 \times 11) + \]
compare costs

\[ \Theta(n^2) \]

\[ T(n) = \begin{cases} k & n \leq B \\ a_1 T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + a_2 \left(\left\lceil \frac{n}{2} \right\rceil \right) + \Theta & \text{otherwise} \end{cases} \]

\[ \Theta(n) \]

\( n \) \( n \)-bit additions versus:

1. divide each factor (roughly) in half
2. multiply the halves (recursively, if they’re too big)
3. combine the products with shifts and adds

\[ a = 4 \]
\[ b = 2 \]
\[ d = 1 \]

\[ a > b^d \]

\[ \Theta(n \log_b a) \]
Gauss’s trick

\[ xy = 2^n x_1 y_1 + 2^{n/2} (x_1 y_0 + x_0 y_1) \]

\[ xy = 2^n x_1 y_1 + x_0 y_0 + 2^{n/2} ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0) \]

Reduce \( a \) from 4 to 3.
Gauss’s payoff
lose one multiplication

\[ \Theta(n^2) \]

1. divide each factor (roughly) in half
2. sum the halves
3. multiply the sum and the halves Gauss-wise
4. combine the products with shifts and adds

\[ T(n) = \begin{cases} k, & n < B \\ a_1 T\left(\lceil \frac{n}{2} \rceil \right) + a_2 T\left(\lfloor \frac{n}{2} \rfloor \right) + \Theta(n), & \text{otherwise} \end{cases} \]

- \( b = 2 \)
- \( d = 1 \)
- \( a = 3 \)
- \( a > b^d \)
- \( \Theta(n \log_2^3) \sim \Theta(n^{1.5}) \)

Do better with FFT
\[ \frac{n^2}{n^2} = 1 \]