CS 314 : Divide & Conquer

Note Title 2/5/2010

Announcements

- HW posted - due (written) next Friday at start of class

Multiplying two numbers

Let's say we have an $\boxed{\text{n-digit}}$ number

(Aside - how big can that number be?)

$\leq 10^n$

$$
\begin{array}{r}
{}^{1}\,{}^{2}\quad\;\;\; {}^{3}\;\; \\
635219 \\
\times \quad\quad\;\; 4 \\
\hline
2540876
\end{array}
$$

if I give you a number $\leq n$, how many bits does it take to represent?

$\log_2 n$

1 digit

$O(n)$ multiplications

$O(n)$ additions

What about $\boxed{2 \text{ n-digit}}$ numbers?

$$
\begin{array}{r}
625341 \\
\times\ 324951 \\
\hline
625341 \\
+\ 3126705 \\
+\ \ldots \\
+\ \ldots \\
+\ \ldots \\
+\ \ldots \\
\hline
\ldots 291
\end{array}
$$

n numbers to add together

$O(n^2)$ 1-digit

How would we code that algorithm?

Two nested for loops

$O(n^2)$

x x x x x x x x

10^4?

More efficient: Recursion!

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

Ex: 4563 × 2729

$$= (10^2 \cdot \underbrace{45}_{=a} + \underbrace{63}_{=b})(10^2 \cdot \underbrace{27}_{=c} + \underbrace{29}_{=d})$$

Instead of calculating 1 big multiplication, compute $a \cdot c$, $b \cdot c$, $a \cdot d$, & $b \cdot d$

Is this better? ←

$x = 453269$ $\quad a = \boxed{453}\cancel{269}$

Pseudocode

$b = 269$

```
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
```

Running time: $\quad$ to add #5

$$R(n) = 4R\left(\frac{n}{2}\right) + O(n)$$

$$R(1) = 1$$

Exercise: use Master Thm

$$R(n) = O(n^2)$$

A trick: (Karatsuba 1962)

$\underbrace{bc + ad}$ can be computing using only one multiplication!

$$\to \underline{bc + ad} = \overline{ac} + \overline{bd} - \underbrace{\overline{(a-b)(c-d)}}$$

we already compute these!

$$= \cancel{ac} + \cancel{bd} - [\cancel{ac} - bc - ad + \cancel{bd}]$$

$$= bc + ad$$

New + improved pseudo code:

```
FastMultiply(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 ← FastMultiply(a, c, m)
        f ← FastMultiply(b, d, m)
        g ← FastMultiply(a − b, c − d, m)
        return 10^{2m} e + 10^m (e + f − g) + f
```

What's this running time?

$$T(n) = 3T\left(\frac{n}{2}\right) + \underbrace{O(n)}_{\text{additions \& subtractions}}$$

$$T(1) = O(1)$$

$$T(n) = O\left(n^{\log_2 3}\right) = O\left(n^{1.58\ldots}\right)$$

Exponentiation

How do we compute a^n?

$$a^n = \underbrace{a \cdot a \cdot a \cdot a \cdots \cdot a}_{n}$$

$\mathcal{O}(n)$ multiplications

Naive algorithm:

```
SlowPower(a, n):
    x ← a
    for i ← 2 to n
        x ← x · a
    return x
```

Running time? (# of multiplications)

$O(n)$

Faster idea:

$$a^n = a^{\lfloor n/2 \rfloor} \cdot a^{\lceil n/2 \rceil}$$

$$a^6 = a^3 \cdot a^3$$

$$a^{1001} = a^{500} \cdot a^{501}$$

Pseudocode:

```
FastPower(a, n):
    if n = 1
        return a
    else
        x ← FastPower(a, ⌊n/2⌋)
        if n is even
            return x · x
        else
            return x · x · a
```

FastMultiply(x, x)

Running time?

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

$$T(1) = 1$$

$$\Rightarrow T(n) = O(\log_2 n)$$

$$= 2 \log_2 n$$