3/3/15 (0)

Course overview:

webpage: www.cc.gatech.edu/~vigoda/300
lecture schedule & homeworks
- will post my notes there (after class)

No laptop/phone use during class.

3 Midterm exams
see webpage for tentative dates
(may change by ± 1 lecture)

No exam in midterm week

Final exam: yes.

Grades: HW 10%
Midterms 60%
Final 30%

Textbook: Algorithms by Dasgupta, Papadimitriou & Vazirani

HW: Not worth much - so don't cheat! Good practice for exams.
Can work together, but write up solutions on your own.
No late homeworks.

3/3/15

Divide & conquer:

Classic example: Merge Sort

Input: array $A = [a_1, \ldots, a_n]$ of n numbers
Output: sorted array

idea: split A into 2 sublists
recursively sort each sublist
then merge the sorted sublists

MergeSort(A): (assume n is a power of 2)

if $n = 1$, return(A)
if $n > 1$,
$B = [a_1, \ldots, a_{\frac{n}{2}}]$
$C = [a_{\frac{n}{2}+1}, \ldots, a_n]$
D = MergeSort(B)
E = MergeSort(C)
F = Merge(D, E)
Return(F)

Merge takes 2 sorted arrays X & Y
& outputs sorted $Z = X \cup Y$

idea: $x_1 = \min(X_1)$, $y_1 = \min(Y)$
so $\min\{x_1, y_1\}$ is the smallest in $X \cup Y$

<u>Merge</u>(X, Y)

<u>input:</u> $X = [x_1, \ldots, x_k]$ & $Y = [y_1, \ldots, y_l]$ where each is sorted

<u>output:</u> sorted $Z = [z_1, \ldots, z_{k+l}] = X \cup Y$

$i = 1$, $j = 1$, $m = 1$.
while ($i \leq k$ & $j \leq l$) {
 if $x_i \leq y_j$ then [$z_m = x_i$, $i{+}{+}$
  else [$z_m = y_j$, $j{+}{+}$

if $i == k$, return(Z, Y)
if $j == l$, return(Z, X)

Running time:

Merge takes $O(k+l)$ time.

For MergeSort,

let $T(n)$ = running time of MergeSort on worst case input for n numbers

Then,

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

Base case: $T(1) = O(1)$

We'll see that this solves to:

$$T(n) = O(n \log n)$$

Divide & conquer

First, digression for a useful idea from Gauss (~1800)

Setting: multiplication is expensive
but adding/subtracting is cheap.

Have 2 complex numbers:

$$a+bi \quad \& \quad c+di$$

We want to compute their product:

$$(a+bi)(c+di) = ac - bd + (bc+ad)i$$

It looks like we need 4 real number multiplications: ac, bd, bc & ad.

But notice that:

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

So we only need $\underline{3}$ multiplications:

$$ac,\ bd \ \& \ (a+b)(c+d)$$

Typically we assume arithmetic operations (add, subtract, multiply & divide) take $O(1)$ time since we can use hardware implementations.

But in some settings the numbers are too HUGE to do arithmetic operations in hardware. For example in cryptography 512 or 1024 bit long numbers.

Let n = # of bits in the input numbers

~~Look~~ at time for basic arithmetic operations as a function of n.

Adding 2 n-bit numbers x & y

example: $x = 53 = (110101)_2$
$y = 35 = (100011)_2$

$$\begin{array}{r} 110101 \\ +\ 100011 \\ \hline 1011000 \end{array}$$

$\leq n+1$ columns
in each column we add ≤ 3 bits
$\Rightarrow O(1)$ time per column.

$\Rightarrow O(n)$ total time.

Multiplying n-bit numbers x & y

example: $x = 13 = (1101)_2$
$y = 11 = (1011)_2$

$$\begin{array}{r} 1101 \\ \times\ 1011 \\ \hline 1101 \\ 1101 \\ 0000 \\ 1101 \\ \hline 10001111 \end{array}$$

← adding n numbers
each has $\leq 2n-1$ bits
Use previous algorithm this takes
$O(n^2)$ total time

Is this the best we can do?
No, we'll see faster.

Here's an alternative algorithm presented by Al-Khwarizmi – mathematician in Baghdad in 9th century who wrote books on algorithms, e.g., solving quadratic equations
term algorithms comes from the latin form of his name.

Take input x & y.

1) Halve y & double x
   ↑ round down

2) Stop when y = 1.

3) Cross out rows where y is even

4) Add remaining x's.

Example: x = 13 & y = 11

halve y	double x
11	13
5	26
~~2~~	~~52~~
1	+ 104
	143

Why does it work?

Note traditional algorithm does:

$$
\begin{array}{rl}
1101 & = 13 \\
11010 & = 26 \\
000000 & \leftarrow \text{because } 3^{rd} \text{ least significant bit of } y \text{ is even } (0) \\
+\ 1101000 & = 104 \\
\hline
\end{array}
$$

So the 2 algorithms are the same.

Divide & conquer approach:

Assume n is a power of 2.

Input: n-bit numbers x & y.

Divide & conquer idea:

break input into 2 halves.

break x into first $\frac{n}{2}$ bits
& last $\frac{n}{2}$ bits

& y into first $\frac{n}{2}$ bits
& last $\frac{n}{2}$ bits.

$x = \boxed{x_L \mid x_R}$ $\qquad$ $y = \boxed{y_L \mid y_R}$

Example: $x = 182 = (\underbrace{1011}\,\underbrace{0110})_2$

$x_L = 1011$

$x_R = 0110$ $\qquad$ $x = 11 \times 2^4 + 6 = 182$

$$x = 2^{n/2} x_L + x_R$$

So $x = 2^{\frac{n}{2}} x_L + x_R$ & $y = 2^{\frac{n}{2}} y_L + y_R$

Then,

$$xy = \left(2^{\frac{n}{2}} x_L + x_R\right)\left(2^{\frac{n}{2}} y_L + y_R\right)$$
$$= 2^n x_L y_L + 2^{\frac{n}{2}}\left(x_L y_R + x_R y_L\right) + x_R y_R.$$

Easy idea:

recursively compute $x_L y_L$
$x_L y_R$
$x_R y_L$
$x_R y_R$

Then get xy by adding & shifting.

Let $T(n)$ = worst case running time for input of size n

It takes $O(n)$ time to break x into x_L, x_R
& y into y_L, y_R

& $O(n)$ time to combine
$x_L y_L, x_L y_R, x_R y_L$ & $x_R y_R$
to get xy

& 4 subproblems each is multiplying 2 $\frac{n}{2}$ bit numbers

Hence, $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$

We'll see in next class that this solves to:

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

So same as classical approach.

Can we do it using only 3 subproblems?

Use earlier idea of Gauss:

let $a = X_L$
$b = X_R$
$c = Y_L$
$d = Y_R$

then $bc + ad = X_R Y_L + X_L Y_R$

the earlier idea was that:

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

$= X_R Y_L + X_L Y_R \qquad (X_L + X_R)(Y_L + Y_R) - X_L Y_L - X_R Y_R$

Thus, $x_R y_L + x_L y_R = (x_L + x_R)(y_L + y_R) - x_L y_L - x_R y_R$

So we recursively solve 3 subproblems:

$$x_L y_L,\ x_R y_R,\ \&\ (x_L + x_R)(y_L + y_R)$$

FastMultiply(x, y):

- x_L = leftmost $\frac{n}{2}$ bits of x
- x_R = rightmost $\frac{n}{2}$ bits of x
- y_L = leftmost $\frac{n}{2}$ bits of y
- y_R = rightmost $\frac{n}{2}$ bits of y
- A = FastMultiply(x_L, y_L)
- B = FastMultiply(x_R, y_R)
- C = FastMultiply$(x_L + x_R, y_L + y_R)$
- return$\left(A \times 2^n + (\cancel{B}C - A - B) \times 2^{n/2} + B\right)$

Running time:

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) = O\left(n^{\log_2 3}\right)$$

we'll see next class

note $\log_2 3 \approx 1.59$