Lecture 3 Divide and Conquer (cont'd)

September 5th, 2017

Scribe: Zachary Liu

1 Integer Multiplication

Input: Two positive integers, a and b.
Output: $a * b$
The naive approach is to simply multiply the two numbers which takes $O(n^2)$ (long multiplication).

$$\begin{array}{r} 3\,8\,4 \\ \times \quad 5\,6 \\ \hline 2\,3\,0\,4 \\ 1\,9\,2\,0\;\;\; \\ \hline 2\,1\,5\,0\,4 \end{array}$$

1.1 1st Attempt

Break both numbers into high digits and low digits. Recursively compute the products. See the following algorithm:

Multiply(a,b)

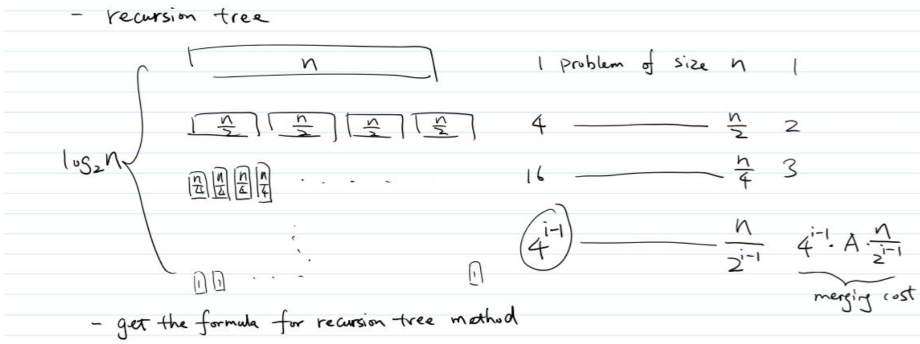
1. wlog assume $n = length(a) = length(b)$, can pad 0's for shorter number
2. if $length(a) <= 1$: return $a * b$
3. Partition a,b into $a = a1 * 10^{n/2} + a2 \quad b = b1 * 10^{n/2} + b2$
4. $A = Multiply(a1, b1)$
5. $B = Multiply(a2, b1)$
6. $C = Multiply(a1, b2)$
7. $D = Multiply(a2, b2)$
8. Return $A * 10^n + (B + C) * 10^{n/2} + D$

Recursion: Let $T(n)$ be the running time to multiply 2 n-digit numbers, we have

$$T(n) = 4T(n/2) + A * n,$$

where A is a constant for the merging procedure.

Analyzing Runtime Using Recursion Tree See the following Recursive Tree



$T(n) = 4T(n/2) + A * n$ — $A * n$ indicates the merging cost at layer 1

$= 16T(n/4) + 4 * A * n/2 + A * n$ — (because $T(n/2) = 4T(n/4) + A * n/2$)

(Note: $4 * A * n/2$ indicates the merging cost for layer 2)

$= 64T(n/8) + 16 * A * n/4 + 4 * A * n/2 + A * n$ — ($T(n/4) = 4T(n/8) + A * n/4$)

(Note: $16 * A * n/4$ indicates merging cost for layer 3)

$= \cdots$

$$= 4^{\log_2 n} T(1) + \sum_{i=1}^{\log_2 n} 4^{i-1} \times A \times \frac{n}{2^{i-1}}$$

(Note: $4^{i-1} \times A \times \dfrac{n}{2^{i-1}}$ is the merging cost for layer i)

We can assume $T(1) = 0$ (doesn't change asymptotic behavior)
Overall cost of the function is the sum of the merging cost of all layers

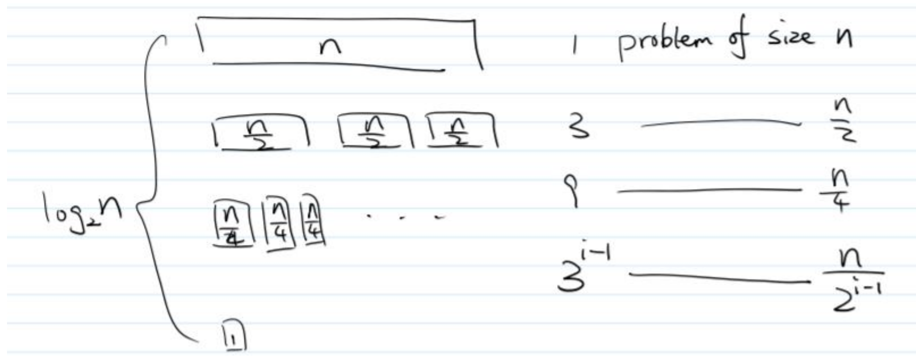
$$\begin{aligned} T(n) &= \sum_{i=1}^{numlayers} \text{merging cost at layer } i \\ &= \sum_{i=1}^{\log_2(n)} 4^{i-1} A \frac{n}{2^{i-1}} \\ &= An \sum_{i=1}^{\log_2(n)} 2^{i-1} \\ &= An(n-1) = O(n^2) \end{aligned}$$

1.2 Improved Algorithm

Observation: $(a1 * b2 + a2 * b1) = (a1 + a2) * (b1 + b2) - (a1 * b1) - (a2 * b2)$
Improved Algorithm as follows:

Multiply(a,b)
1. wlog assume $n = length(a) = length(b)$, can pad 0's for shorter number
2. if $length(a) <= 1$: return $a * b$
3. Partition a,b into $a = a1 * 10^{n/2} + a2 b = b1 * 10^{n/2} + b2$
4. $A = Multiply(a1, b1)$
5. $B = Multiply(a2, b2)$
6. $C = Multiply(a1 + a1, b1 + b2)$
7. Return $A * 10^n + (C - A - B) * 10^{n/2} + B$

Recursion: $T(n) = 3T(\frac{n}{2}) + An$ —— See the following diagram



by recursion tree:

$$T(n) = \sum_{i=1}^{\text{num layers}} \text{merging cost at layer } i \tag{1}$$
$$= \sum_{i=1}^{\log_2 n} 3^{i-1} A \frac{n}{2^{i-1}} \tag{2}$$
$$= An \sum_{i=1}^{\log_2(n)} \left(\frac{3}{2}\right)^{i-1} \tag{3}$$

Note that the sum $k^0 + k^1 + k^2 ... k^{(log_2 n)-1} = \Theta(k^{\log_2 n})$ when $k > 1$, so we have

$$
\begin{aligned}
An \sum_{i=1}^{\log_2(n)} \left(\frac{3}{2}\right)^{i-1} &= An\Theta(\left(\frac{3}{2}\right)^{\log_2(n)}) \\
&= \Theta(An * n^{\log_2(3)}) \\
&= \Theta(An * n^{\log_2(3/2)}) \\
&= \Theta(An * n^{\log_2(3)-1}) \\
&= \Theta(An^{\log_2(3)}) \\
&\approx O(n^{1/585}).
\end{aligned}
$$

Note: $\left(\frac{3}{2}\right)^{\log_2(n)} = 2^{\log_2[\left(\frac{3}{2}\right)^{\log_2(n)}]}$ because $A = 2^{\log_2 A}$.
$2^{\log_2[\frac{3}{2})^{\log_2(n)}]} = 2^{\log_2(\frac{3}{2})\log_2(n)}$ because $\log A^B = B \log A$.
$2^{\log_2(\frac{3}{2})\log_2(n)} = 2^{\log_2(n)*\log_2(\frac{3}{2})} = n^{\log_2(\frac{3}{2})}$ because $2^{AB} = (2^A)^B$ and $2^{\log_2 n} = n$.

2 Master Theorem

The Master Theorem acts as a "cheat sheet" for basic recursions

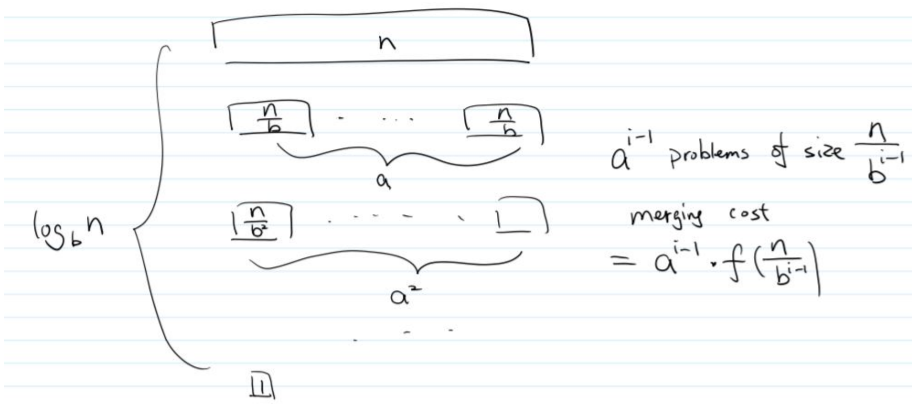
Theorem 1. *Given* $T(n) = aT(\frac{n}{b}) + f(n))$
1. If $f(n) = O(n^c), c < \log_b(a)$ *then* $T(n) = \Theta(n^{\log_b(a)}$
2. If $f(n) = \Theta(n^c \log^t(n)), c < \log_b(a)$ *then* $T(n) = \Theta(n^{\log_b(a)} \log^{t+1}(n))$
3. If $f(n) = \Theta(n^c), c > log_b(a)$ *then* $T(n) = \Theta(n^c)$

Intuition: The three cases of the Master Theorem correspond to 3 possible cases in the recursion tree method:

1. The merging cost is multiplied by a factor greater than 1 every time. The total cost is dominated by the cost of last layer.
2. The merging cost is roughly the same between layers. The total cost is equal to merging cost per layer, multiplied by number of layers.
3. The merging cost is multiplies by a factor smaller than 1 every time. The total cost is dominated by the merging cost of first layer.

Proof Sketch:

$$T(n) = aT(\tfrac{n}{b}) + f(n)$$



$$
\begin{aligned}
T(n) &= \sum_{i=1}^{\text{num layers}} \text{merging cost at layer i} \\
&= \sum_{i=1}^{\text{num layers}} a^{i-1} f(\frac{n}{b^{i-1}})
\end{aligned}
$$

If we assume that $f(n) = n^c$

$$
\begin{aligned}
&= \sum_{i=1}^{\text{num layers}} a^{i-1} (\frac{n^c}{b^{ci-1}}) \\
&= \sum_{i=1}^{\text{num layers}} n^c \left(\frac{a}{b^c}\right)^{i-i}
\end{aligned}
$$

Now we can see the merging cost for each layer is a factor $\frac{a}{b^c}$ multiplied by the merging cost for the previous layer. Depending on the comparison of this number and 1, we have the three cases:

$$
\frac{a}{b^c} \begin{cases} = 1 \text{ Case 2 in Thm } n^c \log n \\ < 1 \text{ Case 3 in Thm } n^c \\ > 1 \text{ Case 1 in Thm } n^{\log_b(a)} \end{cases}
$$