Lecture 3: Divide and Conquer 2

Scriber: Xiaoming Liu

January 27, 2020

1 Integer Multiplications

Problem statement: Given two n-digit numbers x and y, find their multiplication.

1.1 Naive Recursive Approach

Suppose we are given $a = 123456$ and $b = 654321$. While a and b can be rewritten as $a = 123 * 1000 + 456$ and $b = 654 * 1000 + 321$ respectively, and thus the multiplication can be rewritten as $a * b = 123 * 654 * 106 + (123 * 321 + 456 * 654) * 103 + 456 * 321$.

To generalize the multiplication, we assume n is a power of 2 without the loss of generality and we can partition a and b respectively into their upper and lower digits, i.e, $a = a_{upper} * 10^{n/2} + a_{lower}$ and $b = b_{upper} * 10^{n/2} + b_{lower}$.

The recursive multiplication algorithm is thus:

Algorithm 1 Recursion

Result: multiplication of **a** and **b**

Assume n = length(a) = length(b). Pad 0's for shorter number;

if *length(a) ¡= 1* **then**
  return a * b;
else
  partition a into $a = a_{upper} * 10^{n/2} + a_{lower}$
  partition b into $b = b_{upper} * 10^{n/2} + b_{lower}$
  $A = Recursion(a_{upper}, b_{upper})$
  $B = Recursion(a_{lower}, b_{upper})$
  $C = Recursion(a_{upper}, b_{lower})$
  $D = Recursion(a_{lower}, b_{lower})$
  return $A * 10^n + (B + C) * 10^(n/2) + D$
end

The time complexity of the algorithm can thus be represented as:

$$T(n) = 4T(\frac{n}{2}) + O(n)$$

The recursion tree can be illustrated as follows:

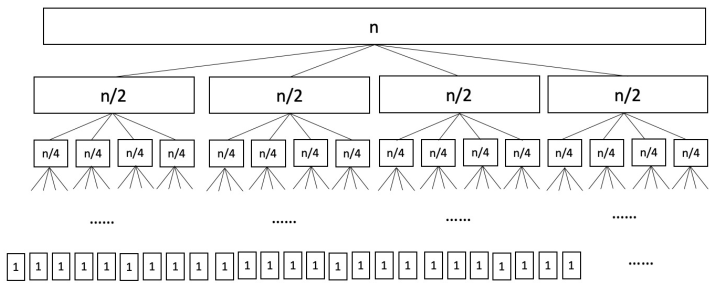

Figure 1: Recursion Tree

As illustrated in the figure above, the recursion tree has a depth of log_2^n. The overall complexity is thus:

$$\begin{aligned} T(n) &= \sum_{i=0}^{log_2^n} 4^i A \frac{n}{2^i} \\ &= An \sum_{i=0}^{log_2^n} 2^i \\ &= An(2n-1) \\ &= O(n^2) \end{aligned} \quad (1)$$

1.2 Improved Recursive Approach

We can improve the algorithm by doing one of the following:

1. Merging faster: However, this is not the bottleneck for integer multiplication. O(n) is not large.
2. Make subproblems smaller: If we do this naively, then that would result in more number of subproblems which defeats the purpose.
3. Decrease the number of subproblem: We see the details below.

The improved algorithm is as follows:

Algorithm 2 Recursion

Result: multiplication of **a** and **b**
Assume n = length(a) = length(b). Pad 0's for shorter number;

if *length(a) ¡= 1* **then**
  return a * b;
else
  partition a into $a = a_{upper} * 10^{n/2} + a_{lower}$
  partition b into $b = b_{upper} * 10^{n/2} + b_{lower}$
  $A = Recursion(a_{upper}, b_{upper})$
  $B = Recursion(a_{lower}, b_{lower})$
  $C = Recursion(a_{upper} + a_{lower}, b_{upper} + b_{lower})$ return $A * 10^n + (C - A - B) * 10^(n/2) + B$
end

The time complexity of the algorithm can thus be represented as:

$$T(n) = 3T(\frac{n}{2}) + O(n)$$

Thus,

$$\begin{aligned} T(n) &= \sum_{i=0}^{log_2^n} 3^i A \frac{n}{2^i} \\ &= An \sum_{i=0}^{log_2^n} (\frac{3}{2})^i \\ &= O(n \frac{3}{2}^{log_2^{\frac{3}{2}}}) \\ &= O(n^{log_2^3}) \\ &= O(n^1.585) << O(n^2) \end{aligned} \tag{2}$$

1.3 Master Theorem

Theorem: If $T(n) = aT(n/b) + f(n)$, then

1. $f(n) = O(n^c), c < log_b^a$, then $T(n) = \Theta(n^{log_b^a})$
2. $f(n) = \Theta(n^c log^t(n)), c = log_b^a$, then $T(n) = \Theta(n^{log_b^a} log^{t+1}(n))$
3. $f(n) = \Theta(n^c), c > log_b^a$ then $T(n) = \Theta(n^c)$

For case 1 and case 3 of the master theorem, the recursion tree can be illustrated as follows. The recursion tree can be illustrated as follows:

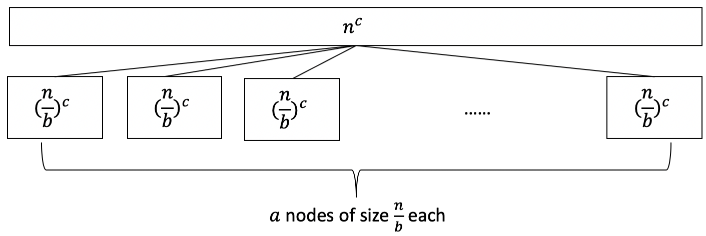

Figure 2: Generalized Recursion Tree for case 1 and 3

For case 2 of the master theorem, the recursion tree can be illustrated as follows. The recursion tree can be illustrated as follows:

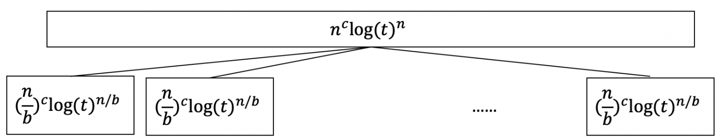

Figure 3: Generalized Recursion Tree for case 2