Divide & Conquer

Lecture 10

The *divide-and-conquer* technique involves solving a particular computational problem by **dividing it into one or more subproblems** of smaller size, recursively solving each subproblem, and then “**merging”** or “marrying” the solutions to the subproblem(s) to produce a solution to the original problem.

We can model the divide-and-conquer approach by using a parameter n to denote the size of the original problem, and let S(n) denote this problem.

We solve the problem S(n) by solving a collection of k subproblems S(n1), S(n2), ..., S(nk), where ni < n for i = 1,...,k, and then merging the solutions to these subproblems.

To analyze the running time of a divide-and-conquer algorithm, we often use a ***recurrence equation***

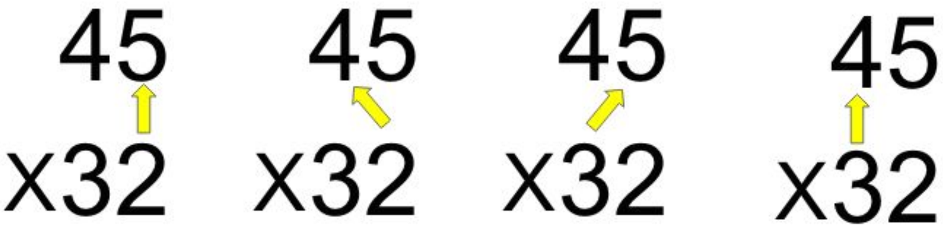
Eg: Merge Sort, Quick Sort.

Integer Multiplication

Input: Two positive n digit integers, x and y.

Output: $x * y$

Suppose we are given x = 45 and y=32



Here's the naive multiplication algorithm to multiply two *n*-bit numbers, *x* and *y* that are in base *b* takes $O(n^2)$.

How can we improve?

Karatsuba Algorithm

The Karatsuba algorithm provides a striking example of how the “Divide and Conquer” technique can achieve an asymptotic speedup over an ancient algorithm.

We assume that the number of digits is a power of 2 so we will not need to worry about rounding. This assumption is justified by padding the number with initial zeros; this will increase the value of n by less than a factor of 2.

To multiply two n-bit numbers, x and y, the Karatsuba algorithm performs three multiplications and a few additions, and shifts on smaller numbers that are roughly half the size of the original x and y.

Partition a,b into $a = a1 * 10^{n/2} + a2$ *and*

$b = b1 * 10^{n/2} + b2$

$A = Multiply(a1, b1)$

$B = Multiply(a2, b2)$

$C = Multiply(a1 + a2, b1 + b2)$

$d = (C - A - B)$

Return $A * 10^{n} + d * 10^{n/2} + B$

Perform the following multiplication using the Karatsuba method**: 1234 × 4321**

First, determine the A value for step 1,
A--this will contain the high bits of x and y since x and y have four bits, and the le -most two are the high bits.
$A = 12 \times 43$. Note, we will have to call the Karatsuba algorithm on $a1$ since a multiplication is necessary to obtain the value (this time, a two-bit multiplication). Before we recurse, though, let's find B and d.

B contains the lower bits of each number since x and y have four bits, and the lower bits in this problem are the two rightmost bits.

$B = 34 \times 21$. Note, we will also have to recurse on $d1$ to obtain the value.

Recall that $C = (xH + xL)(yH + yL) - a - d$.

$d = (12 + 34) \times (43 + 21) - A - B$.

Now we are stuck and can't simplify d further until we have the values of A and B, so it is time to recurse.

Solving for a_1 :

We have

$$\begin{aligned} a_1 &= 12 \times 43 \\ a_2 &= 1 \times 4 = 4 \\ d_2 &= 2 \times 3 = 6 \\ e_2 &= (1+2)(4+3) - a_2 - d_2 \\ &= (1+2)(4+3) - 4 - 6 \\ &= 11. \end{aligned}$$

Recall that $xy = ar^n + er^{\frac{n}{2}} + d$. Therefore, $a_1 = 12 \times 43 = 4 \times 10^2 + 11 \times 10 + 6 = 516.$

Solving for d_1 :

We have

$$\begin{aligned} d_1 &= 34 \times 21 \\ a_2 &= 3 \times 2 = 6 \\ d_2 &= 4 \times 1 = 4 \\ e_2 &= (3+4)(2+1) - a_2 - d_2 \\ &= 11. \end{aligned}$$

Since $xy = ar^n + er^{\frac{n}{2}} + d$, $d_1 = 34 \times 21 = 6 \times 10^2 + 11 \times 10 + 4 = 714.$

Solving for e_1 :

We have

$$\begin{aligned} e_1 &= (46 \times 64) - a_1 - d_1 \\ a_2 &= 4 \times 6 = 24 \\ d_2 &= 6 \times 4 = 24 \\ e_2 &= (4+6)(6+4) - a_2 - d_2 \\ &= 52. \end{aligned}$$

Since $xy = ar^n + er^{\frac{n}{2}} + d$, $e_1 = (46 \times 64) - a_1 - d_1 = 24 \times 10^2 + 52 \times 10 + 24 - 714 - 516 = 1714.$

Now we can get the answer to the original problem as follows:

We have

$$a_1 = 516, \quad d_1 = 714, \quad e_1 = 1714.$$

Plugging into $xy = ar^n + er^{\frac{n}{2}} + d$, we get

$$xy = (516)10^4 + (1714)10^2 + 714 = 5,332,114$$

Procedure Karatsuba(X, Y)

Input: X, Y : n-digit integers.

Output: the product P := XY .

Comment: We assume n is a power of 2.

1. if n = 1 then use multiplication table to find P := XY

2. else split X, Y in half:

3. X =: $10^{n/2}$X1 + X2

4. Y =: $10^{n/2}$Y1 + Y2

5. Comment: X1, X2, Y1, Y2 each have n/2 digits.

6. U := Karatsuba(X1, Y1)

7. V := Karatsuba(X2, Y2)

8. W := Karatsuba(X1 +X2, Y1 +Y2)

9. Z := W -U-V

10. P := 10^{n}U + $10^{n/2}$Z + V

Finally we conclude that P = 10^{n}X1Y1 + $10^{n/2}$ (X1Y2 + X2Y1) + X2Y2 = XY .

11. return P

Analysis

This is a recursive algorithm: during execution, it calls smaller instances of itself. Let $M(n)$ denote the number of digit-multiplications (line 1) required by the Karatsuba algorithm when multiplying two n-digit integers ($n = 2^k$).

In lines 6,7,8 the procedure calls itself three times on n/2-digit integers;

Therefore $M(n) = 3M(n/2)$. -------------------------(1)

This equation is a simple recurrence which we may solve directly as follows. Applying equation (1) to $M(n/2)$ we obtain

$M(n/2) = 3M(n/4)$;

therefore $M(n) = 9M(n/4)$.

Continuing similarly we see that

$M(n) = 27M(n/8)$, and it follows by induction on i that for every i ($i \le k$),

$M(n) = 3^i M(n/2^i)$

Setting i = k we find that

$$M(n) = 3^k M(n/2^k) = 3^k M(1)$$

$= 3^k$.

Notice that $k = \log n$ (recall: in this note, log refers to base-2 logarithm), therefore $\log M(n) = k \log 3$ and

hence $M(n) = 2^{\log M(n)} = 2^{k \log 3} = (2^k)^{\log 3} = n^{\log 3}$

It would seem that we reduced the number of digit-multiplications to $n^{\log 3}$ at the cost of an increased number of additions (lines 9, 10).

To see this, let T(n) be the total number of digit-operations (additions, multiplications, bookkeeping (copying digits, maintaining links)) required by the Karatsuba algorithm.

Then $T(n) = 3T(n/2) + O(n)$ --------------------------- (2)

where the term 3T(n/2) comes, as before, from lines 6,7,8;

the additional O(n) term is the number of digit-additions required to perform the additions and subtractions in lines 9 and 10.

The O(n) term also includes bookkeeping costs. We shall learn later how to analyse recurrences of the form (2).

It turns out that the additive O(n) term does not change the rate of growth, and the result will still be

$$\mathbf{T(n) = O(n^{\log 3})} \text{ ------------------ } \quad (3)$$