CS 170 Lecture Notes, Fall 2020

Algorithms and Intractable Problems

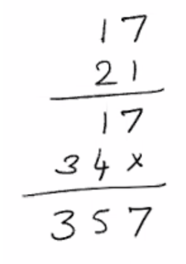
Professors: Avishay Tal, Umesh Vazirani
Scribe: Vishal Raman

Contents

§1 September 1st, 2020

§1.1 Naive Multiplication

Recall the example of Fibonacci: we went from a complexity of $O(2^n) \to O(n^2) \to O(f(n))$, where $f(n)$ is the runtime for multiplying n-bit numbers. The naive algorithm is the usual multiplication algorithm for multiplying by hand.



Listing 1: Naive n-bit multiplication

```
function multiply(x, y):
Input: n-bit integers, x, y, y >= 0
Output: Product

    if (y == 0) return 0;
    s = multiply(x, floor(y/2))
    if y is even:
        return 2z
    else
        return x+2z
```

If we write $x = \sum_{i=0}^{n-1} x_j 2^i, y = \sum_{i=0}^{n-1} y_i 2^i$, so

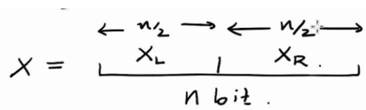
$$xy = \left(\sum_{i=0}^{n-1} x_j 2^i\right)\left(\sum_{i=0}^{n-1} y_i 2^i\right) = \sum_{j,k=0}^{n} x_j y_k 2^{j+k}.$$

§1.2 Divide and Conquer: Karatsuba's Algorithm

For a Divide and Conquer problem, we do the following:

1. Break problem into pieces.
2. Solve pieces recursively.
3. Glue solutions of pieces to get solution of original problem.

We first have x an n-bit number that we break up into x_L, x_R, each $n/2$ bit numbers. Similarly, we break y into y_L, y_R.



Note that $x = 2^{n/2}x_L + x_R, y = 2^{n/2}y_L + y_R,$, so

$$xy = 2^n x_L x_L + 2^{n/2}(x_L y_R + x_R y + L) + x_R y_R.$$

We now have 4 multiplications, involving $n/2$-bit numbers. Multiplication by 2^m can be shifting ($O(m)$ time), and addition is $O(1)$. Hence, we have a recurrence equation for the runtime,

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

Note that the depth of the recursion tree is $\log n$ and there are $4^{\log n} = n^2$ leaves. But this would have the same runtime as the naive algorithm, so more work is required to optimize.

We note the following trick from Gauss:

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

and note that

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

so it suffices to compute $ac, bd, (a + b)(c + d)$, since $(a + b)(c + d) - ac - bd = ad + bc$, which is the other term.

We can apply the trick in the following way:

$$(2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L x_L + 2^{n/2}(x_L y_R + x_R y + L) + x_R y_R$$

and

$$(x_L + x_R)(y_L + y_R) = x_L y_L + x_R y_R + (x_L y_R + y_L x_R).$$

Our new recurrence relation is

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

Now, the branching factor is only 3, so the number of leaves is given by

$$3^{\log n} = n^{\log 3}.$$

§1.3 Master Theorem

We have the following method of solving recurrence relations:

Theorem 1 (Master Theorem)
If $T(n) = aT(\lceil n/b \rceil) + O(n^d)$ for constants $a > 0, b > 1, d \geq 0$, then

$$T(n) = \begin{cases} O(n^d) & \text{if } d > \log_b a \\ O(n^d \log n) & \text{if } d = log_b a \\ O(n^{\log_b a}) & \text{if } d < log_b a \end{cases}$$