• DISCLAIMER: These notes are not necessarily an accurate representation of what I said during the class. They are mostly what I intend to say, and have not been carefully edited.

• Main topics:
  - Why \((a + b)(c + d) - ac - bd\): want \(ac + bd\), but already have \(ac\) and \(bd\).
  - Recursion tree based analysis of \(T(n) = 3T(n/2) + O(n)\).
    * Level \(i\): \(3^i\) calls to inputs with size \(n/2^i\).
    * Number of levels: \(L = \log_2 n = \log n / \log 2\).
    * Total size of the inputs:
      \[
      n + 3 \frac{n}{2} + 3^2 \frac{n}{2^2} \ldots + 3^L \frac{n}{2^L} = n \left(1 + \frac{3}{2} + \left(\frac{3}{2}\right)^2 + \ldots \left(\frac{3}{2}\right)^L\right)
      \]
    * Fact about geometric series: for a constant \(t > 1\), \(1 + t + t^2 + \ldots + t^L = \Theta(t^L)\).
  - Bit-packing:
    * Set of sums = non-zeros in product without carry.
    * Elements coming from multiple sets.

• Comments from last time
  - 38 responses :-).
  - How did people come up with that ‘return’ from Karatsuba?
  - Recursion tree unclear.
  - Definition of polynomial

**A Faster Multiplication Algorithm**

Once we express two the two numbers being multiplied as

\[
10^{n/2}a + b,
\]

and

\[
10^{n/2}c + d,
\]
their product can be expressed as

\[ 10^n ac + 10^{n/2} (ad + bc) + bd. \]

This at a glance requires computing \( ac, ad, bc, \) and \( bd, \) which is four multiplications. A key observation that leads to a better algorithm is that the two middle ‘cross terms’ \( ad \) and \( bc \) both have coefficients \( 10^{n/2} \). So it suffices to directly find

\[ ad + bc. \]

Since we already have \( ac \) and \( bd \) pre-computed, we can obtain it from \((a+b) \times (c+d)\) by:

\[ ad + bc = (a + b) \times (c + d) - ac - bd. \]

\((a + b)\) is ‘simpler’ than \( a \times 10^{n/2} + b \) because both \( a \) and \( b \) have \( n/2 \)-digits, so their sum has at most \( n/2 + 1 \) digits. This is only half as many as the original number which has \( n \) digits.

This means we only need to recursively compute the products \( ac, bd, \) and \((a+b)(c+d)\), gives the recurrence

\[ T(n) = 3T(n/2) + O(n). \]

**Recursion Trees**

The running time of this procedure can be analyzed by considering the recursion tree and bounding its branching factor in the limit. The textbook has such an example in Figure 2.3. The idea is to explicitly count the number of problems at each ‘level’ of the recursion. In this instance we have:

- 1 problem at the top level of size \( n, \)
- 3 instances at the second level of size \( n/2, \)
- 9 instances at the second level of size \( n/4 \ldots \)
- \( 3^i \) instances at the second level of size \( 2^{-i}n \).

The total size at each level is then

\[ n, \frac{3}{2} n, \left(\frac{3}{2}\right)^2 n, \ldots \left(\frac{3}{2}\right)^i n \ldots, \]

which is geometrically increasing. This gives a geometric series

\[ 1 + t + t^2 + \ldots + t^i. \]

This sum can be evaluated by assuming the total is \( x \). Multiplying \( x \) by \( t \) gives:

\[ tx = t + t^2 + \ldots + t^{i+1} = x - 1 + t^{i+1}, \]

\[ 2 \]
or
\[ x = \frac{t^{i+1} - 1}{t - 1} \]

Since \( t \) is a constant strictly greater than 1, this quantity is \( \Theta(t^i) \).

In our case, the depth of the recursive calls satisfies
\[ i \leq \log_2 n, \]
so the total work is
\[ T(n) \leq \Theta(3 \log_2 n). \]

Some arithmetic manipulations (take log, rearrange the multiplication, and exponentiate everything again) gives that this is equal to \( \Theta(n^{\log_2 3}) \):
\[
\log(3 \log_2 n) = \log 3 \log_2 n = \frac{\log 3 \log n}{\log 2} = \log n \frac{\log 3}{\log 2} = \log n \log_2 3.
\]

**More Bit Packing**

We finish with more ‘bit packing’, or encoding values as coefficients of polynomials. For those familiar with generating functions, this is exactly the same setup.

Formally ‘multiplying without carry’ is identical to polynomial multiplication. Instead of numbers, the digits are treated as coefficients of \( x^i \) in some polynomial
\[ P(x) = \sum_i p_i x^i. \]

In fact, another way to think about base 10 is that we’re simply evaluating the value of \( P(10) \). The products of two polynomials is obtained by combining together powers of \( x \), aka
\[
P(x)Q(x) = \left( \sum_i p_i x^i \right) \left( \sum_j q_j x^j \right) = \sum_{i,j} p_i q_j x^{i+j}.
\]

So if we go collect coefficients by values of \( x^{i+j} \), we end up with \( \sum_i p_i q_{k-i} \), which is exactly the same as the formula for multiplication.

An important properties of this is that the exponents on \( x \) add, while the coefficients multiply. This enables us to build representations of sums of elements from sets.

The mcnuggets problem that we talked about last time sets \( p_i = 1 \) if \( i \in S \). Then we have
\[
P^2 = \left( \sum_i p_i x_i \right) \left( \sum_j p_j x_j \right) = \sum_i \sum_j p_i p_j x^{i+j}.
\]

That is, the coefficient of \( x^k \) in \( P^2 \) is non-zero only if there exists \( i, j \) in \( S \) such that \( i + j = k \). In other words, the set of \( k \) with non-zero coefficient on \( x^k \) is precisely the set of numbers that can be formed by summing two entries in \( S \).
The example that we did was $S = \{1, 2, 5\}$, to which we get

$$100110^2 = 10022012100.$$ 

This can also work with many sets. Suppose instead of having infinite boxes of each size, we can use each box size at most once. Then if we don’t have a limit on the number of boxes that we can use, we can multiply all the corresponding numbers/polynomials to form the set of values that can be made using a subset of them. Note that $x^0$ has coefficient 1 because not using a box is now an option:

$$(1 + x)(1 + x^2)(1 + x^5) = 1 + x + x^2 + x^3 + x^5 + x^6 + x^7 + x^8,$$

or with numbers

$$11 \times 101 \times 100001 = 111101111.$$ 

In general, if we have box sizes that sum up to $m$, we can still use divide-and-conquer on the total sizes to obtain a routine that computes all possible subset sums in sub-quadratic time. We won’t be able to go into such a routine in detail though.