Formal Definitions

To show that \( n^2 + n = O(n^3) \), choose \( c = 1 \) and \( n_0 = 2 \). Since at \( n = 2 \) we have \( n^3 > n^2 \), and they are not equal for any value of \( n \) greater than 2, it follows that \( n^2 + n < cn^3 \) for \( n > n_0 \).

To show that \( n^2 + n \neq \Theta(n^3) \), it’s enough to show that \( n^2 + n \neq \Omega(n^3) \). We can do that by contradiction: assume that we chose our constant \( c \). We can solve \( n^2 + n = cn^3 \) and find that the largest value at which they are equal is \( n = \frac{1+\sqrt{1+4c}}{2c} \), and (plugging in any greater value of \( n \)) that \( n^2 + n < cn^3 \). This means that, if we choose some \( n_0 \), we can look at \( n = \max\{n_0, \frac{1+\sqrt{1+4c}}{2c}\} + 1 \). This is certainly greater than \( n_0 \), but for this value of \( n \) we have that \( n^2 + n < cn^3 \).

Recurrences

The recurrence is bounded above by the solution to \( T(n) = 4T(n/5) + n \) and below by \( 4T(n/6) + n \) — having more levels of recursion, in this case, increases the value of the function.

As we know from class, based on the argument with trees, both of these are in \( \Theta(n) \); it follows that \( T(n) \) itself is.

Multiplying Polynomials

Break up \( f(x) \) as \( x^{n/2}f_1(x) + f_2(x) \) and \( g(x) \) as \( x^{n/2}g_1(x) + g_2(x) \) (in practice, we generally have floors; we’re going to ignore them since they make things messy. In practice, we don’t usually have to worry much about them).

We have \( f(x)g(x) = x^nf_1(x)g_1(x) + x^{n/2}(f_1(x)g_2(x) + f_2(x)g_1(x)) + f_2(x)g_2(x) \).

By using one multiplication of degree-\( n/2 \) polynomials, we can compute \( (f_1(x) + f_2(x))(g_1(x) + g_2(x)) = f_1(x)g_1(x) + f_1(x)g_2(x) + f_2(x)g_1(x) + f_2(x)g_2(x) \). We can also compute \( f_1(x)g_1(x) \), the first term in the product above, and \( f_2(x)g_2(x) \), which is the last. By subtracting, we can get the middle term. This means that we can compute the product of two degree-\( n \) polynomials with three multiplications of \( n/2 \)-bit polynomials.

This means that the recurrence of the running time is \( T(n) = 3T(n/2) + cn \), where \( c \) is some constant (for our additions). The solution to this is asymptotically less than \( n^2 \).

Group Problems

Bad Linked Lists

Start two pointers \( p_1 \) and \( p_2 \) at the beginning of the list. Each iteration, advance \( p_1 \) by one node, and \( p_2 \) by two nodes. If \( p_2 \) ever passes \( p_1 \), we know that there is a cycle; if \( p_2 \) ends up reaching \texttt{NULL}, we know that there was no loop.

If the loop consists of \( n \) elements then, once both pointers are within the loop, it will be fewer than \( n \) more cycles until \( p_2 \) passes \( p_1 \) — this is because, every time \( p_1 \) goes around the cycle once (\( n \) iterations), \( p_2 \) goes around twice.

Therefore, if there are \( m \) nodes before the cycle starts, we have at most \( m + n \) iterations before \( p_2 \) passes \( p_1 \) — this is less than the number of nodes in the list; therefore, this algorithm runs in linear time.
Complex Multiplication

Compute \((a + b)(c + d) = ac + ad + bc + bd\) with one multiplication, \(ac\) with another, and \(bd\) with the third. The real part (the part that is not multiplied by \(i\)) is \(ac - bd\) (and we’ve already calculated both terms); the imaginary part (which is multiplied by \(i\)) is \((ac + ad + bc + bd) - ac - bd\) — again, we’ve already computed all three terms, so once again we don’t need any further multiplications.

$1,000,000$

The running time must be polynomial in the size of the input. \(c\) can be represented in \(\log_2(c)\) bits, meaning that an algorithm that runs in time \(O(c^2)\) is exponential in the size of the input \(O(2^{(\log_2(c))^2})\). Sadly, it’s a bit harder than that to get rich in theoretical computer science. ☺