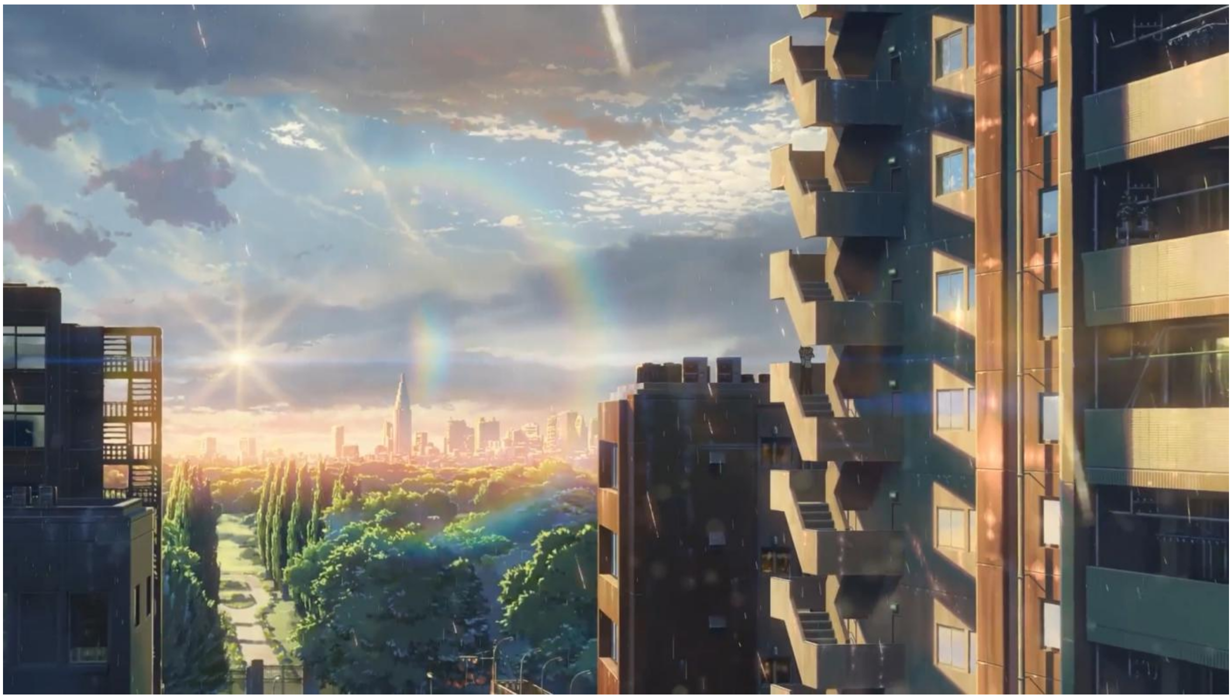
COMPSCI 611 Advanced Algorithm

Email: zhumaxy@gmail.com

Amherst, MA, 01002



Polynomial Multiplication

Now we consider the problem of multiplying two polynomials again. Given that two polynomials $A(x) = \sum_{i=0}^{n-1} a_i x^i$ and $B(x) = \sum_{i=0}^{n-1} b_i x^i$, both of degree $n-1$, compute $C(x) = A(x) \cdot B(x)$. In the lecture, we learned that using FFT, we could compute $C(x)$ using $O(nlogn)$ operations on real numbers. However, if $A(x)$ and $B(x)$ both have integer coefficients, we typically want to multiply $A(x)$ and $B(x)$ using operations on integers only. Design a divide and conquer that only uses operations on integers to multiply $A(x)$ and $B(x)$. Your algorithm must run in time $o(n^2)$ (where $o(.)$ is little-o), assuming that n is a power of 2.

Hints: Reduce to multiplying polynomials with integer coefficients of degree at most $n/2 - 1$ and use the following fact $(a + b)(c + d) - ac - bd = ad + bc$.

Let $A(x) = A_1(x) + x^{n/2}A_2(x)$ and $B(x) = B_1(x) + x^{n/2}B_2(x)$, where $A_1(x)$, $A_2(x)$, $B_1(x)$, $B_2(x)$ are polynomials of degree at most $n/2 - 1$. Specifically, we can get:

$$A_1(x) = \sum_{i=0}^{\frac{n}{2}-1} a_i x^i \qquad A_2(x) = \sum_{i=\frac{n}{2}}^{n-1} a_i x^i$$

$$B_1(x) = \sum_{i=0}^{\frac{n}{2}-1} b_i x^i \qquad B_2(x) = \sum_{i=\frac{n}{2}}^{n-1} b_i x^i$$

We can further observe that:

$$A(x) \cdot B(x) = (A_1(x) + x^{n/2}A_2(x)) \cdot (B_1(x) + x^{n/2}B_2(x))$$

$$= A_1(x)B_1(x) + (A_1(x)B_2(x) + A_2(x)B_1(x)) \cdot x^{n/2} + A_2(x)B_2(x) \cdot x^n$$

Let $P_1(x) = A_1(x)B_1(x)$, $P_2(x) = A_2(x)B_2(x)$ and $P_3(x) = (A_1(x) + A_2(x)) \cdot (B_1(x) + B_2(x))$. We compute $P_1(x)$, $P_2(x)$ and $P_3(x)$ recursively as each polynomial is the multiplication of two polynomial of degree at most $n/2 - 1$. And then we compute $A_2(x)B_1(x) + A_1(x)B_2(x) = P_3(x) - P_1(x) - P_2(x)$, and finally compute $A(x) \cdot B(x)$ using the previous equation.

The correctness follows directly from the algorithm. We now bound the running time. Let $T(n)$ be the running time to multiply two polynomials of degree at most $n - 1$. Computing $P_1(x)$, $P_2(x)$ and $P_3(x)$ takes $3T(n/2) + O(n)$ time. Given $P_1(x)$, $P_2(x)$ and $P_3(x)$, computing $A(x) \cdot B(x)$ takes $O(n)$ time. Thus, we have $T(n) = 3T(n/2) + O(n)$, which solves to $O(n^{log_2^{(3)}}) \sim O(n^{1.58})$ by Master Theorem.