Divide & Conquer #1 - Polynomial Multiplication

Divide & Conquer Algorithm Form

:

- **Divide** - Split the array or list into smaller pieces
- **Conquer** - Solve the same problem recursively on smaller pieces.
- **Combine** - Build the full solution from the recursive solution.
    - Sometimes these might be really simple, other times this is the core of the algorithm.

Master Theorem (simpler version)

If $T(n) = a \cdot T(\frac{n}{b}) + f(n)$, then the solution is:

- $T(n) = \theta(n^{log_b a})$ if $f(n) = O(n^{log_b a - \epsilon})$ for some constant $\epsilon > 0$.
- $T(n) = \theta(n^{log_b a} \lg n)$ if $f(n) = \theta(n^{log_b a})$.
- $T(n) = \theta(f(n))$ if $f(n) = \Omega(n^{log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ for some constant $c < 1$ and all sufficiently large n.

Divide and Conquer Examples

Merge Sort:

$T(n) = 2 \cdot T(\frac{n}{2}) + \theta(n)$

$a = 2, b = 2, k = 1$

$log_2(2) = 1$

So $T(n) = \theta(n^k \cdot lg(n)) = \theta(n \cdot lg(n))$

Binary Search:

$T(n) = T(\frac{n}{2}) + \theta(1)$

$a = 1, b = 2, k = 0$

$log_2 1 = 0$

So $T(n) = \theta(n^k \cdot lg(n)) = \theta(lg(n))$

There's no combine step on binary search, either we find the element or we don't (hence $\theta(1)$ work.

Polynomial Multiplication problem (divide and conquer approach)

Given two polynomials, how would we multiply them? Here's this in math form:

$P = 5x^3 + 7x^2 + 6x + 2 \Rightarrow [5, 7, 6, 2] = [p_3, p_2, p_1, p_0]$

We're storing these coefficients in an array.

$Q = x^3 - 8x^2 + 9x - 1 \Rightarrow [1, -8, 9, -1] = [q_3, q_2, q_1, q_0]$

$P \cdot Q = (5x^3 + 7x^2 + 6x + 2)(x^3 - 8x^2 + 9x - 1)$

We're multiplying the polynomials

How would write an algorithm for this specific problem? **We split up the problem into smaller problems**.

Let's represent the polynomials in the form $Ax^2 + B$ (where A and B are both polynomials):

$$P = 5x^3 + 7x^2 + 6x + 2 = (5x + 7)x^2 + (6x + 2) = Ax^2 + B$$
$$Q = x^3 - 8x^2 + 9x - 1 = (x - 8)x^2 + (9x - 1) = Cx^2 + D$$

A and B here should be the *same size polynomials*. Here's the results in variable form:

$$A = 5x + 7 \Rightarrow [5, 7]$$
$$B = 6x + 2 \Rightarrow [6, 2]$$
$$C = x - 8 \Rightarrow [1, -8]$$
$$D = 9x - 1 \Rightarrow [9, -1]$$

We're breaking the pieces down into this form (where n is the number of terms in P and Q:

$$P = Ax^{\frac{n}{2}} + B$$
$$Q = Cx^{\frac{n}{2}} + D$$

where A, B, C, D are polynomials with $\frac{n}{2}$ terms.

Now that we have this form, let's rewrite the multiplication of our two polynomials using the variables we have. Here's the result:

$$P \cdot Q = (Ax^{\frac{n}{2}} + B)(Cx^{\frac{n}{2}} + D) = (AC)x^n + (BC + AD)x^{\frac{n}{2}} + (BD)$$

So we can take these pieces we got and plug them into this equation to give the answer.

This now gives us for recursive subproblems. We only need to solve the *smaller polynomial multiplication problems*. We only have to solve these four problems to find our solution:

$$A \cdot C$$
$$B \cdot C$$
$$A \cdot D$$
$$B \cdot D$$

Once we've solved these, we have our answer! We can stop the recursion when $n = 1$ on all of these problems.

▼ **Runtime analysis (with recurrence relation)**

Here's the recurrence relation for our problem:

$$T(n) = 4T(\frac{n}{2}) + \theta(n)$$

> We're splitting the problem into 4 subproblems (those four equations), and we're only doing $\frac{n}{2}$ the amount of work with each of those subproblems. The work we do per problem is $\theta(n)$ because we have to traverse the entire list of each.
>
> Using the master theorem to solve for the runtime, we get:

$$a = 4, b = 2, k = 1$$
$$log_2(4) = 2,\ 2 > 1$$
$$T(n) = \theta(n^{log_b(a)}) = \theta(n^2)$$

So the runtime for this specific algorithm is $\theta(n^2)$. This really isn't any better than the naive approach (how we would do it by hand).

▼ A better polynomial multiplication algorithm (Karatsuba's algorithm)

How do we get our runtime better than $\theta(n^2)$? Let's look at **Karatsuba's algorithm**. It makes a *very simple optimization* to our previous algorithm that reduces the runtime.

Taking the same form for splitting up the problem into smaller pieces:

$$P \cdot Q = (Ax^{\frac{n}{2}} + B)(Cx^{\frac{n}{2}} + D) = (AC)x^n + (BC + AD)x^{\frac{n}{2}} + (BD)$$

Let's rewrite part of the middle term in a different way:

$$(BC + AD) = (A + B)(C + D) - (AC) - (BD)$$

This might seem like a small change, but it gives us only three recursive calls to make now:

$$A \cdot C$$
$$B \cdot D$$
$$(A + B) \cdot (C + D)$$

Let's use the same algorithm we used before, stopping recursion when $n = 1$.

▼ Runtime analysis (with master theorem)

The recurrence relation for this is:

$$T(n) = 3T(\frac{n}{2}) + \theta(n)$$

We only have three problems we're doing. This seems like it would reduce the runtime:

$$a = 3, b = 2, k = 1$$
$$log_2(3) => 1$$
$$T(n) = \theta(n^{log_b(a)}) = \theta(n^{lg(3)}) \approx \theta(n^{1.58})$$

And we were right, the size of our runtime decreased! It still uses the same case of the master theorem.

So the runtime for this is $\theta(n^{lg\ 3})$, or approximately $\theta(n^{1.58})$. It's less than $\theta(n^2)$!