Chapter 23

Fast Fourier Transform

By Sariel Har-Peled, November 28, 2018[1] Version: 0.11

> "But now, reflecting further, there begins to creep into his breast a touch of fellow-feeling for his imitators. For it seems to him now that there are but a handful of stories in the world; and if the young are to be forbidden to prey upon the old then they must sit for ever in silence."
>
> – J.M. Coetzee,

23.1. Introduction

In this chapter, we will address the problem of multiplying two polynomials quickly.

Definition 23.1.1. A ***polynomial*** $p(x)$ of degree n is a function of the form $p(x) = \sum_{j=0}^{n} a_j x^j = a_0 + x(a_1 + x(a_2 + \ldots + x a_n))$.

Note, that given x_0, the polynomial can be evaluated at x_0 in $O(n)$ time.

There is a "dual" (and equivalent) representation of a polynomial. We sample its value in enough points, and store the values of the polynomial at those points. The following theorem states this formally. We omit the proof as you should have seen it already at some earlier math class.

Theorem 23.1.2. *For any set* $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$ *of* n **point-value pairs** *such that all the* x_k *values are distinct, there is a* unique *polynomial* $p(x)$ *of degree* $n-1$*, such that* $y_k = p(x_k)$*, for* $k = 0, \ldots, n-1$.

An explicit formula for $p(x)$ as a function of those point-value pairs is

$$p(x) = \sum_{i=0}^{n-1} y_i \frac{\prod_{j \neq i}(x - x_j)}{\prod_{j \neq i}(x_i - x_j)}.$$

Note, that the ith term in this summation is zero for $X = x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n-1}$, and is equal to y_i for $x = x_i$.

It is easy to verify that given n point-value pairs, we can compute $p(x)$ in $O(n^2)$ time (using the above formula).

The point-value pairs representation has the advantage that we can multiply two polynomials quickly. Indeed, if we have two polynomials p and q of degree $n-1$, both represented by $2n$ (we are using more points than we need) point-value pairs

$$\begin{aligned} &\{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\} \quad \text{for} \quad p(x),\\ \text{and} \quad &\{(x_0, y'_0), (x_1, y'_1), \ldots, (x_{2n-1}, y'_{2n-1})\} \quad \text{for} \quad q(x). \end{aligned}$$

Let $r(x) = p(x)q(x)$ be the product of these two polynomials. Computing $r(x)$ directly requires $O(n^2)$ using the naive algorithm. However, in the point-value representation we have, that the representation of $r(x)$ is

$$\begin{aligned} \left\{(x_0, r(x_0)), \ldots, (x_{2n-1}, r(x_{2n-1}))\right\} &= \left\{(x_0, p(x_0)q(x_0)), \ldots, (x_{2n-1}, p(x_{2n-1})q(x_{2n-1}))\right\} \\ &= \left\{(x_0, y_0 y'_0), \ldots, (x_{2n-1}, y_{2n-1} y'_{2n-1})\right\}. \end{aligned}$$

Namely, once we computed the representation of $p(x)$ and $q(x)$ using point-value pairs, we can multiply the two polynomials in linear time. Furthermore, we can compute the standard representation of $r(x)$ from this representation.

Thus, if could translate quickly (i.e., $O(n \log n)$ time) from the standard representation of a polynomial to point-value pairs representation, and back (to the regular representation) then we could compute the product of two polynomials in $O(n \log n)$ time. The ***Fast Fourier Transform*** is a method for doing exactly this. It is based on the idea of choosing the x_i values carefully and using divide and conquer.

23.2. Computing a polynomial quickly on n values

In the following, we are going to assume that the polynomial we work on has degree $n-1$, where $n = 2^k$. If this is not true, we can pad the polynomial with terms having zero coefficients.

Assume that we magically were able to find a set of numbers $\Psi = \{x_1, \ldots, x_n\}$, so that it has the following property: $|\mathsf{SQ}(\Psi)| = n/2$, where $\mathsf{SQ}(\Psi) = \left\{ x^2 \;\middle|\; x \in \Psi \right\}$. Namely, when we square the numbers of Ψ, we remain with only $n/2$ distinct values, although we started with n values. It is quite easy to find such a set.

What is much harder is to find a set that have this property repeatedly. Namely, $\mathsf{SQ}(\mathsf{SQ}(\Psi))$ would have $n/4$ distinct values, $\mathsf{SQ}(\mathsf{SQ}(\mathsf{SQ}(\Psi)))$ would have $n/8$ values, and $\mathsf{SQ}^i(\Psi)$ would have $n/2^i$ distinct values.

Predictably, maybe, it is easy to show that there is no such set of real numbers (verify...). But let us for the time being ignore this technicality, and fly, for a moment, into the land of fantasy, and assume that we do have such a set of numbers, so that $|\mathsf{SQ}^i(\Psi)| = n/2^i$ numbers, for $i = 0, \ldots, k$. Let us call such a set of numbers ***collapsible***.

Given a set of numbers $\mathcal{X} = \{x_0, \ldots, x_n\}$ and a polynomial $p(x)$, let

$$p(\mathcal{X}) = \left\langle (x_0, p(x_0)), \ldots, \left(x_n, p(x_n)\right) \right\rangle.$$

Furthermore, let us rewrite $p(x) = \sum_{i=0}^{n-1} a_i x^i$ as $p(x) = u(x^2) + x \cdot v(x^2)$, where

$$u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i \qquad \text{and} \qquad v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i.$$

Namely, we put all the even degree terms of $p(x)$ into $u(\cdot)$, and all the odd degree terms into $v(\cdot)$. The maximum degree of the two polynomials $u(y)$ and $v(y)$ is $n/2$.

We are now ready for the kill: To compute $p(\Psi)$ for Ψ, which is a collapsible set, we have to compute $u(\mathsf{SQ}(\Psi)), v(\mathsf{SQ}(\Psi))$. Namely, once we have the value-point pairs of $u(\mathsf{SQ}(A)), v(\mathsf{SQ}(A))$ we can, in *linear* time, compute $p(\Psi)$. But, $\mathsf{SQ}(\Psi)$ have $n/2$ values because we assumed that Ψ is collapsible. Namely,

```
FFTAlg(p, X)
        input: p(x): A polynomial of degree n: p(x) = \sum_{i=0}^{n-1} a_i x^i
                  X: A collapsible set of n elements.
        output: p(X)
begin
        u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i
        v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i.
        Y = SQ(X) = { x^2 | x ∈ X }.
        U = FFTAlg(u,Y)          /* U = u(Y) */
        V = FFTAlg(v, Y)         /* V = v(Y) */

        Out ← ∅
        for x ∈ A do
            /* p(x) = u(x^2) + x * v(x^2) */
            /* U[x^2] is the value u(x^2) */
            (x, p(x)) ← (x, U[x^2] + x · V[x^2])
            Out ← Out ∪ {(x, p(x))}

        return Out
end
```

Figure 23.1: The FFT algorithm.

to compute n point-value pairs of $p(\cdot)$, we have to compute $n/2$ point-value pairs of two polynomials of degree $n/2$ over a set of $n/2$ numbers.

Namely, we reduce a problem of size n into two problems of size $n/2$. The resulting algorithm is depicted in Figure 23.1.

What is the running time of FFTAlg? Well, clearly, all the operations except the recursive calls takes $O(n)$ time (assume, for the time being, that we can fetch $U[x^2]$ in $O(1)$ time). As for the recursion, we call recursively on a polynomial of degree $n/2$ with $n/2$ values (Ψ is collapsible!). Thus, the running time is $T(n) = 2T(n/2) + O(n)$, which is $O(n \log n)$ – exactly what we wanted.

23.2.1. Generating Collapsible Sets

Nice! But how do we resolve this "technicality" of not having collapsible set? It turns out that if we work over the complex numbers (instead of over the real numbers), then generating collapsible sets is quite easy. Describing complex numbers is outside the scope of this writeup, and we assume that you already have encountered them before. Nevertheless a quick reminder is provided in Section 23.4.1. Everything you can do over the real numbers you can do over the complex numbers, and much more (complex numbers are your friend).

In particular, let γ denote a nth root of unity. There are n such roots, and let $\gamma_j(n)$ denote the jth root, see Figure 23.2_{p6}. In particular, let

$$\gamma_j(n) = \cos((2\pi j)/n) + \mathbf{i}\sin((2\pi j)/n) = \gamma^j.$$

Let $\mathcal{A}(n) = \{\gamma_0(n), \ldots, \gamma_{n-1}(n)\}$. It is easy to verify that $|\mathsf{SQ}(\mathcal{A}(n))|$ has exactly $n/2$ elements. In fact, $\mathsf{SQ}(\mathcal{A}(n)) = \mathcal{A}(n/2)$, as can be easily verified. Namely, if we pick n to be a power of 2, then $\mathcal{A}(n)$ is the *required* collapsible set.

Theorem 23.2.1. *Given polynomial $p(x)$ of degree n, where n is a power of two, then we can compute $p(X)$ in $O(n \log n)$ time, where $X = \mathcal{A}(n)$ is the set of n different powers of the nth root of unity over the complex numbers.*

We can now multiply two polynomials quickly by transforming them to the point-value pairs representation over the nth root of unity, but we still have to transform this representation back to the regular representation.

23.3. Recovering the polynomial

This part of the writeup is somewhat more technical. Putting it shortly, we are going to apply the FFTAlg algorithm once again to recover the original polynomial. The details follow.

It turns out that we can interpret the FFT as a matrix multiplication operator. Indeed, if we have $p(x) = \sum_{i=0}^{n-1} a_i x^i$ then evaluating $p(\cdot)$ on $\mathcal{A}(n)$ is equivalent to:

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & \gamma_0 & \gamma_0^2 & \gamma_0^3 & \cdots & \gamma_0^{n-1} \\ 1 & \gamma_1 & \gamma_1^2 & \gamma_1^3 & \cdots & \gamma_1^{n-1} \\ 1 & \gamma_2 & \gamma_2^2 & \gamma_2^3 & \cdots & \gamma_2^{n-1} \\ 1 & \gamma_3 & \gamma_3^2 & \gamma_3^3 & \cdots & \gamma_3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \gamma_{n-1} & \gamma_{n-1}^2 & \gamma_{n-1}^3 & \cdots & \gamma_{n-1}^{n-1} \end{pmatrix}}_{\text{the matrix } V} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix},$$

where $\gamma_j = \gamma_j(n) = (\gamma_1(n))^j$ is the jth power of the nth root of unity, and $y_j = p(\gamma_j)$.

This matrix V is very interesting, and is called the ***Vandermonde*** matrix. Let V^{-1} be the inverse matrix of this Vandermonde matrix. And let multiply the above formula from the left. We get:

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = V^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

Namely, we can recover the polynomial $p(x)$ from the point-value pairs

$$\left\{ (\gamma_0, p(\gamma_0)), (\gamma_1, p(\gamma_1)), \ldots, (\gamma_{n-1}, p(\gamma_{n-1})) \right\}$$

by doing a single matrix multiplication of V^{-1} by the vector $[y_0, y_1, \ldots, y_{n-1}]$. However, multiplying a vector with n entries with a matrix of size $n \times n$ takes $O(n^2)$ time. Thus, we had not benefit ted anything so far.

However, since the Vandermonde matrix is so well behaved[2], it is not too hard to figure out the inverse matrix.

[2]Not to mention famous, beautiful and well known – in short a celebrity matrix.

Claim 23.3.1.

$$V^{-1} = \frac{1}{n}\begin{pmatrix} 1 & \beta_0 & \beta_0^2 & \beta_0^3 & \cdots & \beta_0^{n-1} \\ 1 & \beta_1 & \beta_1^2 & \beta_1^3 & \cdots & \beta_1^{n-1} \\ 1 & \beta_2 & \beta_2^2 & \beta_2^3 & \cdots & \beta_2^{n-1} \\ 1 & \beta_3 & \beta_3^2 & \beta_3^3 & \cdots & \beta_3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \beta_{n-1} & \beta_{n-1}^2 & \beta_{n-1}^3 & \cdots & \beta_{n-1}^{n-1} \end{pmatrix},$$

where $\beta_j = (\gamma_j(n))^{-1}$.

Proof: Consider the (u, v) entry in the matrix $C = V^{-1}V$. We have

$$C_{u,v} = \sum_{j=0}^{n-1} \frac{(\beta_u)^j(\gamma_j)^v}{n}.$$

We need to use the fact here that $\gamma_j = (\gamma_1)^j$ as can be easily verified. Thus,

$$C_{u,v} = \sum_{j=0}^{n-1} \frac{(\beta_u)^j((\gamma_1)^j)^v}{n} = \sum_{j=0}^{n-1} \frac{(\beta_u)^j((\gamma_1)^v)^j}{n} = \sum_{j=0}^{n-1} \frac{(\beta_u\gamma_v)^j}{n}.$$

Clearly, if $u = v$ then

$$C_{u,u} = \frac{1}{n}\sum_{j=0}^{n-1}(\beta_u\gamma_u)^j = \frac{1}{n}\sum_{j=0}^{n-1}(1)^j = \frac{n}{n} = 1.$$

If $u \neq v$ then,

$$\beta_u\gamma_v = (\gamma_u)^{-1}\gamma_v = (\gamma_1)^{-u}\gamma_1^v = (\gamma_1)^{v-u} = \gamma_{v-u}.$$

And

$$C_{u,v} = \frac{1}{n}\sum_{j=0}^{n-1}(\gamma_{v-u})^j = \frac{1}{n}\cdot\frac{\gamma_{v-u}^n - 1}{\gamma_{v-u} - 1} = \frac{1}{n}\cdot\frac{1-1}{\gamma_{v-u} - 1} = 0,$$

this follows by the formula for the sum of a geometric series, and as γ_{v-u} is an nth root of unity, and as such if we raise it to power n we get 1.

We just proved that the matrix C have ones on the diagonal and zero everywhere else. Namely, it is the identity matrix, establishing our claim that the given matrix is indeed the inverse matrix to the Vandermonde matrix. ■

Let us recap, given n point-value pairs $\{(\gamma_0, y_0), \ldots, (\gamma_{n-1}, y_{n-1})\}$ of a polynomial $p(x) = \sum_{i=0}^{n-1} a_i x^i$ over the set of nth roots of unity, then we can recover the coefficients of the polynomial by multiplying the vector $[y_0, y_1, \ldots, y_n]$ by the matrix V^{-1}. Namely,

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \underbrace{\frac{1}{n}\begin{pmatrix} 1 & \beta_0 & \beta_0^2 & \beta_0^3 & \cdots & \beta_0^{n-1} \\ 1 & \beta_1 & \beta_1^2 & \beta_1^3 & \cdots & \beta_1^{n-1} \\ 1 & \beta_2 & \beta_2^2 & \beta_2^3 & \cdots & \beta_2^{n-1} \\ 1 & \beta_3 & \beta_3^2 & \beta_3^3 & \cdots & \beta_3^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & \beta_{n-1} & \beta_{n-1}^2 & \beta_{n-1}^3 & \cdots & \beta_{n-1}^{n-1} \end{pmatrix}}_{V^{-1}} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

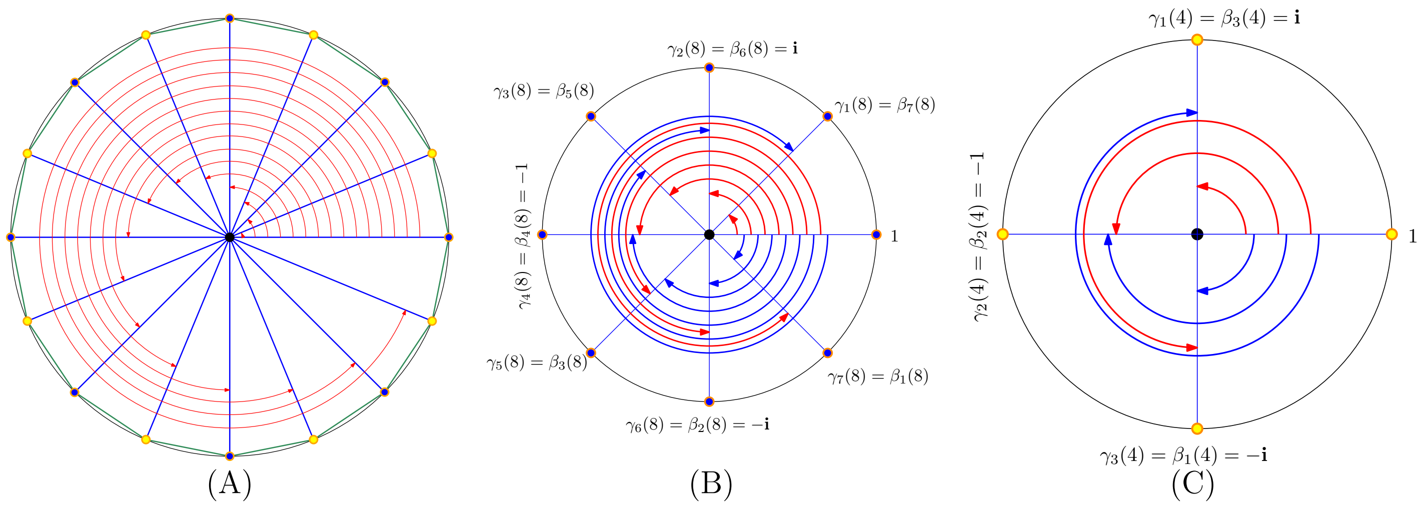

Figure 23.2: (A) The 16 roots of unity. (B) The 8 roots of unity. (C) The 4 roots of unity.

Let us write a polynomial $W(x) = \sum_{i=0}^{n-1} (y_i/n)x^i$. It is clear that $a_i = W(\beta_i)$. That is to recover the coefficients of $p(\cdot)$, we have to compute a polynomial $W(\cdot)$ on n values: $\beta_0, \ldots, \beta_{n-1}$.

The final stroke, is to observe that $\{\beta_0, \ldots, \beta_{n-1}\} = \{\gamma_0, \ldots, \gamma_{n-1}\}$; indeed $\beta_i^n = (\gamma_i^{-1})^n = (\gamma_i^n)^{-1} = 1^{-1} = 1$. Namely, we can apply the FFTAlg algorithm on $W(x)$ to compute $a_0, \ldots, a_{n-1}$.

We conclude:

Theorem 23.3.2. *Given n point-value pairs of a polynomial $p(x)$ of degree $n-1$ over the set of n powers of the nth roots of unity, we can recover the polynomial $p(x)$ in $O(n \log n)$ time.*

Theorem 23.3.3. *Given two polynomials of degree n, they can be multiplied in $O(n \log n)$ time.*

23.4. The Convolution Theorem

Given two vectors: $A = [a_0, a_1, \ldots, a_n]$ and $B = [b_0, \ldots, b_n]$, their dot product is the quantity

$$A \cdot B = \langle A, B \rangle = \sum_{i=0}^{n} a_i b_i.$$

Let A_r denote the shifting of A by $n - r$ locations to the left (we pad it with zeros; namely, $a_j = 0$ for $j \notin \{0, \ldots, n\}$).

$$A_r = \left[a_{n-r}, a_{n+1-r}, a_{n+2-r}, \ldots, a_{2n-r}\right]$$

where $a_j = 0$ if $j \notin \left[0, \ldots, n\right]$.

Observation 23.4.1. *$A_n = A$.*

Example 23.4.2. For $A = [3, 7, 9, 15]$, $n = 3$

$A_2 = [7, 9, 15, 0]$,

$A_5 = [0, 0, 3, 7]$.

Definition 23.4.3. Let $c_i = A_i \cdot B = \sum_{j=n-i}^{2n-i} a_j b_{j-n+i}$, for $i = 0, \ldots, 2n$. The vector $[c_0, \ldots, c_{2n}]$ is the *convolution* of A and B.

Question 23.4.4. *How to compute the convolution of two vectors of length n?*

Definition 23.4.5. The resulting vector $[c_0, \ldots, c_{2n}]$ is the ***convolution*** of A and B.

Let $p(x) = \sum_{i=0}^{n} \alpha_i x^i$, and $q(x) = \sum_{i=0}^{n} \beta_i x^i$. The coefficient of x^i in $r(x) = p(x)q(x)$ is

$$d_i = \sum_{j=0}^{i} \alpha_j \beta_{i-j}.$$

On the other hand, we would like to compute $c_i = A_i \cdot B = \sum_{j=n-i}^{2n-i} a_j b_{j-n+i}$, which seems to be a very similar expression. Indeed, setting $\alpha_i = a_i$ and $\beta_l = b_{n-l-1}$ we get what we want.

To understand whats going on, observe that the coefficient of x^2 in the product of the two respective polynomials $p(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ and $q(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$ is the sum of the entries on the anti diagonal in the following matrix, where the entry in the ith row and jth column is $a_i b_j$.

	a_0+	$a_1 x$	$+a_2 x^2$	$+a_3 x^3$
b_0			$a_2 b_0 x^2$	
$+b_1 x$		$a_1 b_1 x^2$		
$+b_2 x^2$	$a_0 b_2 x^2$			
$+b_3 x^3$				

Theorem 23.4.6. *Given two vectors $A = [a_0, a_1, \ldots, a_n]$, $B = [b_0, \ldots, b_n]$ one can compute their convolution in $O(n \log n)$ time.*

Proof: Let $p(x) = \sum_{i=0}^{n} a_{n-i} x^i$ and let $q(x) = \sum_{i=0}^{n} b_i x^i$. Compute $r(x) = p(x)q(x)$ in $O(n \log n)$ time using the convolution theorem. Let $c_0, \ldots, c_{2n}$ be the coefficients of $r(x)$. It is easy to verify, as described above, that $[c_0, \ldots, c_{2n}]$ is the convolution of A and B. ∎

23.4.1. Complex numbers – a quick reminder

A complex number is a pair of real numbers x and y, written as $\tau = x + \mathbf{i}y$, where x is the ***real*** part and y is the ***imaginary*** part. Here $\mathbf{i}$ is of course the root of -1. In ***polar form***, we can write $\tau = r\cos\phi + \mathbf{i}r\sin\phi = r(\cos\phi + \mathbf{i}\sin\phi) = re^{i\phi}$, where $r = \sqrt{x^2 + y^2}$ and $\phi = \arcsin(y/x)$. To see the last part, define the following functions by their Taylor expansion

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots,$$
$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots,$$
$$\text{and} \qquad e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots.$$

Since $\mathbf{i}^2 = -1$, we have that

$$e^{\mathbf{i}x} = 1 + \mathbf{i}\frac{x}{1!} - \frac{x^2}{2!} - \mathbf{i}\frac{x^3}{3!} + \frac{x^4}{4!} + \mathbf{i}\frac{x^5}{5!} - \frac{x^6}{6!} \cdots = \cos x + \mathbf{i}\sin x.$$

The nice thing about polar form, is that given two complex numbers $\tau = re^{\mathbf{i}\phi}$ and $\tau' = r'e^{\mathbf{i}\phi'}$, multiplying them is now straightforward. Indeed, $\tau \cdot \tau' = re^{\mathbf{i}\phi} \cdot r'e^{\mathbf{i}\phi'} = rr'e^{\mathbf{i}(\phi+\phi')}$. Observe that the function $e^{\mathbf{i}\phi}$ is 2π periodic (i.e., $e^{\mathbf{i}\phi} = e^{\mathbf{i}(\phi+2\pi)}$), and $1 = e^{\mathbf{i}0}$. As such, an nth root of 1, is a complex number $\tau = re^{\mathbf{i}\phi}$ such that $\tau^n = r^n e^{\mathbf{i}n\phi} = e^{\mathbf{i}0}$. Clearly, this implies that $r = 1$, and there must be an integer j, such that

$$n\phi = 0 + 2\pi j \implies \phi = j(2\pi/n).$$

These are all distinct values for $j = 0, \ldots, n-1$, which are the n distinct roots of unity.

Bibliography