The **prime-factor algorithm (PFA)**, also called the **Good–Thomas algorithm** (1958/1963), is a [fast Fourier transform](https://en.wikipedia.org/wiki/Fast_Fourier_transform) (FFT) algorithm that re-expresses the [discrete Fourier transform](https://en.wikipedia.org/wiki/Discrete_Fourier_transform) (DFT) of a size *N* = *N*1*N*2 as a two-dimensional *N*1×*N*2 DFT, but *only* for the case where *N*1 and *N*2 are [relatively prime](https://en.wikipedia.org/wiki/Relatively_prime). These smaller transforms of size *N*1 and *N*2 can then be evaluated by applying PFA [recursively](https://en.wikipedia.org/wiki/Recursion) or by using some other FFT algorithm.

PFA should not be confused with the **mixed-radix** generalization of the popular [Cooley–Tukey algorithm](https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm), which also subdivides a DFT of size *N* = *N*1*N*2 into smaller transforms of size *N*1 and *N*2. The latter algorithm can use *any* factors (not necessarily relatively prime), but it has the disadvantage that it also requires extra multiplications by roots of unity called [twiddle factors](https://en.wikipedia.org/wiki/Twiddle_factor), in addition to the smaller transforms. On the other hand, PFA has the disadvantages that it only works for relatively prime factors (e.g. it is useless for [power-of-two](https://en.wikipedia.org/wiki/Power_of_two) sizes) and that it requires a more complicated re-indexing of the data based on the [Chinese remainder theorem](https://en.wikipedia.org/wiki/Chinese_remainder_theorem) (CRT). Note, however, that PFA can be combined with mixed-radix Cooley–Tukey, with the former factorizing *N* into relatively prime components and the latter handling repeated factors.

PFA is also closely related to the nested [Winograd FFT algorithm](https://en.wikipedia.org/w/index.php?title=Winograd_FFT_algorithm&action=edit&redlink=1), where the latter performs the decomposed *N*1 by *N*2 transform via more sophisticated two-dimensional convolution techniques. Some older papers therefore also call Winograd's algorithm a PFA FFT.

(Although the PFA is distinct from the Cooley–Tukey algorithm, Good's 1958 work on the PFA was cited as inspiration by Cooley and Tukey in their famous 1965 paper, and there was initially some confusion about whether the two algorithms were different. In fact, it was the only prior FFT work cited by them, as they were not then aware of the earlier research by Gauss and others.)

**Algorithm**

Recall that the DFT is defined by the formula:

X k = ∑ n = 0 N − 1 x n e − 2 π i N n k k = 0 , … , N − 1. {\displaystyle X\_{k}=\sum \_{n=0}^{N-1}x\_{n}e^{-{\frac {2\pi i}{N}}nk}\qquad k=0,\dots ,N-1.}

The PFA involves a re-indexing of the input and output arrays, which when substituted into the DFT formula transforms it into two nested DFTs (a two-dimensional DFT).

**Re-indexing**

Suppose that *N* = *N*1*N*2, where *N*1 and *N*2 are relatively prime. In this case, we can define a [bijective](https://en.wikipedia.org/wiki/Bijection) re-indexing of the input *n* and output *k* by:

n = n 1 N 2 + n 2 N 1 mod N , {\displaystyle n=n\_{1}N\_{2}+n\_{2}N\_{1}\mod N,}

k = k 1 N 2 − 1 N 2 + k 2 N 1 − 1 N 1 mod N , {\displaystyle k=k\_{1}N\_{2}^{-1}N\_{2}+k\_{2}N\_{1}^{-1}N\_{1}\mod N,}

where *N*1−1 denotes the [modular multiplicative inverse](https://en.wikipedia.org/wiki/Modular_multiplicative_inverse) of *N*1 [modulo](https://en.wikipedia.org/wiki/Modular_arithmetic) *N*2 and vice versa for *N*2−1; the indices *ka* and *na* run from 0,...,*Na*−1 (for *a* = 1, 2). These inverses only exist for relatively prime *N*1 and *N*2, and that condition is also required for the first mapping to be bijective.

This re-indexing of *n* is called the *Ruritanian* mapping (also *Good's* mapping), while this re-indexing of *k* is called the *CRT* mapping. The latter refers to the fact that *k* is the solution to the Chinese remainder problem *k* = *k*1 mod *N*1 and *k* = *k*2 mod *N*2.

(One could instead use the Ruritanian mapping for the output *k* and the CRT mapping for the input *n*, or various intermediate choices.)

A great deal of research has been devoted to schemes for evaluating this re-indexing efficiently, ideally [in-place](https://en.wikipedia.org/wiki/In-place_algorithm), while minimizing the number of costly modulo (remainder) operations (Chan, 1991, and references).

**DFT re-expression**

The above re-indexing is then substituted into the formula for the DFT, and in particular into the product *nk* in the exponent. Because *e*2*πi* = 1, this exponent is evaluated modulo *N*: any *N*1*N*2 = *N* cross term in the *nk* product can be set to zero. (Similarly, *Xk* and *xn* are implicitly periodic in *N*, so their subscripts are evaluated modulo *N*.) The remaining terms give:

X k 1 N 2 − 1 N 2 + k 2 N 1 − 1 N 1 = ∑ n 1 = 0 N 1 − 1 ( ∑ n 2 = 0 N 2 − 1 x n 1 N 2 + n 2 N 1 e − 2 π i N 2 n 2 k 2 ) e − 2 π i N 1 n 1 k 1 . {\displaystyle X\_{k\_{1}N\_{2}^{-1}N\_{2}+k\_{2}N\_{1}^{-1}N\_{1}}=\sum \_{n\_{1}=0}^{N\_{1}-1}\left(\sum \_{n\_{2}=0}^{N\_{2}-1}x\_{n\_{1}N\_{2}+n\_{2}N\_{1}}e^{-{\frac {2\pi i}{N\_{2}}}n\_{2}k\_{2}}\right)e^{-{\frac {2\pi i}{N\_{1}}}n\_{1}k\_{1}}.}

The inner and outer sums are simply DFTs of size *N*2 and *N*1, respectively.

(Here, we have used the fact that *N*1−1*N*1 is unity when evaluated modulo *N*2 in the inner sum's exponent, and vice versa for the outer sum's exponent.)

Applications of fft

FFT's importance derives from the fact that in signal processing and image processing it has made working in
frequency domain equally computationally feasible as working in temporal or spatial domain. Some of the
important applications of FFT includes,[16][46]
Fast large integer and polynomial multiplication
Efficient matrix-vector multiplication for Toeplitz, circulant and other structured matrices
Filtering algorithms
Fast algorithms for discrete cosine or sine transforms (example, Fast DCT used for JPEG, MP3/MPEG
encoding)
Fast Chebyshev approximation
Fast discrete Hartley transform
Solving difference equations
Fast Fourier transform - Wikipedia https://en.wikipedia.org/wiki/Fast\_Fourier\_transform
9 of 13 06-02-2018, 08:13 PM
Computation of isotopic distributions.[47]