Title-The Polynomial Multiplication

Problem-Given two polynomials- $A(x) = a0+a1x+a2x^2+\dots\ an-1x^n-1$ and $B(x) = b0+b1x+b2x^2+\dots\ bn-1x^n-1$, find the polynomial $C(x) = A(x)*B(x)$.

General Algorithm-If we think of A and B as vectors, then C vector is called '*Convolution*' of A and B (*represented as {A⊗B}*).In this algorithm each coefficient in vector *a* must be multiplied by each coefficient in vector *b* this makes it's time complexity $O(n^2)$.

Objective-To study an algorithm that runs faster than the naive algorithm.

Fast Fourier Transform-

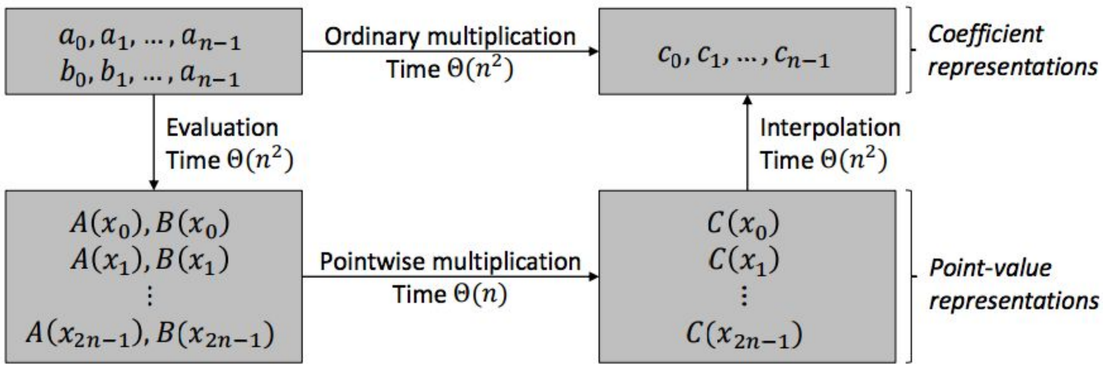
FFT was invented around 1805 by Carl Friedrich Gauss but his work was not widely recognized.FFTs became popular after James Cooley of IBM and John Tukey of Princeton published a paper in 1965 reinventing the algorithm and describing how to perform it conveniently on a computer.Therefore it is also known as Cooley–Tukey FFT algorithm.

The idea is to represent polynomials in point-value form and then compute the product. Using Horner's method evaluation of the polynomial at n-points takes $O(n^2)$.Now that the polynomials are converted in point-value representation evaluation of $C(x)=A(x)*B(x)$ takes $O(n)$ time.

An important point here is that $C(x)$ has degree bound 2n,then n points will give n points of $C(x)$,so for that case we need 2n different values of x to calculate 2n different values of y.This is because of the Fundamental Theorem of Algebra that states-***"A degree n-1 polynomial A(x) is uniquely specified by its evaluation at n distinct values of x".***

Now that we have the product calculated,we can convert it back to coefficient vector form.The inverse evaluation of determining the coefficient form of a polynomial from a point-value representation is called interpolation. Interpolation can be done by using Lagrange's formula in $O(n^2)$.

The road so far-



Now the efficiency of our algorithm depends on the efficiency of conversion between the two representations i.e coefficient representation to point-value representation and visa-versa.This conversion can be done efficiently by choosing the evaluation points carefully.If we choose "complex roots of unity"as the evaluation points,we can produce a point-value representation by taking Discrete Fourier Transform(DFT) of coeff. vector.

Discrete Fourier Transform (DFT)-

DFT is evaluating values of the polynomial at n complex roots of unity $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$ ($\omega_n^k = e^{2\pi i k/n}$ $k=0,1,2,\ldots,n-1$).Here we assume n is power of 2,we can meet this requirement by adding high-order zero coeff.

Fast Fourier Transform (FFT) -

Fast Fourier Transform (FFT) takes advantage of the special properties of the complex roots of unity to compute DFT in time $O(n \log n)$.

Special properties of the complex roots of unity are-

1. Cancellation lemma.
2. Halving lemma.

3. Summation lemma.

Divide-and-conquer strategy-

Define two new polynomials of degree-bound n/2, using even-index and odd-index coefficients of $A(x)$ separately –

A_even =a0+a2x+a4x^2+..+an-2x^n/2-1
A_odd =a1+a3x+a5x^2+..+an-1x^n/2-1
A(x)=A_even(x^2)+xA_odd(x^2)

The problem of evaluating A(x) at ωn^0,ωn^1,...,ωn^n-1 reduces to evaluating the degree-bound n/2 polynomials A_even and A_odd at the points (ωn^0)^2,(ωn^1)^2,...,(ωn^n-1)^2.Polynomials A_even and A_odd are recursively evaluated at the n/2 complex n/2 th roots of unity.Subproblems have exactly the same form as the original problem,but half the size.

So, the recurrence formed is T(n)=2T(n/2)+O(n) i.e O(nlogn).

Interpolation-Just replace ωn with ωn^-1 and divide each element of the result by n.

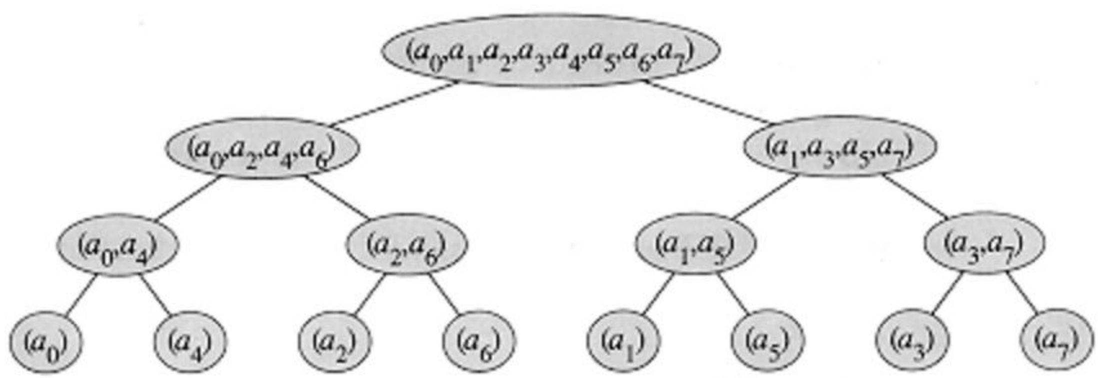
Pseudocode for recursive fft -

Let A be an array of length n, ω be primitive nth root of unity.
Goal: produce DFT F(A): evaluation of A at 1, ω, ω^2,...,ω^{n-1}.

```
FFT(A, n,ω)
{
 if (n==1) return vector (a_0)
 else {
   A_even = (a_0, a_2, ..., a_{n-2})
   A_odd  = (a_1, a_3, ..., a_{n-1})
   F_even = FFT(A_even, n/2, ω^2)     //ω^2 is a n/2-th root of unity
   F_odd = FFT(A_odd, n/2, ω^2)
   F = new vector of length n
```

```
  x = 1
  for (j=0; j < n/2; ++j) {
    F[j] = F_even[j] + x*F_odd[j]
    F[j+n/2] = F_even[j] - x*F_odd[j]
    x = x * ω
 }
 return F
}
```

Here is the tree of input vectors having n=8 to the recursive calls of the FFT procedure.



A popular FFT application area is in the field of signal processing.It is also widely used in image processing to analyze data in 2 or more dimensions.There has been a recent modification of 2 dimensional fft with an analog of the Cooley–Tukey algorithm for image processing.

References-

- J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series", *Math. of Comput.*, vol. 19, pp. 297-301, April 1965.
- T. H. Cormen et al. Introduction to Algorithms. McGraw-Hill, Inc., 2nd edition, 2001.

- Cooley, James W.; Lewis, Peter A. W.; Welch, Peter D. (1967). "Historical notes on the Fast Fourier transform" (PDF). *IEEE Transactions on Audio and Electroacoustics*.
- Fast Fourier Transform retrieved from https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2012/lecture-notes/MIT6_046JS12_lec05.pdf
- Noskov.M & Tulatlchikov V. ,(2017) Modification of a two-dimensional fast Fourier transform algorithm with an analog of the Cooley–Tukey algorithm for image processing.*Institute of Space and Information Technology* . doi 10.1134/S1054661817010096.
- Cooley–Tukey FFT algorithm wikipedia page- **https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm**