Key recurrences for divide-and-conquer algorithms:

$$T(n) = T\left(\frac{n}{2}\right) + O(1) = O(\log n) \quad \text{(binary search)}$$

$$T(n) = T\left(\frac{n}{2}\right) + O(n) = O(n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1) = O(n)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n\log n) \quad \text{(mergeSort)}$$

From the desired recurrence we get a high level idea of the algorithm.

Divide & conquer approach:

1) Break into subproblems of the same type
   Typically problems of half the size.
2) Recursively solve these subproblems
3) Combine/merge solutions to subproblems to get solution to whole problem.

Multiplying Polynomials:

Thursday March 12, 2015

Example: 2 polynomials of degree $d=2$:

$$A(x) = 1 + 2x + 3x^2 = a_0 + a_1 x + a_2 x^2$$
$$B(x) = 2 - x + 4x^2 = b_0 + b_1 x + b_2 x^2$$

Goal: compute their product:

$$C(x) = A(x)B(x) = (1+2x+3x^2)(2-x+4x^2)$$
$$= 2 + 3x + 8x^2 + 5x^3 + 12x^4$$
$$= c_0 + c_1 x + c_2 x^2 + c_3 x^3 + c_4 x^4$$

$c_0 = a_0 b_0$, $c_1 = a_0 b_1 + a_1 b_0, \ldots$

In general, given the coefficients:

$a = (a_0, a_1, \ldots, a_d)$ & $b = (b_0, b_1, \ldots, b_d)$

for polynomials $A(x) = \sum_{i=0}^{d} a_i x^i$ & $B(x) = \sum_{i=0}^{d} b_i x^i$

we want to compute the coefficients

$c = (c_0, c_1, \ldots, c_{2d})$ for $C(x) = \sum_{i=0}^{2d} c_i x^i = A(x)B(x)$,

where

$$c_k = a_0 b_k + a_1 b_{k-1} + \cdots + a_k b_0$$
$$= \sum_{i=0}^{k} a_i b_{k-i}$$

(α)

Vector c is called the <u>convolution</u> of vectors a & b.

Naive approach: $O(k)$ time for c_k & then $O(d^2)$ total time.

Using FFT: $O(d \log d)$ total time.

Two ways to represent a polynomial $A(x) = a_0 + a_1 x + \dots + a_d x^d$

1) coefficients: $a_0, a_1, \dots, a_d$

or 2) values: $A(x_0), A(x_1), \dots, A(x_d)$

<u>Lemma:</u> A degree d polynomial is uniquely characterized by its values at any $d+1$ distinct points

(example: line has $d=1$ & is defined by 2 points)

We assume input/output is in coefficients representation but the values representation is more useful for multiplying polynomials:

Given $A(x_0), \dots, A(x_{2d})$ & $B(x_0), \dots, B(x_{2d})$

then $C(x_i) = A(x_i) B(x_i)$

takes $O(1)$ time per i, $O(d)$ choices of i

$\Rightarrow O(d)$ total time.

& $C(x_0), \dots, C(x_{2d})$ defines $C(x)$.

FFT: converts between coefficients $\Leftrightarrow$ values
in $O(\ell \log \ell)$ time
(does it for carefully chosen set of points)

Consider polynomial $A(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$
where n is a power of 2

Given $a = (a_0, a_1, \ldots, a_{n-1})$
we want to output $A(x_1), \ldots, A(x_{2n})$
for $2n$ points that we choose.
How should we choose these points?

<u>Key idea:</u> Suppose we have n points $x_1, \ldots, x_n$
& the other n points are $x_{n+1} = -x_1, \ldots, x_{2n} = -x_n$
So the $2n$ points are $\pm x_1, \pm x_2, \ldots, \pm x_n$
Note $A(x_i)$ & $A(-x_i)$ are the same for the even terms
and opposite for odd terms

So let's split $A(x)$ into
even and odd terms.

For $A(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \cdots + a_{n-1} x^{n-1}$

let $A_{even}(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{n-2} y^{(n-2)/2}$

So $a_{even} = (a_0, a_2, a_4, \ldots, a_{n-2})$

and $A_{odd}(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{n-1} y^{(n-2)/2}$

So $a_{odd} = (a_1, a_3, a_5, \ldots, a_{n-1})$

Notice that: $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$

Hence, $A(x_i) = A_{even}(x_i^2) + x_i A_{odd}(x_i^2)$

and $A(x_{n+i}) = A(-x_i) = A_{even}(x_i^2) - x_i A_{odd}(x_i^2)$

So given $A_{even}(y_1), \ldots, A_{even}(y_n)$ & $A_{odd}(y_1), \ldots, A_{odd}(y_n)$

for $y_1 = x_1^2, \ldots, y_n = x_n^2$

then we get in $O(n)$ time

$$\begin{array}{cccc} A(x_1), \ldots, A(x_n), & A(x_{n+1}), & \ldots, & A(x_{2n}) \\ & \| & & \| \\ & A(-x_1) & & A(-x_n) \end{array}$$

$A_{even}(y)$ & $A_{odd}(y)$ are of degree $\frac{n-2}{2} = \frac{n}{2} - 1$

whereas $A(x)$ has degree $n-1$.

So to solve the problem of evaluating $A(x)$ of degree $n-1$ at $2n$ points

we need to know $A_{even}(y)$ and $A_{odd}(y)$ which are of degree $\frac{n}{2}-1$ at n points

$\Rightarrow$ Divide & conquer

to solve the problem for size n need to solve 2 subproblems of size $\frac{n}{2}$

$$\Rightarrow T(n) = 2T\left(\frac{n}{2}\right) + O(n) = O(n \log n).$$

What about the next round?

Need that $x_1^2, x_2^2, \ldots, x_n^2$ are $\pm$ pairs so:

$$x_1^2 = -\left(x_{\frac{n}{2}+1}^2\right)$$

$$x_2^2 = -\left(x_{\frac{n}{2}+2}^2\right)$$

$$\vdots$$

$$x_{\frac{n}{2}}^2 = -\left(x_n^2\right)$$

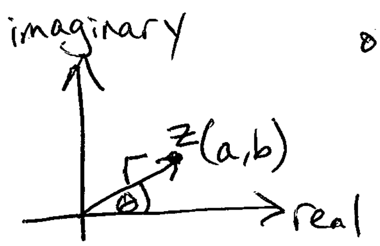
but these are both ≥ 0

So it's impossible unless ~~we~~ use complex numbers.

Review of complex numbers:

$a+bi$ represented as (a,b) in the complex plane

or (r,θ) in polar coordinates



for polar (r,θ): $\quad z = r(\cos\theta + i\sin\theta) = re^{i\theta}$

↑ Euler's formula

Polar is convenient:

Multiplying: $(r_1,\theta_1)\times(r_2,\theta_2) = (r_1 r_2, \theta_1+\theta_2)$

So if $r=1$, for $z=(1,\theta)$

then $z^n = (1, n\theta)$

Since $-1 = (1,\pi)$

then if $z=(r,\theta)$ then $-z = (r, \theta+\pi)$

n^{th} complex roots of unity are those z

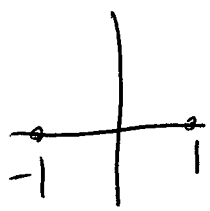
where $z^n = 1$

Thus, they are $z=(1,\theta)$ where $z^n = (1, 2\pi)$

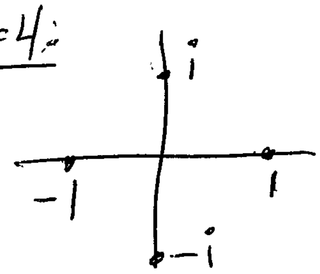
thus $\theta = \frac{2\pi}{n} j$ where $j = 0, 1, \ldots, n-1$.

Let's look at the n^{th} complex roots of unity – for n which is a power of 2 so $n = 2^k$ for some k.

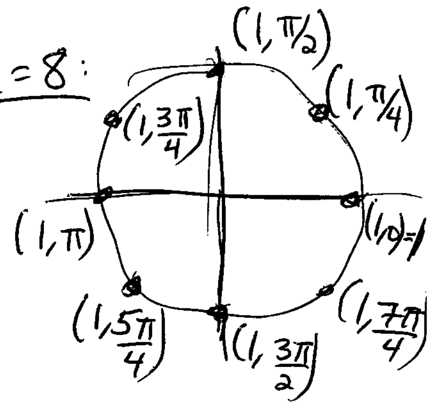
$n=2$:



$n=4$:



$n=8$:



$$\text{Let } \omega = \left(1, \frac{2\pi}{n}\right) = e^{2\pi i/n}$$

Then the n^{th} roots of unity are

$$1, \omega, \omega^2, \ldots, \omega^{n-1}$$

<u>Note:</u> $\omega^j = \left(1, \frac{2\pi}{n} j\right) = e^{2\pi i j/n}$

for $j = 0, 1, 2, \ldots, n-1$

<u>Two key Properties:</u>

1) Satisfy $\pm$ property

$$\omega^j = -\omega^{\frac{n}{2}+j} \quad \text{for } j = 0, 1, \ldots, \tfrac{n}{2}-1$$

So the 1^{st} $\frac{n}{2}$ of the n^{th} roots $=$ $-$ last $\frac{n}{2}$ of the n^{th} roots

$$1 = -\omega^{\frac{n}{2}}$$
$$\omega = -\omega^{\frac{n}{2}+1}$$
$$\vdots$$
$$\omega^{\frac{n}{2}-1} = -\omega^{n-1}$$

2) Look at the square of the n^{th} roots:

$$(1)^2, (\omega)^2, (\omega^2)^2, (\omega^3)^2, \ldots, (\omega^{n-1})^2$$

$$(\omega^j)^2 = \left(1, \frac{2\pi j}{n}\right) \times \left(1, \frac{2\pi j}{n}\right) = \left(1, \frac{2\pi j}{n/2}\right) = j^{th} \text{ of the } \tfrac{n}{2} \text{ root}$$

$$\& \quad (\omega^{\frac{n}{2}+j})^2 = (-\omega^j)^2 = (\omega^j)^2$$

So if we square the n^{th} roots we get the $\left(\frac{n}{2}\right)^{nd}$ roots

If we want to evaluate a polynomial

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

at the n^{th} roots of unity

then let $A_{even}(y) = a_0 + a_2 y + a_4 y^2 + \cdots + a_{n-2} y^{(n-2)/2}$

$A_{odd}(y) = a_1 + a_3 y + a_5 y^2 + \cdots + a_{n-1} y^{(n-2)/2}$

Note: $A(x) = A_{even}(x^2) + x A_{odd}(x^2)$

Degree of $A(x)$ is $n-1$ & degrees of $A_{even}(y)$ & $A_{odd}(y)$ is $\frac{n}{2}-1$

To get $A(x)$ at n^{th} roots, need $A_{even}(y)$ & $A_{odd}(y)$ at $\frac{n}{2}$ th roots

So 2 subproblems of half the size

FFT: takes coefficients of poly $A(x)$ & outputs the value of $A(x)$ at the n^{th} roots.

FFT algorithm:

Let $\omega = e^{2\pi i/n}$

FFT(a, ω):

<u>input</u>: vector $a = (a_0, a_1, \ldots, a_{n-1})$ which are coefficients for polynomial $A(x)$ of degree $\leq n-1$ where n is a power of 2, & ω is a $n^{\underline{th}}$ root of unity

<u>output</u>: $A(\omega^0), A(\omega), A(\omega^2), \ldots, A(\omega^{n-1})$

if $n=1$, return$(A(1))$

Let $a_{even} = (a_0, a_2, \ldots, a_{n-2})$ & $a_{odd} = (a_1, a_3, \ldots, a_{n-1})$

$(s_0, s_1, \ldots, s_{\frac{n}{2}-1}) = \text{FFT}(a_{even}, \omega^2)$

$(t_0, t_1, \ldots, t_{\frac{n}{2}-1}) = \text{FFT}(a_{odd}, \omega^2)$

For $j = 0 \rightarrow \frac{n}{2}-1$:

$$r_j = s_j + \omega^j t_j$$

$$r_{\frac{n}{2}+j} = s_j - \omega^j t_j$$

Return$(r_0, r_1, \ldots, r_{n-1})$

Running time: $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$
$= O(n \log n)$

Original problem:

Given $a=(a_0, a_1, \dots, a_{n-1})$ & $b=(b_0, b_1, \dots, b_{n-1})$ which are coefficients for poly $A(x)$ & $B(x)$,

Compute coefficients $c=(c_0, \dots, c_{2n-2})$ for $C(x)=A(x)B(x)$.

So we run $FFT(a,\omega)$ & $FFT(b,\omega)$

for $\omega = e^{2\pi i/2n} = 2n^{th}$ root of unity

to get $A(x)$ & $B(x)$ at $x=\omega^j$ for $j=0,1,\dots,2n-1$

then $C(\omega^j) = A(\omega^j)\,B(\omega^j)$

So we have $C(x)$ at $2n$ points

But then we need to interpolate to get the coefficients for $C(x)$.

To do that we do a reverse FFT, which is just another FFT.

For points $x_0, x_1, \ldots, x_{n-1}$ notice that:

$$\begin{bmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ & & \vdots & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

for FFT we have $x_j = \omega^j$ so:

$$\underbrace{\begin{bmatrix} A(1) \\ A(\omega) \\ A(\omega^2) \\ \vdots \\ A(\omega^{n-1}) \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(n-1)} \\ & & \vdots & & \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \cdots & \omega^{(n-1)(n-1)} \end{bmatrix}}_{M_n(\omega)} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}}_{a}$$

So: $A = M_n(\omega) a$

Multiplying a by $M_n(\omega)$ gives A
But we want to go from A to a
So we need $M_n(\omega)^{-1}$

<u>Lemma</u>: $M_n(\omega)^{-1} = \frac{1}{n} M_n(\omega^{-1})$

What's ω^{-1}? $\omega^{-1} = \omega^{n-1}$ because:

$$\omega^{n-1} * \omega = \omega^n = 1$$

Then to compute $M_n(\omega)^{-1} A$ we just run $FFT(A, \omega^{n-1})$ this gives:

$$M_n(\omega^{-1}) A = M_n(\omega)^{-1} A$$ which is what we want.