Polynomials

1. Definitions and Operations.

A polynomial $p(x)$ of degree $n-1$ is a function in x s.t. there are numbers $p_0, p_1, \dots, p_{n-1}$ $(p_{n-1} \neq 0)$ with

$$p(x) = p_0 + p_1 x + p_2 x^2 + \dots + p_{n-1} x^{n-1}$$

The p_i's are typically real or complex numbers.
If we define $p_j = 0$ for $j > n-1$, then

$$p(x) = \sum_{i=0}^{\infty} p_i x^i.$$

Ex. $p(x) = 1 + x^2 - 4x^3$, $n=4$, $p_0=1$, $p_1=0$, $p_2=1$, $p_3=-4$

$q(x) = 1 - x^3 + 7 + 5x^3$, $n=4$, $q_0=8$, $q_1=q_2=0$, $q_3=4$.

Typical oprn. for $p(x) = \sum_i p_i x^i$ and $q(x) = \sum_i q_i x^i$:

Addition $p(x) + q(x) = \sum_i (p_i + q_i) x^i$

Multiplication $p(x)\,q(x) = \sum_i \left(\sum_{j=0}^{i} p_j q_{i-j} \right) x^i$

Evaluation given x_0, compute $y_0 = p(x_0)$.

Interpolation given pairs $(x_0, y_0), \dots, (x_{n-1}, y_{n-1})$, $x_i \neq x_j$ for $i \neq j$
find $p(x)$ of deg. $n-1$ or less $p(x_i) = y_i$, $0 \le i \le n-1$

A common representation uses an array for the coefficients.

```
type Polynomial = array [0,...,max] of real;
```

For a polynomial p, let $n(p)$ exceed its degree. We call $n(p)$ the degree bound for p.

- Easy to check that 2 polynomials can be added in ~~linear~~ time linear in the sum of two degree bounds.

Simple Multiplication.

We just follow the definition.

$$r(x) = p(x)q(x) = \sum_i \left(\sum_{j=0}^{i} p_j q_{i-j} \right) x^i$$

To avoid complications, we fill the upper ends of p&q

```
for i := n(p) to n(p)+n(q)-2 do p[i] := 0 endfor;
for i := n(q) to n(p)+n(q)-2 do q[i] := 0 endfor;
for i := 0 to n(p)+n(q)-2 do
        r[i] := 0;
        for j := 0 to i do r[i] := r[i] + p[j] * q[i-j]
endfor
```

This takes $O(n^2)$ time with $n = n(p)+n(q)$.

It may appear that the polynomials cannot multiplied in much less than quadratic time, this is not true.

Multiplying with divide and conquer.

There is a way to multiply 2 degree 1 polynomials using 3 instead of 4 real multiplications.

$$(p_0 + p_1 x)(q_0 + q_1 x) = A + Bx + Cx^2 \text{ where}$$

$$A = p_0 q_0$$

$$B = (p_0 + p_1)(q_0 + q_1) - A - C \quad [= p_0 q_1 + p_1 q_0]$$

$$C = p_1 q_1$$

This method can be used for general degree polynomials if we partition them into 2 pieces, the pieces are multiplied recursively.

$$p(x) = p_0 + p_1 x + \cdots + p_{n-1} x^{n-1}$$
$$q(x) = q_0 + q_1 x + \cdots + q_{n-1} x^{n-1} \qquad \text{assume } n = 2^k$$

Define

$$a(x) = p_0 + p_1 x + \cdots + p_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$
$$b(x) = p_{\frac{n}{2}} + p_{\frac{n}{2}+1} x + \cdots + p_{n-1} x^{\frac{n}{2}-1}$$
$$c(x) = q_0 + q_1 x + \cdots + q_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$
$$d(x) = q_{\frac{n}{2}} + q_{\frac{n}{2}+1} x + \cdots + q_{n-1} x^{\frac{n}{2}-1}$$

$$p(x) = a(x) + b(x) x^{n/2}, \quad q(x) = c(x) + d(x) \cdot x^{n/2}$$

$$r(x) = \underline{A(x)} + \underline{B(x)} x^{\frac{n}{2}} + \underline{C(x)} x^n \quad \text{where}$$

$$A(x) = a(x)\, c(x)$$
$$B(x) = (a(x) + b(x))(c(x) + d(x)) - A(x) - C(x)$$
$$C(x) = b(x) \cdot d(x)$$

The additions/subtractions are more here than the original algorithm. But, saving in in the number of recursive calls.

So, the algorithm is:

1. copy appropriate parts of $p(x)$ and $q(x)$ to get $a(x), b(x), c(x), d(x)$
   and $e(x) = a(x) + b(x)$, $f(x) = c(x) + d(x)$

2. Recursively compute $A(x) = a(x)\,c(x)$
   $C(x) = b(x)\,d(x)$
   $B(x) = e(x)\,f(x) - A(x) - C(x)$

3. Return $A(x) + B(x)\,x^{n/2} + C(x)\,x^{n}$

The time complexity is

$$T(n) = O(n) + 3T\left(\frac{n}{2}\right) = O\left(n^{\log_2 3}\right)$$
$$= O(n^{1.58})$$

Using an entirely different method called Fast Fourier transform, it is possible to multiply two polynomials even faster.

1. Point-value Representation

A polynomial p with degree-bound n can be represented by a set of n point-value pairs $\{(x_0, y_0), (x_1, y_1), \cdots, (x_{n-1}, y_{n-1})\}$ where $y_i = p(x_i)$ and $x_i \neq x_j$ if $i \neq j$.

Claim For any set $\{(x_i, y_i) \mid 0 \leq i < n-1 \text{ and } x_i \neq x_j \text{ if } i \neq j\}$ there is a unique polynomial of degree $n-1$ or less s.t. $y_i = p(x_i)$ $\forall i$.

proof. The requirement that $y_i = p(x_i)$ for $0 \leq i \leq n-1$ can be written as matrix-vector multiplicati

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Vandermode matrix V

This is a system of n linear equations and n unknowns (the p_is). It has a unique solution if the determinant of the matrix is not zero. Indeed,

$$\det V = \prod_{j<k} (x_k - x_j) \neq 0$$ because all x_i are pairwise different.

Thus,

$$\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{bmatrix} = V^{-1} \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

Evaluation and interpolation can thus be viewed as transforms between the 2 representations, by coefficients and point-value pairs.

Evaluations.

The most common algorithm for evaluation uses Horner's rule.

$$p(x) = p_0 + x(p_1 + x(p_2 + \cdots + x(p_{n-2} + x(p_{n-1}))\cdots))$$

Given a point x_0, y_0 can be computed as:

```
y0 := 0; for i := n-1 downto 0 do
            y0 := y0 * x0 + p[i]
         endfor
```

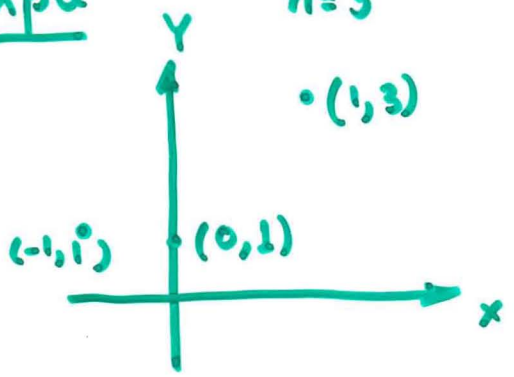
So, we get a point-value pairs representation in $O(n^2)$ time by evaluating p for n different x_i.

Interpolation.

The most common method for interpolation is based on Lagrange's formula:

$$P(x) = \sum_{k=0}^{n-1} y_k \left(\prod_{\substack{j=0 \\ j \neq k}}^{n-1} \frac{x - x_j}{x_k - x_j} \right)$$

Example $n = 3$



$$P(x) = y_0 \frac{x-x_1}{x_0-x_1} \cdot \frac{x-x_2}{x_0-x_2} + y_1 \frac{x-x_0}{x_1-x_0} \cdot \frac{x-x_2}{x_1-x_2} + y_2 \frac{x-x_0}{x_2-x_0} \cdot \frac{x-x_1}{x_2-x_1}$$

$$= 1 \cdot \frac{x}{-1} \cdot \frac{x-1}{-2} + 1 \cdot \frac{x+1}{1} \cdot \frac{x-1}{-1} + 3 \frac{x+1}{2} \cdot \frac{x}{1}$$

$$= \frac{1}{2}(x^2 - x) - (x^2 - 1) + \frac{3}{2}(x^2 + x)$$

$$= x^2 + x + 1$$

Lagrange's formula can be computed in $O(n^2)$ time if we make use of the similarity between terms.

$$a_0(x) = (x-x_1)(x-x_2)(x-x_3)\cdots(x-x_{n-1})$$

$$a_1(x) = (x-x_0)(x-x_2)(x-x_3)\cdots(x-x_{n-1})$$

$$a_2(x) = (x-x_0)(x-x_1)(x-x_3)\cdots(x-x_{n-1})$$

$$\vdots$$

If $a(x) = \prod_{k=0}^{n-1} (x - x_k)$, then $P(x) = \sum_{k=0}^{n-1} y_k \frac{a(x)}{x - x_k} \Big/ \left(\prod_{j \neq k} (x_j - x_k) \right)$

The $O(n^2)$ time bound follows if we can compute $\frac{a(x)}{x - x_k}$ in $O(n)$ time.