School of Computer Science and En
University of New South Wales

4. FAST LARGE INTEGER MULTIPLICATION - part A

- The primary school algorithm:


```
                X X X X   <- first input integer



            X X X X       / O(n^2) elementary  multiplications
          X X X X       /    + O(n^2) elementary additions
      ---------------
      X X X X X X X X   <-  result of length 2n
```


- **Can we do it faster than in n^2 many steps??**

- Take the two input numbers A and B, and split them into two halves:

$$A = A_1 2^{\frac{n}{2}} + A_0 \qquad A = \overbrace{\underbrace{XX}\ldots\underbrace{X}}^{A_1}\ \overbrace{\underbrace{XX}\ldots\underbrace{X}}^{A_0}$$

- AB

$$AB = A_1B_1 2^n + (A_1B_0 + A_0B_1)2^{\frac{n}{2}}$$

$$= A_1B_1 2^n + ((A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0)2^{\frac{n}{2}} + A_0B_0$$

- We have saved one multiplication, now we have only three: A_0B_0, A_1B_1 and $(A_1 + A_0)(B_1 + B_0)$.

$AB = A_1B_12^n + ((A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0)2^{\frac{n}{2}} + A_0B_0$


```
function MULT(A, B)
    if |A| = |B| = 1 then return AB
    else
        U ← A_0 + A_1;
        V ← B_0 + B_1;
        X ← MULT(A_0, B_0);
        W ← MULT(A_1, B_1);
        Y ← MULT(U, V);
        return W 2^n + (Y − X − W) 2^{n/2} + X
    end if
end function
```

- How many steps does this algorithm take? (remember, addition is in linear time!)
- Recurrence: $$T(n) = 3T\left(\frac{n}{2}\right) + c\,n$$

$\log_2 3$

- sinc 1

$$f(n) = c\,n = O(n^{\log_2 3-\varepsilon}) \quad \text{for}$$

- Thus, the first case of the Master Theorem applies.
- Consequently, $$T(n) = \Theta(n^{\log_2 3}) < \Theta(n^{1.585})$$ without going through the messy calculations!

Generalizing Karatsuba's algorithm

- Can we do better if we break the numbers in more than two pieces?
- Lets try breaking the numbers A, B into 3 pieces; then with $k = n/3$ we obtain

$$A = XXX\dots XX\ XXX\dots XX\ XXX\dots XX$$

0

i.e.,

2 1 0

$$B = B_2\, 2^{2k} +$$

- So,

$$AB = A_2B_2\, 2^{4k} + (A_2B_1 + A_1B_2)2^{3k} + (A_2B_0 + A_1B_1 + A_0B_2)2^{2k} + \\ + (A_1B_0 + A_0B_1)2^{k} + A_0B_0$$

$$AB = \underbrace{A_2B_2}_{C_4}\, 2^{4k} + \underbrace{(A_2B_1 + A_1B_2)}_{C_3}2^{3k} + \underbrace{(A_2B_0 + A_1B_1 + A_0B_2)}_{C_2}2^{2k} +$$

$$+ \underbrace{(A_1B_0 + A_0B_1)}_{C_1}2^k + \underbrace{A_0B_0}_{C_0}$$

- we ne

$$C_2 = A_2B_0 + A_1B_1 + A_0B_2$$
$$C_1 = A_1B_0 + A_0$$
$$C_0 = A_0B_0$$

- Can we get these with 5 multiplications only?
- Should we perhaps look at

$$(A_2 + A_1 + A_0)(B_2 + B_1 + B_0) =$$
$$A_0B_0 + A_1B_0 + A_2B_0 + A_0B_1 + A_1B_1 + A_2B_1 + A_0B_2 + A_1B_2 + A_2B_2 \quad ???$$

- Not clear at all how to get $C_0 - C_4$ with 5 multiplications only ...

- We now look for a method for getting these coefficients without any guesswork!

- Let
$$A = A_2\, 2^{2k} + A_1\, 2^k + A_0$$

- We f

$$P_A(x) = A_2\, x^2 +$$
$$P_B(x) = B_2\, x^2 +$$

- Note that
$$\begin{aligned} A &= A_2\,(2^k)^2 + A_1\, 2^k + A_0 = P_A(2^k);\\ B &= B_2\,(2^k)^2 + B_1\, 2^k + B_0 = P_B(2^k). \end{aligned}$$

The Karatsuba trick: slicing into 3 pieces

- If we manage to compute somehow the product polynomial

$$P_C(x) = P_A(x)P_B(x) = C_4\,x^4 + C_3\,x^3 + C_2\,x^2 + C_1\,x + C_0,$$

  with only 5 multiplications, we can then obtain the product of numbers A a

  $A \cdot$ ... $+ C_1 2^k + C_0,$

- Not

- Since the product polynomial $P_C(x) =$
  need 5 values to **uniquely determine**

- We choose **the smallest possible** 5 **i** ... heir absolute value), i.e., -2, -1, 0, 1, 2.

- Thus, we compute

$$P_A(-2), P_A(-1), P_A(0), P_A(1), P_A(2)$$

$$P_B(-2), P_B(-1), P_B(0), P_B(1), P_B(2)$$

- For $P_A(x) = A_2x^2 + A_1x + A_0$ we have

$$
\begin{aligned}
P_A(-2) &= A_2(-2)^2 + A_1(-2) + A_0 = 4A_2 - 2A_1 + A_0 \\
P_A(-1) &= A_2(-1)^2 + A_1(-1) + A_0 = A_2 - A_1 + A_0 \\
P_A(0) &= A_2 0^2 + A_1 0 + A_0 = A_0
\end{aligned}
$$

- Similarly, for $P_B(x) = B_2x^2 + B_1x + B_0$ we have

$$
\begin{aligned}
P_B(-2) &= B_2(-2)^2 + B_1(-2) \\
P_B(-1) &= B_2(-1)^2 + B_1(-1) \\
P_B(0) &= B_2 0^2 + B_1 0 + B_0 = B_0 \\
P_B(1) &= B_2 1^2 + B_1 1 + B_0 = B_2 + B_1 + B_0 \\
P_B(2) &= B_2 2^2 + B_1 2 + B_0 = 4B_2 + 2B_1 + B_0.
\end{aligned}
$$

- These evaluations involve only additions because $2A = A + A$; $4A = 2A + 2A$.

The Karatsuba trick: slicing into 3 pieces

- Having obtained $P_A(-2), P_A(-1), P_A(0), P_A(1), P_A(2)$ and $P_B(-2), P_B(-1), P_B(0), P_B(1), P_B(2)$ we can now obtain $P_C(-2), P_C(-1), P_C(0), P_C(1), P_C(2)$ with only 5 multiplications of large numbers:

$$\begin{aligned} P_C(-2) &= P_A(-2)P_B(-2) \\ &= (A_0 \quad 2A_1 + 4A_2)(B_0 \quad 2B_1 + 4B_2) \end{aligned}$$

$$\begin{aligned} P_C(0) &= P_A(0)P_B(0) \\ &= A_0B_0 \end{aligned}$$

$$\begin{aligned} P_C(1) &= P_A(1)P_B(1) \\ &= (A_0 + A_1 + A_2)(B_0 + B_1 + B_2) \end{aligned}$$

$$\begin{aligned} P_C(2) &= P_A(2)P_B(2) \\ &= (A_0 + 2A_1 + 4A_2)(B_0 + 2B_1 + 4B_2) \end{aligned}$$

- Thus, if we represent the product $C(x) = P_A(x)P_B(x)$ in the coefficient form as $C(x) = C_4x^4 + C_3x^3 + C_2x^2 + C_1x + C_0$ we get

$$
\begin{aligned}
C_4(-2)^4 + C_3(-2)^3 + C_2(-2)^2 + C_1(-2) + C_0 &= P_C(-2) = P_A(-2)P_B(-2)\\
C_4(-1)^4 + C_3(-1)^3 + C_2(-1)^2 + C_1(-1) + C_0 &= P_C(-1) = P_A(-1)P_B(-1)\\
C_4 0^4 + C_3 0^3 + C_2 0^2 + C_1 \cdot 0 + C_0 &= P_C(0) = P_A(0)P_B(0)\\
C_4 1^4 + C_3 1^3 + C_2 1^2 + C_1 \cdot 1 + C_0 &= P_C(1) = P_A(1)P_B(1)\\
C_4 2^4 + C_3 2^3 + C_2 2^2 + C_1 \cdot 2 + C_0 &= P_C(2) = P_A(2)P_B(2).
\end{aligned}
$$

- Simplifying the left side we obtain

$$
\begin{aligned}
16C_4 - 8C_3 + 4C_2 - 2C_1 + C_0 &= P_C(-2)\\
C_4 - C_3 + C_2 - C_1 + C_0 &= P_C(-1)\\
C_0 &= P_C(0)\\
C_4 + C_3 + C_2 + C_1 + C_0 &= P_C(1)\\
16C_4 + 8C_3 + 4C_2 + 2C_1 + C_0 &= P_C(2)
\end{aligned}
$$

- Solving this system of linear equations for C_0, C_1, C_2, C_3, C_4 produces (as an exercise solve this system by hand, using the Gaussian elimination)

$$C_0 = P_C(0)$$

$$C_1 = \frac{P_C(-2)}{12} - \frac{2P_C(-1)}{3} + \frac{2P_C(1)}{3} - \frac{P_C(2)}{12}$$

$$\frac{P_C(\;\;2)}{} \quad \frac{2P_C(\;\;1)}{} \quad \frac{5P_C(0)}{} \quad \frac{2P_C(1)}{} \quad \frac{P_C(2)}{4}$$

$$C_4 = \frac{C}{24} - \frac{C}{6} + \frac{C}{4} \quad \frac{C}{} + \frac{C}{}$$

- Note that these expressions do not involve any multip numbers and thus can be done in linear time.
- With the coefficients C_0, C_1, C_2, C_3, C_4 ob polynomial $P_C(x) = C_0 + C_1 x + C_2 x^2 + C_3 x^3 + C_4 x^4$.
- We can now compute $P_C(2^k) = C_0 + C_1 2^k + C_2 2^{2k} + C_3 2^{3k} + C_4 2^{4k}$ in linear time, because computing $P_C(2^k)$ involves only binary shifts of the coefficients plus $O(k)$ additions.
- Thus we have obtained $A \cdot B = P_A(2^k) P_B(2^k) = P_C(2^k)$ with only 5 multiplications! Here is the complete algorithm:

function $\text{MULT}(A, B)$

obtain A_0, A_1, A_2 and B_0, B_1, B_2 such that $A = A_2\, 2^{2\,k} + A_1\, 2^k + A_0$; $\quad B = B_2\, 2^{2\,k} + B_1\, 2^k + B_0$;

form polynomials $P_A(x) = A_2 x^2 + A_1 x + A_0$; $\quad P_B(x) = B_2 x^2 + B_1 x + B_0$;

$$
\begin{array}{ll}
P_A(-2) \leftarrow 4A_2 - 2A_1 + A_0 & P_B(-2) \leftarrow 4B_2 - 2B_1 + B_0 \\
P_A(-1) \leftarrow A_2 - A_1 + A_0 & P_B(-1) \leftarrow B_2 - B_1 + B_0 \\
P_A(0) \leftarrow A_0 & P_B(0) \leftarrow B_0 \\
P_A(1) \leftarrow A_2 + A_1 + A_0 & P_B(1) \leftarrow B_2 + B_1 + B_0 \\
P_A(2) \leftarrow 4A_2 + 2A_1 + A_0 & P_B(2) \leftarrow 4B_2 + 2B_1 + B_0
\end{array}
$$

[illegible]

$$C_0 \leftarrow P_C(0); \qquad C_1 \leftarrow \frac{P_C(-2)}{12} - \frac{2P_C(\ \ 1)}{} \quad \frac{2P_C(1)}{} \quad \frac{P_C(2)}{}$$

$$C_2 \leftarrow -\frac{P_C(-2)}{24} + \frac{2P_C(-1)}{3} \quad \frac{5P_C(0)}{4} \quad \frac{}{} \quad \frac{}{}$$

$$C_3 \leftarrow -\frac{P_C(-2)}{12} + \frac{P_C(-1)}{6} - \frac{P_C(1)}{6} + \frac{}{12}$$

$$C_4 \leftarrow \frac{P_C(-2)}{24} - \frac{P_C(-1)}{6} + \frac{P_C(0)}{4} - \frac{P_C(1)}{6} + \frac{P_C(2)}{24}$$

form $P_C(x) = C_4 x^4 + C_3 x^3 + C_2 x^2 + C_1 x + C_0$; compute

$P_C(2^k) = C_4 2^{4k} + C_3 2^{3k} + C_2 2^{2k} + C_1 2^k + C_0$

return $P_C(2^k) = A \cdot B$.

end function

- How fast is this algorithm?
- We have replaced a multiplication of two n bit numbers with 5 mul
  and t
- thu

3

- We now apply the Master Theorem:
  we have $a = 5$, $b = 3$, so we consider n

- Clearly, the first case of the MT applies and we get $T(n) = O(n^{\log_3 5}) < O(n^{1.47})$.

- Recall that the original Karatsuba algorithm runs in time

$$n^{\log_2 3} \approx n^{1.58} > n^{1.47}.$$

- Thu

- Then why not slice numbers A and B into even larger number of slices? Maybe we can get even faster algorithm?

- The answer is, in a sense, BOTH yes and no, so lets see w we slice numbers into $p+1$ many (approxi $p = 1, 2, 3, \ldots$....

The general case - slicing the input numbers A, B into $p+1$ many slices

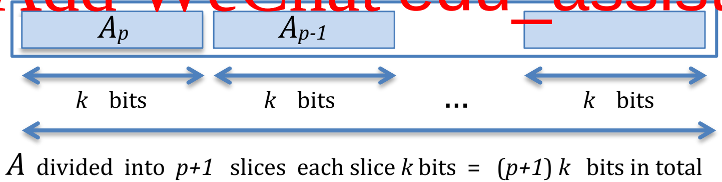
- For simplicity, let us assume A and B have exactly $(p+1)k$ bits (otherwise one of the slices will have to be shorter);

- Note: p is a fixed (smallish) number, a fixed parameter of our design – $p+1$ is the number of slices we are going to make, but k depends on the inp

- Slic

$p \quad p-1 \quad 0$

$$B = B_p 2^{kp} + B_{p-1}$$



- We form the naturally corresponding polynomials:

$$P_A(x) = A_p x^p + A_{p-1} x^{p-1} + \cdots + A_0$$
$$P_B(x) = B_p x^p + B_{p-1} x^{p-1} + \cdots + B_0$$

- As before, we have:
$A = P_A(2^k);\;\; B = P_B(2^k);\;\; AB = P_A(2^k)P_B(2^k) = (P_A(x)\;\; P_B(x))\,|_{x=2^k}$

- Sinc

we a
  - we will first figure out how to multiply polyno

$$P_C(x) = P_A$$

  - then we evaluate $P_C(2^k)$.

- Note that $P_C(x) = P_A(x) \cdot P_B(x)$ is of degree $2p$:

$$P_C(x) = \sum_{j=0}^{2p} C_j x^j$$

Generalizing Karatsuba's algorithm

- Example:

$$(A_3x^3 + A_2x^2 + A_1x + A_0)(B_3x^3 + B_2x^2 + B_1x + B_0) =$$
$$A_3B_3x^6 + (A_2B_3 + A_3B_2)x^5 + (A_1B_3 + A_2B_2 + A_3B_1)x^4$$
$$+(A_0B_3 + A_1B_2 + A_2B_1 + A_3B_0)x^3 + (A_0B_2 + A_1B_1 + A_2B_0)x^2$$
$$+(A_0B_1 + A_1B_0)x + A_0B_0$$

- In ge

$$P_B(x) = B_px^p + B_{p-1}x^{p-1} + \dots + B_0$$

we have

$$P_A(x) \cdot P_B(x) = \sum_{j=0}^{2p} \left(\sum_{i+k=j} A_iB_k \right) x^j = \sum_{j=0}^{2p} C_j\, x^j$$

- We need to find the coefficients $C_j = \sum_{i+k=j} A_iB_k$ without performing $(p+1)^2$ many multiplications necessary to get all products of the form A_iB_k.

A VERY IMPORTANT DIGRESSION:

If you have two sequences $\vec{A} = (A_0, A_1, \ldots, A_{p-1}, A_p)$ and $\vec{B} = (B_0, B_1, \ldots, B_{m-1}, B_m)$, and if you form the two corresponding polynomials

$$P_A(x) = A_0 + A_1 x + \ldots + A_{p-1}x^{p-1} + A_p x^p$$
$$P_B(x) = B_0 + B_1 x + \ldots + B_{m-1}x^{m-1} + B_m x^m$$

and if you mu

$j=0$ $i+k=j$ $j=0$

then the sequence $\vec{C} = (C_0, C_1, \ldots, C_{p+m})$ of the coe polynomial, with these coefficients given by

$$C_j = \sum_{i+k=j} A_i B_k, \quad \text{for} \quad 0 \leq j \leq p+m,$$

is **extremely important** and is called the **LINEAR CONVOLUTION** of sequences $\vec{A}$ and $\vec{B}$ and is denoted by $\vec{C} = \vec{A} \star \vec{B}$.

AN IMPORTANT DIGRESSION:

- For example, if you have an audio signal and you want to emphasise the bass sounds, you would pass the sequence of discrete samples of the signal through a digital filter which amplifies the low frequencies more than the medium and the high audio frequencies.
- This is accomplished by computing the linear convolution of the sequence of discrete samples of the signal with a sequence of values which correspond to that filter, called *the impulse response* of the filter.
- This

  the pr

  1. e input

  2. polynomial $P_B(x)$ whose coefficients

     called impulse response of the filter (they de

     filtering you want to do).
- Convolutions are bread-and-butter of signal proce

  is **extremely important** to find fast ways of multiplying two polynomials of possibly very large degrees.
- In signal processing these degrees can be greater than 1000.
- This is the main reason for us to study methods of fast computation of convolutions (aside of finding products of large integers, which is what we are doing at the moment).

Coefficient vs value representation of polynomials

- Every polynomial $P_A(x)$ of degree p is uniquely determined by its values at any $p+1$ distinct input values $x_0, x_1, \ldots, x_p$:

$$P_A(x) \leftrightarrow \{(x_0, P_A(x_0)), (x_1, P_A(x_1)), \ldots, (x_p, P_A(x_p))\}$$

- For … ia a
mat…

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^p \\ 1 & x_1 & x_1^2 & \ldots & x_1^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_p & x_p^2 & \ldots & x_p^p \end{pmatrix} \tag{1}$$

- It can be shown that if x_i are all distinct then this matrix is invertible.
- Such a matrix is called *the Vandermonde matrix*.

- Thus, if all x_i are all distinct, given any values $P_A(x_0), P_A(x_1), \ldots, P_A(x_p)$ the coefficients $A_0, A_1, \ldots, A_p$ of the polynomial $P_A(x)$ are uniquely determined:

$$\begin{pmatrix} A_0 \\ A_1 \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^p \\ 1 & x_1 & x_1^2 & \ldots & x_1^p \end{pmatrix}^{-1} \begin{pmatrix} P_A(x_0) \\ P_A(x_1) \end{pmatrix} \quad (2)$$

- Equations (1) and (2) show how we can commute be

1. a representation of a polynomial P $A_p, A_{p-1}, \ldots, A_0$, i.e. $P_A(x) = A_p$
2. a representation of a polynomial $P_A(x)$ via its values

$$P_A(x) \leftrightarrow \{(x_0, P_A(x_0)), (x_1, P_A(x_1)), \ldots, (x_p, P_A(x_p))\}$$

- If we fix the inputs $x_0, x_1, \ldots, x_p$ then commuting between a representation of a polynomial $P_A(x)$ via its coefficients and a representation via its values at these points is done via the following two matrix multiplications, with matrices made up from **constants**:

$$\begin{pmatrix} P_A(x_0) \\ P_A(x_1) \\ \vdots \\ P_A(x_p) \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^p \\ 1 & x_1 & x_1^2 & \ldots & x_1^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_p & x_p^2 & \ldots & x_p^p \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ \vdots \\ A_p \end{pmatrix}$$

$$\begin{pmatrix} A_0 \\ A_1 \\ \vdots \\ A_p \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^p \\ 1 & x_1 & x_1^2 & \ldots & x_1^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_p & x_p^2 & \ldots & x_p^p \end{pmatrix}^{-1} \begin{pmatrix} P_A(x_0) \\ P_A(x_1) \\ \vdots \\ P_A(x_p) \end{pmatrix}$$

- Thus, for fixed input values $x_0, \ldots, x_p$ this switch between the two kinds of representations is done in **linear time**!

1. Given two polynomials of degree at most p,

$$P_A(x) = A_p x^p + \ldots + A_0; \qquad P_B(x) = B_p x^p + \ldots + B_0$$

convert them into value representation at $2p+1$ distinct points $x_0, x_1, \ldots, x_{2p}$:

$$P_A(x) \leftrightarrow \{(x_0, P_A(x_0)), (x_1, P_A(x_1)), \ldots, (x_{2p}, P_A(x_{2p}))\}$$

$))\}$

- p A B ther than just $p+1$ points!

2. Multiply these two polynomials point-wise, using 2 ly.

$$P_A(x)P_B(x) \leftrightarrow \{(x_0, \underbrace{P_A(x_0)P_B(x_0)}_{P_C(x_0)}), (x_1, \underbrace{P_A(x_1) \quad B \quad 1}_{P_C(x_1)} \quad 2p \quad \underbrace{A \quad 2p \quad B(x_{2p})}_{P_C(x_{2p})})\}$$

3. Convert such value representation of $P_C(x) = P_A(x)P_B(x)$ back to coefficient form

$$P_C(x) = C_{2p}x^{2p} + C_{2p-1}x^{2p-1} + \ldots + C_1 x + C_0;$$

Fast multiplication of polynomials - continued

- What values should we choose for $x_0, x_1, \ldots, x_{2p}$??
- Key idea: use $2p + 1$ smallest possible integer values!

$$\{-p, -(p-1), \ldots, -1, 0, 1, \ldots, p-1, p\}$$

- So we find the values $P_A(m)$ and $P_B(m)$ for all m such that $p \quad m \leq p$.
- Rem A, B.
- Mul d can be don i

$$d \cdot A = A + A$$

- Thus, all the values

$$P_A(m) = A_p m^p + A_{p-1} m^{p-1} + \cdots + A_0 : \quad -p \leq m \leq p,$$

$$P_B(m) = B_p m^p + B_{p-1} m^{p-1} + \cdots + B_0 : \quad -p \leq m \leq p.$$

can be found in time linear in the number of bits of the input numbers!

Fast multiplication of polynomials - ctd.

- We now perform $2p+1$ **multiplications of large numbers** to obtain

$$P_A(-p)P_B(-p),\ \dots,\ P_A(-1)P_B(-1),\quad P_A(0)P_B(0),\quad P_A(1)P_B(1),\ \dots,\ P_A(p)P_B(p)$$

- For $P_C(x) = P_A(x)P_B(x)$ these products are $2p+1$ many values of $P_C(x)$:

$$P_C(-p) = P_A(-p)P_B(-p),\ \dots,\ P_C(0) = P_A(0)P_B(0),\ \dots,\ P_C(p) = P_A(p)P_B(p)$$

- Let ... $C(x)$, i.e., let

- We now have:

$$C_{2p}(-p)^{2p} + C_{2p-1}(-p)^{2p-1} + \cdots +$$

$$C_{2p}(-(p-1))^{2p} + C_{2p-1}(-(p-1))^{2} \ \dots\ 1))$$

$$\vdots$$

$$C_{2p}(p-1)^{2p} + C_{2p-1}(p-1)^{2p-1} + \cdots + C_0 = P_C(p-1)$$

$$C_{2p}p^{2p} + C_{2p-1}p^{2p-1} + \cdots + C_0 = P_C(p)$$

Fast multiplication of polynomials - ctd.

- This is just a system of linear equations, that can be solved for $C_0, C_1, \ldots, C_{2p}$:

$$\begin{pmatrix} 1 & -p & (-p)^2 & \ldots & (-p)^{2p} \\ 1 & -(p-1) & (-(p-1))^2 & \ldots & (-(p-1))^{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & p-1 & (p-1)^2 & \ldots & (p-1)^{2p} \\ 1 & p & p^2 & \ldots & p^{2p} \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{2p-1} \\ C_{2p} \end{pmatrix} = \begin{pmatrix} P_C(-p) \\ P_C(-(p-1)) \\ \vdots \\ P_C(p-1) \\ P_C(p) \end{pmatrix}$$

- i.e., w

$$\begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{2p} \end{pmatrix} = \begin{pmatrix} 1 & -p & (-p)^2 & \ldots & (-p)^{2p} \\ 1 & -(p-1) & (-(p-1))^2 & \ldots & (-(p-1))^{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & p-1 & (p-1)^2 & \ldots & (p-1)^{2p} \\ 1 & p & p^2 & \ldots & p^{2p} \end{pmatrix}^{-1} \begin{pmatrix} P_C(-p) \\ P_C(-(p-1)) \\ \vdots \\ P_C(p-1) \\ P_C(p) \end{pmatrix}$$

- But the inverse matrix also involves only constants depending on p only;
- Thus the coefficients C_i can be obtained in linear time.
- So here is the algorithm we have just described:

```
function Mult(A, B)
    if |A| = |B| < p + 1 then return AB
    else
        obtain p + 1 slices A_0, A_1, ..., A_p and B_0, B_1, ..., B_p such that
```

$$A = A_p 2^{p\,k} + A_{p-1} 2^{(p-1)\,k} + \ldots + A_0$$

$$B = B_p 2^{p\,k} + B_{p-1} 2^{(p-1)\,k} + \ldots + B_0$$


```
        form polynomials
```

$$P_A(x) = A_p x^p + A_{p-1} x^{(p-1)} + \ldots + A_0$$

$$p \qquad (p \quad 1)$$

```
        f
```



```
        end for
        compute C_0, C_1, ... C_2p via
```


$$\begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{2p} \end{pmatrix} = \begin{pmatrix} 1 & -p & (-p)^2 & \ldots & \\ 1 & -(p-1) & (-(p-1))^2 & \ldots & \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & p & p^2 & \ldots & p^{2p} \end{pmatrix} \begin{pmatrix} \\ 1)) \\ \vdots \\ P_C(p) \end{pmatrix}.$$

```
        form P_C(x) = C_2p x^2p + ... + C_0 and compute P_C(2^k)
        return P_C(2^k) = A · B
    end if
end function
```

How fast is our algorithm?

- it is easy to see that the values of the two polynomials we are multiplying have at most $k + s$ bits where s is a constant which depends on p but does NOT depend on k:

$$P_A(m) = A_p m^p + A_{p-1} m^{p-1} + \cdots + A_0 ; \quad -p \le m \le p.$$

This is because each A_i is smaller than 2^k because each A_k has k bits; thus

$$k = s + k$$

- Thu $p + 1$ mult the nu

- So we get the following recurrence for the complexity of

$$T((p+1)k) = (2p+1)$$

- Let $n = (p+1)k$. Then

$$T(n) = \underbrace{(2p+1)}_{a} \, T\Big(\underbrace{\frac{n}{p+1}}_{b} + s\Big) + \frac{c}{p+1}\, n$$

- Since s is constant, its impact can be neglected.

How fast is our algorithm?

$$T(n) = \underbrace{(2p+1)}_{a}\, T\Big(\underbrace{\frac{n}{p+1}}_{b} + s\Big) + \frac{c}{p+1}\, n$$

- Sinc
  $\log_b$

- Cons $\log_b a - \varepsilon$).

- Thus, with $a = 2p+1$ and $b = p+1$ the first case of t

- so we get:

$$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{\log_{p+1}(2p+1)}\right)$$

- Note that

$$n^{\log_{p+1}(2p+1)} < n^{\log_{p+1} 2(p+1)} = n^{\log_{p+1} 2+\log_{p+1}(p+1)}$$

$$= n^{1+\log_{p+1} 2} = n^{1+\frac{1}{\log_2(p+1)}}$$

- Thus, by choosing a sufficiently large p, we can get a run time arbitrarily clos

- Ho
in ti

$$n^{1.1} = n^{1+\frac{1}{\log_2(p+1)}} \to \frac{1}{\log_2(p+1)} = \frac{1}{10}$$

- Thus, we would have to slice the input numbers into $2^{10} = 1024$ pieces!!

- We would have to evaluate polynomials $P_A(x)$ and $P_B(x)$ both of degree p at values up to p.

- However, $p = 2^{10}$, so evaluating $P_A(p) = A_p p^p + \ldots + A_0$ involves multiplication of A_p with $p^p = (2^{10})^{2^{10}} \approx 1.27 \times 10^{3079}$.

- Thu can **the** nt c is abs

- Consequently, slicing the input numbers in more than just a few slices results in a hopelessly slow algorithm, despite the f asymptotic bounds improve as we increase the nu

- The moral is: **In practice, asymptotic es size of the constants hidden by the O-notation are not estimated and found to be reasonably small!!!**

- **Crucial question:** Are there numbers $x_0, x_1, \ldots, x_p$ such that the size of x_i^p does not grow uncontrollably?
- Answer: YES; they are the complex numbers z_i lying on the unit circle, i.e., such that $|z_i| = 1$!

- This motivates us to consider values of polynomials at inputs which are equally spaced complex numbers all lying on the unit circle.
- The s **sform** **(D** eval

- We will present a very fast algorithm for computin **the Fast Fourier Transform**, abbrevi

- The Fast Fourier Transform is **the most** and is thus arguably **the most important algorithm of all**.
- Every mobile phone performs thousands of FFT runs each second, for example to compress your speech signal or to compress images taken by your camera, to mention just a few uses of the FFT.

PUZZLE!

The warden meets with 23 new prisoners when they arrive. He tells them, “You may meet today and plan a strategy. But after today, you will be in isolated cells and will have no communication with one another. In the prison there is a switch room, which contains two light switches labeled A and B, each of which can be in either the on or the off position. I am not telling you their present positions. The switches are not conn
inclined, I wi
This prison
move one, b
either. The
until I lead the next prisoner there, and he’ll be instructed to do the same thing. I’m going to choose prisoners at random. I may choose the same g
a row, or I may jump around and come back. But, given enough ti
would eventually visit the switch room many times. At any ti
declare to me: “We have all visited the switch room. If it is true, the
be set free. If it is false, and somebody has not yet visited the switch room, you will be fed to the alligators.”

What is the strategy the prisoners can devise to gain their freedom?