Algorithms
COMP3121/3821/9101/9801
4. FAST LARGE INTEGER MULTIPLICATION

School of Computer Science and Engineering
University of New South Wales Sydney

Basics revisited: how do we multiply two numbers?

- The primary school algorithm:

```
          X X X X  <- first input integer
        * X X X X  <- second input integer
          -------
          X X X X  \
        X X X X     \ O(n^2) intermediate operations:
      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??**

The Karatsuba trick

- Take the two input numbers $\mathbf{a}$ and $\mathbf{b}$, and split them into halves:

$$\mathbf{a} = \mathbf{a}_1 2^{\frac{n}{2}} + \mathbf{a}_0 \qquad \mathbf{a} = \overbrace{XX\ldots X}^{\mathbf{a}_1}\overbrace{XX\ldots X}^{\mathbf{a}_0}$$

$$\underbrace{}_{n/2 \text{ bits}} \quad \underbrace{}_{n/2 \text{ bits}}$$

$$\mathbf{b} = \mathbf{b}_1 2^{\frac{n}{2}} + \mathbf{b}_0$$

- $\mathbf{a}_1 = \text{MoreSignificantPart}(\mathbf{a})$; $\mathbf{a}_0 = \text{LessSignificantPart}(\mathbf{a})$;

- $\mathbf{ab}$ can now be calculated as follows:

$$\begin{aligned} \mathbf{ab} &= \mathbf{a}_1\mathbf{b}_1 2^n + (\mathbf{a}_1\mathbf{b}_0 + \mathbf{a}_0\mathbf{b}_1)2^{\frac{n}{2}} + \mathbf{a}_0\mathbf{b}_0 \\ &= \mathbf{a}_1\mathbf{b}_1 2^n + ((\mathbf{a}_1 + \mathbf{a}_0)(\mathbf{b}_1 + \mathbf{b}_0) - \mathbf{a}_1\mathbf{b}_1 - \mathbf{a}_0\mathbf{b}_0)2^{\frac{n}{2}} + \mathbf{a}_0\mathbf{b}_0 \end{aligned}$$

The Karatsuba trick

`mult(a, b)`

if $|\mathbf{a}| = |\mathbf{b}| = 1$ **then return ab**

$\mathbf{a}_1 \leftarrow \text{MoreSignificantPart}(\mathbf{a})$

$\mathbf{a}_0 \leftarrow \text{LessSignificantPart}(\mathbf{a})$

$\mathbf{b}_1 \leftarrow \text{MoreSignificantPart}(\mathbf{b})$

$\mathbf{b}_0 \leftarrow \text{LessSignificantPart}(\mathbf{b})$

$u \leftarrow \mathbf{a}_0 + \mathbf{a}_1$

$v \leftarrow \mathbf{b}_0 + \mathbf{b}_1$

$x \leftarrow$ `mult(`$\mathbf{a}_0$`,` $\mathbf{b}_0$`)`

$w \leftarrow$ `mult(`$\mathbf{a}_1$`,` $\mathbf{b}_1$`)`

$y \leftarrow$ `mult(`u`,` v`)`

return $w\,2^n + (y - x - w)\,2^{n/2} + x$

How many steps does this take?

Recurrence: $T(n) = 3\,T\left(\frac{n}{2}\right) + cn \rightsquigarrow$ Master Theorem, case 1 applies: $T(n) = \Theta(n^{\log_2 3}) < \Theta(n^{1.585})$

Generalizing Karatsuba's algorithm

- Can we do better if we break the numbers in more than two pieces?
- Let's try breaking the numbers $\mathbf{a}$ and $\mathbf{b}$ into 3 pieces; then with $k = n/3$ we obtain

$$\mathbf{a} = \underbrace{XXX\ldots XX}_{k \text{ bits of } \mathbf{a}_2}\underbrace{XXX\ldots XX}_{k \text{ bits of } \mathbf{a}_1}\underbrace{XXX\ldots XX}_{k \text{ bits of } \mathbf{a}_0}$$

i.e.,

$$\begin{aligned} \mathbf{a} &= \mathbf{a}_2 2^{2k} + \mathbf{a}_1 2^k + \mathbf{a}_0 \\ \mathbf{b} &= \mathbf{b}_2 2^{2k} + \mathbf{b}_1 2^k + \mathbf{b}_0 \end{aligned}$$

- So,

$$\begin{aligned} \mathbf{ab} = \mathbf{a}_2\mathbf{b}_2\, 2^{4k} + (\mathbf{a}_2\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_2)2^{3k} + (\mathbf{a}_2\mathbf{b}_0 + \mathbf{a}_1\mathbf{b}_1 + \mathbf{a}_0\mathbf{b}_2)2^{2k} + \\ + (\mathbf{a}_1\mathbf{b}_0 + \mathbf{a}_0\mathbf{b}_1)2^k + \mathbf{a}_0\mathbf{b}_0 \end{aligned}$$

The Karatsuba trick

$$\mathbf{c} = \mathbf{ab} = \mathbf{a}_2\mathbf{b}_2\,2^{4k} + (\mathbf{a}_2\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_2)2^{3k} + (\mathbf{a}_2\mathbf{b}_0 + \mathbf{a}_1\mathbf{b}_1 + \mathbf{a}_0\mathbf{b}_2)2^{2k} + \\ + (\mathbf{a}_1\mathbf{b}_0 + \mathbf{a}_0\mathbf{b}_1)2^{k} + \mathbf{a}_0\mathbf{b}_0$$

- we need only 5 coefficients:

$$\mathbf{c}_4 = \mathbf{a}_2\mathbf{b}_2 \quad \mathbf{c}_3 = \mathbf{a}_2\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_2 \quad \mathbf{c}_2 = \mathbf{a}_2\mathbf{b}_0 + \mathbf{a}_1\mathbf{b}_1 + \mathbf{a}_0\mathbf{b}_2 \quad \mathbf{c}_1 = \mathbf{a}_1\mathbf{b}_0 + \mathbf{a}_0\mathbf{b}_1 \quad \mathbf{c}_0 = \mathbf{a}_0\mathbf{b}_0$$

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

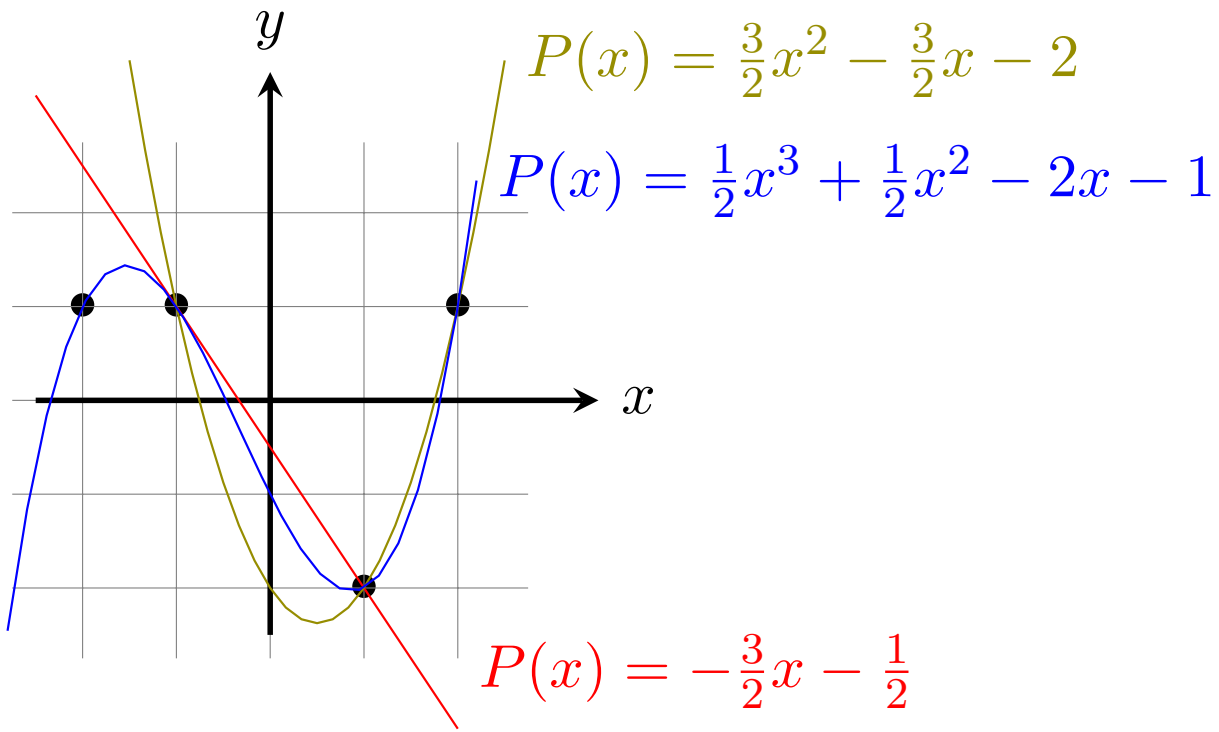
$$(\mathbf{a}_2 + \mathbf{a}_1 + \mathbf{a}_0)(\mathbf{b}_2 + \mathbf{b}_1 + \mathbf{b}_0) = \\ \mathbf{a}_0\mathbf{b}_0 + \mathbf{a}_1\mathbf{b}_0 + \mathbf{a}_2\mathbf{b}_0 + \mathbf{a}_0\mathbf{b}_1 + \mathbf{a}_1\mathbf{b}_1 + \mathbf{a}_2\mathbf{b}_1 + \mathbf{a}_0\mathbf{b}_2 + \mathbf{a}_1\mathbf{b}_2 + \mathbf{a}_2\mathbf{b}_2 \quad ?$$

- Not clear at all how to get $\mathbf{c}_0 - \mathbf{c}_4$ with 5 multiplications only ...

Theorem

Let $(x_0, y_0), \ldots, (x_n, y_n)$ be pairs of real numbers. If all x_i are distinct, then there exists a unique polynomial P of degree n or less such that $P(x_i) = y_i$ for all $0 \leq i \leq n$.

$$x_0 = -1 \quad y_0 = 1$$
$$x_1 = 1 \quad y_1 = -2$$
$$x_2 = 2 \quad y_2 = 1$$
$$x_3 = -2 \quad y_3 = 1$$



Interlude: Polynomial Interpolation

Theorem

Let $(x_0, y_0), \ldots, (x_n, y_n)$ be pairs of real numbers. If all x_i are distinct, then there exists a unique polynomial P of degree n or less such that $P(x_i) = y_i$ for all $0 \leq i \leq n$.

$$\begin{aligned} x_0 &= -1 & y_0 &= 1 \\ x_1 &= 1 & y_1 &= -2 \\ x_2 &= 2 & y_2 &= 1 \\ x_3 &= -2 & y_3 &= 1 \end{aligned}$$

How to find the coefficients?

Solve the following system:

$$\begin{aligned} y_0 = P(x_0) &= p_3 x_0^3 + p_2 x_0^2 + p_1 x_0 + p_0 \\ y_1 = P(x_1) &= p_3 x_1^3 + p_2 x_1^2 + p_1 x_1 + p_0 \\ y_2 = P(x_2) &= p_3 x_2^3 + p_2 x_2^2 + p_1 x_2 + p_0 \\ y_3 = P(x_3) &= p_3 x_3^3 + p_2 x_3^2 + p_1 x_3 + p_0 \\ 1 &= p_3(-1)^3 + p_2(-1)^2 + p_1(-1) + p_0 \\ -2 &= p_3(1)^3 + p_2(1)^2 + p_1(1) + p_0 \\ 1 &= p_3(2)^3 + p_2(2)^2 + p_1(2) + p_0 \\ 1 &= p_3(-2)^3 + p_2(-2)^2 + p_1(-2) + p_0 \\ 1 &= -p_3 + p_2 - p_1 + p_0 \end{aligned}$$

The Karatsuba trick: slicing into 3 pieces

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

$$\mathbf{a} = \mathbf{a}_2\, 2^{2k} + \mathbf{a}_1\, 2^k + \mathbf{a}_0$$
$$\mathbf{b} = \mathbf{b}_2\, 2^{2k} + \mathbf{b}_1\, 2^k + \mathbf{b}_0$$

- We form naturally corresponding polynomials:

$$A(x) = \mathbf{a}_2\, x^2 + \mathbf{a}_1\, x + \mathbf{a}_0;$$
$$B(x) = \mathbf{b}_2\, x^2 + \mathbf{b}_1\, x + \mathbf{b}_0.$$

- Note that

$$\mathbf{a} = \mathbf{a}_2\, (2^k)^2 + \mathbf{a}_1\, 2^k + \mathbf{a}_0 = A(2^k);$$
$$\mathbf{b} = \mathbf{b}_2\, (2^k)^2 + \mathbf{b}_1\, 2^k + \mathbf{b}_0 = B(2^k).$$

The Karatsuba trick: slicing into 3 pieces

- If we manage to compute somehow the product polynomial

$$C(x) = A(x)B(x) = \mathbf{c}_4\, x^4 + \mathbf{c}_3\, x^3 + \mathbf{c}_2\, x^2 + \mathbf{c}_1\, x + \mathbf{c}_0,$$

  with only 5 multiplications, we can then obtain the product of numbers a and b simply as

$$\mathbf{a} \cdot \mathbf{b} = A(2^k)B(2^k) = C(2^k) = \mathbf{c}_4\, 2^{4k} + \mathbf{c}_3\, 2^{3k} + \mathbf{c}_2\, 2^{2k} + \mathbf{c}_1\, 2^k + \mathbf{c}_0,$$

- Note that the right hand side involves only shifts and additions.
- Since the product polynomial $C(x) = A(x)B(x)$ is of degree 4 we need 5 values to **uniquely determine** $C(x)$.
- We choose **the smallest possible** 5 **integer values** (smallest by their absolute value), i.e., -2, -1, 0, 1, 2.
- Thus, we compute

$$A(-2), A(-1), A(0), A(1), A(2)$$
$$B(-2), B(-1), B(0), B(1), B(2)$$

The Karatsuba trick: slicing into 3 pieces

- For $A(x) = \mathbf{a}_2 x^2 + \mathbf{a}_1 x + \mathbf{a}_0$ we have

$$\begin{aligned}
A(-2) &= \mathbf{a}_2(-2)^2 + \mathbf{a}_1(-2) + \mathbf{a}_0 = 4\mathbf{a}_2 - 2\mathbf{a}_1 + \mathbf{a}_0 \\
A(-1) &= \mathbf{a}_2(-1)^2 + \mathbf{a}_1(-1) + \mathbf{a}_0 = \mathbf{a}_2 - \mathbf{a}_1 + \mathbf{a}_0 \\
A(0) &= \mathbf{a}_2 0^2 + \mathbf{a}_1 0 + \mathbf{a}_0 = \mathbf{a}_0 \\
A(1) &= \mathbf{a}_2 1^2 + \mathbf{a}_1 1 + \mathbf{a}_0 = \mathbf{a}_2 + \mathbf{a}_1 + \mathbf{a}_0 \\
A(2) &= \mathbf{a}_2 2^2 + \mathbf{a}_1 2 + \mathbf{a}_0 = 4\mathbf{a}_2 + 2\mathbf{a}_1 + \mathbf{a}_0.
\end{aligned}$$

- Similarly, for $B(x) = \mathbf{b}_2 x^2 + \mathbf{b}_1 x + \mathbf{b}_0$ we have

$$\begin{aligned}
B(-2) &= \mathbf{b}_2(-2)^2 + \mathbf{b}_1(-2) + \mathbf{b}_0 = 4\mathbf{b}_2 - 2\mathbf{b}_1 + \mathbf{b}_0 \\
B(-1) &= \mathbf{b}_2(-1)^2 + \mathbf{b}_1(-1) + \mathbf{b}_0 = \mathbf{b}_2 - \mathbf{b}_1 + \mathbf{b}_0 \\
B(0) &= \mathbf{b}_2 0^2 + \mathbf{b}_1 0 + \mathbf{b}_0 = \mathbf{b}_0 \\
B(1) &= \mathbf{b}_2 1^2 + \mathbf{b}_1 1 + \mathbf{b}_0 = \mathbf{b}_2 + \mathbf{b}_1 + \mathbf{b}_0 \\
B(2) &= \mathbf{b}_2 2^2 + \mathbf{b}_1 2 + \mathbf{b}_0 = 4\mathbf{b}_2 + 2\mathbf{b}_1 + \mathbf{b}_0.
\end{aligned}$$

- These evaluations involve only additions because $2\mathbf{a}_i = \mathbf{a}_i + \mathbf{a}_i$; $4\mathbf{a}_i = 2\mathbf{a}_i + 2\mathbf{a}_i$.

The Karatsuba trick: slicing into 3 pieces

- Having obtained $A(-2), A(-1), A(0), A(1), A(2)$ and $B(-2), B(-1), B(0), B(1), B(2)$ we can now obtain $C(-2), C(-1), C(0), C(1), C(2)$ with only 5 multiplications of large numbers:

$$\begin{aligned} C(-2) &= A(-2)B(-2) \\ &= (\mathbf{a}_0 - 2\mathbf{a}_1 + 4\mathbf{a}_2)(\mathbf{b}_0 - 2\mathbf{b}_1 + 4\mathbf{b}_2) \end{aligned}$$

$$\begin{aligned} C(-1) &= A(-1)B(-1) \\ &= (\mathbf{a}_0 - \mathbf{a}_1 + \mathbf{a}_2)(\mathbf{b}_0 - \mathbf{b}_1 + \mathbf{b}_2) \end{aligned}$$

$$\begin{aligned} C(0) &= A(0)B(0) \\ &= \mathbf{a}_0\mathbf{b}_0 \end{aligned}$$

$$\begin{aligned} C(1) &= A(1)B(1) \\ &= (\mathbf{a}_0 + \mathbf{a}_1 + \mathbf{a}_2)(\mathbf{b}_0 + \mathbf{b}_1 + \mathbf{b}_2) \end{aligned}$$

$$\begin{aligned} C(2) &= A(2)B(2) \\ &= (\mathbf{a}_0 + 2\mathbf{a}_1 + 4\mathbf{a}_2)(\mathbf{b}_0 + 2\mathbf{b}_1 + 4\mathbf{b}_2) \end{aligned}$$

The Karatsuba trick: slicing into 3 pieces

- Thus, if we represent the product $C(x) = A(x)B(x)$ in the coefficient form as $C(x) = \mathbf{c}_4x^4 + \mathbf{c}_3x^3 + \mathbf{c}_2x^2 + \mathbf{c}_1x + \mathbf{c}_0$ we get

$$\begin{aligned}
\mathbf{c}_4(-2)^4 + \mathbf{c}_3(-2)^3 + \mathbf{c}_2(-2)^2 + \mathbf{c}_1(-2) + \mathbf{c}_0 &= C(-2) = A(-2)B(-2)\\
\mathbf{c}_4(-1)^4 + \mathbf{c}_3(-1)^3 + \mathbf{c}_2(-1)^2 + \mathbf{c}_1(-1) + \mathbf{c}_0 &= C(-1) = A(-1)B(-1)\\
\mathbf{c}_4 0^4 + \mathbf{c}_3 0^3 + \mathbf{c}_2 0^2 + \mathbf{c}_1 \cdot 0 + \mathbf{c}_0 &= C(0) = A(0)B(0)\\
\mathbf{c}_4 1^4 + \mathbf{c}_3 1^3 + \mathbf{c}_2 1^2 + \mathbf{c}_1 \cdot 1 + \mathbf{c}_0 &= C(1) = A(1)B(1)\\
\mathbf{c}_4 2^4 + \mathbf{c}_3 2^3 + \mathbf{c}_2 2^2 + \mathbf{c}_1 \cdot 2 + \mathbf{c}_0 &= C(2) = A(2)B(2).
\end{aligned}$$

- Simplifying the left side we obtain

$$\begin{aligned}
16\mathbf{c}_4 - 8\mathbf{c}_3 + 4\mathbf{c}_2 - 2\mathbf{c}_1 + \mathbf{c}_0 &= C(-2)\\
\mathbf{c}_4 - \mathbf{c}_3 + \mathbf{c}_2 - \mathbf{c}_1 + \mathbf{c}_0 &= C(-1)\\
\mathbf{c}_0 &= C(0)\\
\mathbf{c}_4 + \mathbf{c}_3 + \mathbf{c}_2 + \mathbf{c}_1 + \mathbf{c}_0 &= C(1)\\
16\mathbf{c}_4 + 8\mathbf{c}_3 + 4\mathbf{c}_2 + 2\mathbf{c}_1 + \mathbf{c}_0 &= C(2)
\end{aligned}$$

The Karatsuba trick: slicing into 3 pieces

- Solving this system of linear equations for $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4$ we obtain

$$\mathbf{c}_0 = C(0)$$
$$\mathbf{c}_1 = \frac{C(-2)}{12} - \frac{2C(-1)}{3} + \frac{2C(1)}{3} - \frac{C(2)}{12}$$
$$\mathbf{c}_2 = -\frac{C(-2)}{24} + \frac{2C(-1)}{3} - \frac{5C(0)}{4} + \frac{2C(1)}{3} - \frac{C(2)}{24}$$
$$\mathbf{c}_3 = -\frac{C(-2)}{12} + \frac{C(-1)}{6} - \frac{C(1)}{6} + \frac{C(2)}{12}$$
$$\mathbf{c}_4 = \frac{C(-2)}{24} - \frac{C(-1)}{6} + \frac{C(0)}{4} - \frac{C(1)}{6} + \frac{C(2)}{24}$$

- Note that these expressions do not involve any multiplications of TWO large numbers and thus can be done in linear time.
- With the coefficients $\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4$ obtained, we can now form the polynomial $C(x) = \mathbf{c}_0 + \mathbf{c}_1 x + \mathbf{c}_2 x^2 + \mathbf{c}_3 x^3 + \mathbf{c}_4 x^4$.
- We can now compute $C(2^k) = \mathbf{c}_0 + \mathbf{c}_1 2^k + \mathbf{c}_2 2^{2k} + \mathbf{c}_3 2^{3k} + \mathbf{c}_4 2^{4k}$ in linear time, because computing $C(2^k)$ involves only binary shifts of the coefficients plus $O(k)$ additions.
- Thus we have obtained $\mathbf{a} \cdot \mathbf{b} = A(2^k)B(2^k) = C(2^k)$ with only 5 multiplications!
- Here is the complete algorithm:

`mult(a, b)`

if $|\mathbf{a}| = |\mathbf{b}| = 1$ **then return ab**

obtain $\mathbf{a}_0, \mathbf{a}_1, \mathbf{a}_2$ and $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2$ such that

$\mathbf{a} = \mathbf{a}_2\, 2^{2k} + \mathbf{a}_1\, 2^k + \mathbf{a}_0$

$\mathbf{b} = \mathbf{b}_2\, 2^{2k} + \mathbf{b}_1\, 2^k + \mathbf{b}_0$;

$$
\left.
\begin{aligned}
A(-2) &\leftarrow 4\mathbf{a}_2 - 2\mathbf{a}_1 + \mathbf{a}_0 \\
A(-1) &\leftarrow \mathbf{a}_2 - \mathbf{a}_1 + \mathbf{a}_0 \\
A(0) &\leftarrow \mathbf{a}_0 \\
A(1) &\leftarrow \mathbf{a}_2 + \mathbf{a}_1 + \mathbf{a}_0 \\
A(2) &\leftarrow 4\mathbf{a}_2 + 2\mathbf{a}_1 + \mathbf{a}_0
\end{aligned}
\right\} \text{Compute interpolants for } A
$$

$$
\left.
\begin{aligned}
B(-2) &\leftarrow 4\mathbf{b}_2 - 2\mathbf{b}_1 + \mathbf{b}_0 \\
B(-1) &\leftarrow \mathbf{b}_2 - \mathbf{b}_1 + \mathbf{b}_0 \\
B(0) &\leftarrow \mathbf{b}_0 \\
B(1) &\leftarrow \mathbf{b}_2 + \mathbf{b}_1 + \mathbf{b}_0 \\
B(2) &\leftarrow 4\mathbf{b}_2 + 2\mathbf{b}_1 + \mathbf{b}_0
\end{aligned}
\right\} \text{Compute interpolants for } B
$$

$$
\left.
\begin{aligned}
C(-2) &\leftarrow \texttt{mult}(A(-2),\, B(-2)) \\
C(-1) &\leftarrow \texttt{mult}(A(-1),\, B(-1)) \\
C(0) &\leftarrow \texttt{mult}(A(0),\, B(0)) \\
C(1) &\leftarrow \texttt{mult}(A(1),\, B(1)) \\
C(2) &\leftarrow \texttt{mult}(A(2),\, B(2))
\end{aligned}
\right\} \text{Compute interpolants for } C
$$

$$
\left.
\begin{aligned}
\mathbf{c}_0 &\leftarrow C(0) \\
\mathbf{c}_1 &\leftarrow \frac{C(-2)}{12} - \frac{2C(-1)}{3} + \frac{2C(1)}{3} - \frac{C(2)}{12} \\
\mathbf{c}_2 &\leftarrow -\frac{C(-2)}{24} + \frac{2C(-1)}{3} - \frac{5C(0)}{4} + \frac{2C(1)}{3} - \frac{C(2)}{24} \\
\mathbf{c}_3 &\leftarrow -\frac{C(-2)}{12} + \frac{C(-1)}{6} - \frac{C(1)}{6} + \frac{C(2)}{12} \\
\mathbf{c}_4 &\leftarrow \frac{C(-2)}{24} - \frac{C(-1)}{6} + \frac{C(0)}{4} - \frac{C(1)}{6} + \frac{C(2)}{24}
\end{aligned}
\right\} \text{Interpolate } C
$$

return ab $= \mathbf{c}_4 2^{4k} + \mathbf{c}_3 2^{3k} + \mathbf{c}_2 2^{2k} + \mathbf{c}_1 2^k + \mathbf{c}_0$

The Karatsuba trick: slicing into 3 pieces

- How fast is this algorithm?

- We have replaced a multiplication of two n bit numbers with 5 multiplications of $n/3$ bit numbers with an overhead of additions, shifts and the similar, all doable in linear time $c\,n$;
- thus,

$$T(n) = 5T\left(\frac{n}{3}\right) + c\,n$$

- We now apply the Master Theorem:

  we have $a = 5, \;\; b = 3$, so we consider $n^{\log_b a} = n^{\log_3 5} \approx n^{1.465...}$
- Clearly, the first case of the MT applies and we get $T(n) = O(n^{\log_3 5}) < O(n^{1.47})$.

The Karatsuba trick: slicing into 3 pieces

- Recall that the original Karatsuba algorithm runs in time

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

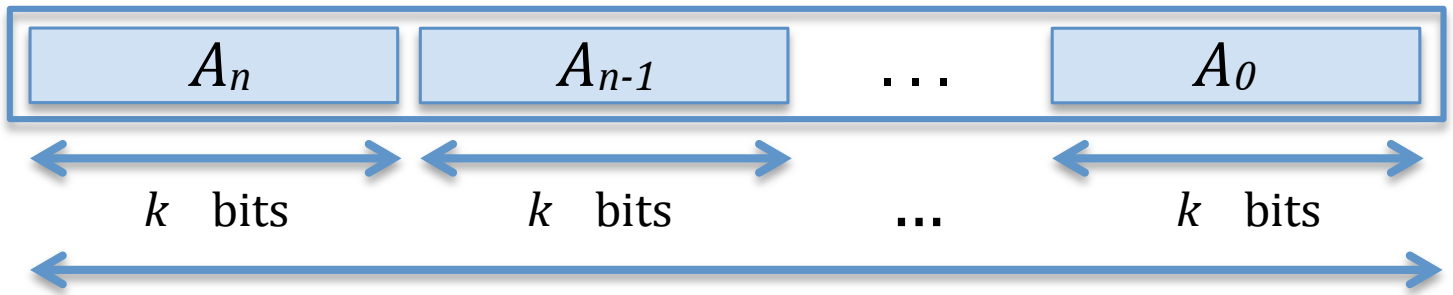
- Thus, we got a significantly faster algorithm.
- Then why not slice numbers **a** and **b** into even larger number of slices? Maybe we can get even faster algorithm?
- The answer is, in a sense, BOTH yes and no, so lets see what happens if we slice numbers into $n + 1$ many equal slices...

Generalizing Karatsuba's algorithm

The general case—slicing the input numbers $\mathbf{a}$, $\mathbf{b}$ into $n+1$ many slices

- For simplicity, let $\mathbf{a}$, $\mathbf{b}$ have $(n+1)k$ bits; (k can be arbitrarily large)
- Slice $\mathbf{a}$, $\mathbf{b}$ into $n+1$ pieces each:

$$\begin{aligned}\mathbf{a} = &\quad \mathbf{a}_n 2^{kn} &+ &\quad \mathbf{a}_{n-1} 2^{k(n-1)} + &\cdots + &\quad \mathbf{a}_0 \\ \mathbf{b} = &\quad \mathbf{b}_n 2^{kn} &+ &\quad \mathbf{b}_{n-1} 2^{k(n-1)} + &\cdots + &\quad \mathbf{b}_0\end{aligned}$$



A divided into $n+1$ slices each slice k bits = $(n+1)\,k$ bits in total

- We form the naturally corresponding polynomials:

$$\begin{aligned}A(x) = &\quad \mathbf{a}_n x^n &+ &\quad \mathbf{a}_{n-1} x^{n-1} + &\cdots + &\quad \mathbf{a}_0 \\ B(x) = &\quad \mathbf{b}_n x^n &+ &\quad \mathbf{b}_{n-1} x^{n-1} + &\cdots + &\quad \mathbf{b}_0\end{aligned}$$

Generalizing Karatsuba's algorithm

- As before, we have:
  $\mathbf{a} = A(2^k)$; $\mathbf{b} = B(2^k)$; $\mathbf{ab} = A(2^k)B(2^k) = (A(x) \cdot B(x))\,|_{x=2^k}$
- Since
$$\mathbf{ab} = (A(x) \cdot B(x))\,|_{x=2^k}$$
we adopt the following strategy:
  - we will first figure out how to multiply polynomials fast to obtain
$$C(x) = A(x) \cdot B(x);$$
  - then we evaluate $C(2^k)$.
- Note that $C(x) = A(x) \cdot B(x)$ is of degree $2n$:
$$C(x) = \sum_{j=0}^{2n} \mathbf{c}_j x^j$$

Generalizing Karatsuba's algorithm

- Example:

$$(\mathbf{a}_3x^3+\mathbf{a}_2x^2+\mathbf{a}_1x+\mathbf{a}_0)(\mathbf{b}_3x^3+\mathbf{b}_2x^2+\mathbf{b}_1x+\mathbf{b}_0) =$$
$$\mathbf{a}_3\mathbf{b}_3x^6+(\mathbf{a}_2\mathbf{b}_3+\mathbf{a}_3\mathbf{b}_2)x^5+(\mathbf{a}_1\mathbf{b}_3+\mathbf{a}_2\mathbf{b}_2+\mathbf{a}_3\mathbf{b}_1)x^4$$
$$+(\mathbf{a}_0\mathbf{b}_3+\mathbf{a}_1\mathbf{b}_2+\mathbf{a}_2\mathbf{b}_1+\mathbf{a}_3\mathbf{b}_0)x^3+(\mathbf{a}_0\mathbf{b}_2+\mathbf{a}_1\mathbf{b}_1+\mathbf{a}_2\mathbf{b}_0)x^2$$
$$+(\mathbf{a}_0\mathbf{b}_1+\mathbf{a}_1\mathbf{b}_0)x+\mathbf{a}_0\mathbf{b}_0$$

- In general: for

$$A(x)=\mathbf{a}_nx^n+\mathbf{a}_{n-1}x^{n-1}+\cdots+\mathbf{a}_0$$
$$B(x)=\mathbf{b}_nx^n+\mathbf{b}_{n-1}x^{n-1}+\cdots+\mathbf{b}_0$$

we have

$$A(x)\cdot B(x)=\sum_{j=0}^{2n}\left(\sum_{i+k=j}\mathbf{a}_i\mathbf{b}_k\right)x^j=\sum_{j=0}^{2n}\mathbf{c}_j\,x^j$$

- We need to find the coefficients $\mathbf{c}_j=\sum_{i+k=j}\mathbf{a}_i\mathbf{b}_k$ without performing $(n+1)^2$ many multiplications necessary to get all products of the form $\mathbf{a}_i\mathbf{b}_k$.

A VERY IMPORTANT DIGRESSION:

If you have two sequences $\vec{A} = (\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_{n-1}, \mathbf{a}_n)$ and $\vec{B} = (\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_{m-1}, \mathbf{b}_m)$, and if you form the two corresponding polynomials

$$A(x) = \mathbf{a}_n x^n + \mathbf{a}_{n-1} x^{n-1} + \ldots + \mathbf{a}_1 x + \mathbf{a}_0$$
$$B(x) = \mathbf{b}_m x^m + \mathbf{b}_{m-1} x^{m-1} + \ldots + \mathbf{b}_1 x + \mathbf{b}_0$$

and if you multiply these two polynomials to obtain their product

$$A(x) \cdot B(x) = \sum_{j=0}^{m+n} \left(\sum_{i+k=j} \mathbf{a}_i \mathbf{b}_k \right) x^j = \sum_{j=0}^{n+m} \mathbf{c}_j \, x^j$$

then the sequence $\vec{C} = (\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{n+m})$ of the coefficients of the product polynomial, with these coefficients given by

$$\mathbf{c}_j = \sum_{i+k=j} \mathbf{a}_i \mathbf{b}_k, \quad \text{for} \quad 0 \leq j \leq n+m,$$

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 means that the samples of the output sound are simply the coefficients of the product of two polynomials:
  1. polynomial $A(x)$ whose coefficients $\mathbf{a}_i$ are the samples of the input signal;
  2. polynomial $B(x)$ whose coefficients $\mathbf{b}_k$ are the samples of the so called impulse response of the filter (they depend of what kind of filtering you want to do).
- Convolutions are bread-and-butter of signal processing, and for that reason it 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 $A(x)$ of degree n is uniquely determined by its values at any $n+1$ distinct input values $x_0, x_1, \ldots, x_n$:

$$A(x) \leftrightarrow \{(x_0, A(x_0)), (x_1, A(x_1)), \ldots, (x_n, A(x_n))\}$$

- For $A(x) = \mathbf{a}_n x^n + \mathbf{a}_{n-1} x^{n-1} + \ldots + \mathbf{a}_0$, these values can be obtained via a matrix multiplication:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{pmatrix} = \begin{pmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_n) \end{pmatrix}. \quad (1)$$

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

Coefficient vs value representation of polynomials - ctd.

- Thus, if all x_i are all distinct, given any values $A(x_0), A(x_1), \ldots, A(x_n)$ the coefficients $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ of the polynomial $A(x)$ are uniquely determined:

$$\begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix}^{-1} \begin{pmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_n) \end{pmatrix} \tag{2}$$

- Equations (1) and (2) show how we can commute between:
  1. a representation of a polynomial $A(x)$ via its coefficients $\mathbf{a}_n, \mathbf{a}_{n-1}, \ldots, \mathbf{a}_0$, i.e. $A(x) = \mathbf{a}_n x^n + \ldots + \mathbf{a}_1 x + \mathbf{a}_0$
  2. a representation of a polynomial $A(x)$ via its values

$$A(x) \leftrightarrow \{(x_0, A(x_0)), (x_1, A(x_1)), \ldots, (x_n, A(x_n))\}$$

Coefficient vs value representation of polynomials- ctd.

- If we fix the inputs $x_0, x_1, \ldots, x_n$ then commuting between a representation of a polynomial $A(x)$ 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**:

$$\begin{pmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_n) \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{pmatrix};$$

$$\begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_n \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix}^{-1} \begin{pmatrix} A(x_0) \\ A(x_1) \\ \vdots \\ A(x_n) \end{pmatrix}.$$

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

Our strategy to multiply polynomials fast:

1. Given two polynomials of degree at most n,

$$A(x) = \mathbf{a}_n x^n + \ldots + \mathbf{a}_0; \qquad B(x) = \mathbf{b}_n x^n + \ldots + \mathbf{b}_0$$

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

$$A(x) \leftrightarrow \{(x_0, A(x_0)), (x_1, A(x_1)), \ldots, (x_{2n}, A(x_{2n}))\}$$

$$B(x) \leftrightarrow \{(x_0, B(x_0)), (x_1, B(x_1)), \ldots, (x_{2n}, B(x_{2n}))\}$$

   - **Note:** since the product of the two polynomials will be of degree $2n$ we need the values of $A(x)$ and $B(x)$ at $2n+1$ points, rather than just $n+1$ points!

2. Multiply these two polynomials point-wise, using $2n+1$ multiplications only.

$$A(x)B(x) \leftrightarrow \{(x_0, \underbrace{A(x_0)B(x_0)}_{C(x_0)}), (x_1, \underbrace{A(x_1)B(x_1)}_{C(x_1)}), \ldots, (x_{2n}, \underbrace{A(x_{2n})B(x_{2n})}_{C(x_{2n})})\}$$

3. Convert such value representation of $C(x) = A(x)B(x)$ back to coefficient form

$$C(x) = \mathbf{c}_{2n} x^{2n} + \mathbf{c}_{2n-1} x^{2n-1} + \ldots + \mathbf{c}_1 x + \mathbf{c}_0;$$

Fast multiplication of polynomials - continued

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

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

- So we find the values $A(m)$ and $B(m)$ for all m such that $-n \leq m \leq n$.
- Remember that $n+1$ is the number of slices we split the input numbers $\mathbf{a}$, $\mathbf{b}$.
- Multiplication of a large number with k bits by a constant integer d can be done in time linear in k because it is reducible to $d-1$ additions:

$$d \cdot \mathbf{a} = \underbrace{\mathbf{a} + \mathbf{a} + \ldots + \mathbf{a}}_{d}$$

- Thus, all the values

$$A(m) = \mathbf{a}_n m^n + \mathbf{a}_{n-1} m^{n-1} + \cdots + \mathbf{a}_0 : \quad -n \leq m \leq n,$$

$$B(m) = \mathbf{b}_n m^n + \mathbf{b}_{n-1} m^{n-1} + \cdots + \mathbf{b}_0 : \quad -n \leq m \leq n.$$

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

Fast multiplication of polynomials - ctd.

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

$$A(-n)B(-n),\ \ldots\ ,\ A(-1)B(-1),\quad A(0)B(0),\quad A(1)B(1),\ \ldots\ ,\ A(n)B(n)$$

- For $C(x) = A(x)B(x)$ these products are $2n+1$ many values of $C(x)$:

$$C(-n) = A(-n)B(-n),\ \ldots\ ,\ C(0) = A(0)B(0),\ \ldots\ ,\ C(n) = A(n)B(n)$$

- Let $\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{2n}$ be the coefficients of the product polynomial $C(x)$, i.e., let

$$C(x) = \mathbf{c}_{2n}x^{2n} + \mathbf{c}_{2n-1}x^{2n-1} + \cdots + \mathbf{c}_0,$$

- We now have:

$$\begin{aligned}
\mathbf{c}_{2n}(-n)^{2n} + \mathbf{c}_{2n-1}(-n)^{2n-1} + \cdots + \mathbf{c}_0 &= C(-n)\\
\mathbf{c}_{2n}(-(n-1))^{2n} + \mathbf{c}_{2n-1}(-(n-1))^{2n-1} + \cdots + \mathbf{c}_0 &= C(-(n-1))\\
&\vdots\\
\mathbf{c}_{2n}(n-1)^{2n} + \mathbf{c}_{2n-1}(n-1)^{2n-1} + \cdots + \mathbf{c}_0 &= C(n-1)\\
\mathbf{c}_{2n}n^{2n} + \mathbf{c}_{2n-1}n^{2n-1} + \cdots + \mathbf{c}_0 &= C(n)
\end{aligned}$$

Fast multiplication of polynomials - ctd.

- This is just a system of linear equations, that can be solved for $\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{2n}$:

$$\begin{pmatrix} 1 & -n & (-n)^2 & \ldots & (-n)^{2n} \\ 1 & -(n-1) & (-(n-1))^2 & \ldots & (-(n-1))^{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & n & n^2 & \ldots & n^{2n} \end{pmatrix} \begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{2n} \end{pmatrix} = \begin{pmatrix} C(-n) \\ C(-(n-1)) \\ \vdots \\ C(n) \end{pmatrix},$$

- i.e., we can obtain $\mathbf{c}_0, \mathbf{c}_1, \ldots, \mathbf{c}_{2n}$ as

$$\begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{2n} \end{pmatrix} = \begin{pmatrix} 1 & -n & (-n)^2 & \ldots & (-n)^{2n} \\ 1 & -(n-1) & (-(n-1))^2 & \ldots & (-(n-1))^{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & n & n^2 & \ldots & n^{2n} \end{pmatrix}^{-1} \begin{pmatrix} C(-n) \\ C(-(n-1)) \\ \vdots \\ C(n) \end{pmatrix}.$$

- But the inverse matrix also involves only constants depending on n only;
- Thus the coefficients $\mathbf{c}_i$ can be obtained in linear time.
- So here is the algorithm we have just described:

`mult(`n`, a, b)`

if $|\mathbf{a}| = |\mathbf{b}| = 1$ **then return ab**

obtain $n+1$ slices $\mathbf{a}_0, \mathbf{a}_1, \ldots, \mathbf{a}_n$ and $\mathbf{b}_0, \mathbf{b}_1, \ldots, \mathbf{b}_n$ such that

$\mathbf{a} = \mathbf{a}_n 2^{n\,k} + \mathbf{a}_{n-1} 2^{(n-1)\,k} + \ldots + \mathbf{a}_0$

$\mathbf{a} = \mathbf{b}_n 2^{n\,k} + \mathbf{b}_{n-1} 2^{(n-1)\,k} + \ldots + \mathbf{b}_0$

form polynomials

$$A(x) = \mathbf{a}_n x^n + \mathbf{a}_{n-1} x^{(n-1)} + \ldots + \mathbf{a}_0$$

$$B(x) = \mathbf{b}_n x^n + \mathbf{b}_{n-1} x^{(n-1)} + \ldots + \mathbf{b}_0$$

for $m = -n$ **to** $m = n$ **do**

compute $A(m)$ and $B(m)$

$C(m) \leftarrow$ `mult(`n`,` $A(m)$`,` $B(m)$`)`

compute $\mathbf{c}_0,\ \mathbf{c}_1, \ldots\ \mathbf{c}_{2n}$ via

$$\begin{pmatrix} \mathbf{c}_0 \\ \mathbf{c}_1 \\ \vdots \\ \mathbf{c}_{2n} \end{pmatrix} = \begin{pmatrix} 1 & -n & (-n)^2 & \ldots & (-n)^{2n} \\ 1 & -(n-1) & (-(n-1))^2 & \ldots & (-(n-1))^{2n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & n & n^2 & \ldots & n^{2n} \end{pmatrix}^{-1} \begin{pmatrix} C(-n) \\ C(-(n-1)) \\ \vdots \\ C(n) \end{pmatrix}.$$

form $C(x) = \mathbf{c}_{2n} x^{2n} + \ldots + \mathbf{c}_0$ and compute $C(2^k)$

return $C(2^k) = \mathbf{a} \cdot \mathbf{b}$

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 n but does NOT depend on k:

$$A(m) = \mathbf{a}_n m^n + \mathbf{a}_{n-1} m^{n-1} + \cdots + \mathbf{a}_0 : \quad -n \leq m \leq n.$$

  This is because each $\mathbf{a}_i$ is smaller than 2^k because each $\mathbf{a}_k$ has k bits; thus

$$|A(m)| < n^n(|\mathbf{a}_n| + |\mathbf{a}_{n-1}| + \cdots + |\mathbf{a}_0|) < n^n \times n \times 2^k$$

- Thus, we have reduced a multiplication of two $k(n+1)$ digit numbers to $2n+1$ multiplications of $k+s$ digit numbers plus a linear overhead (of additions, splitting the numbers, etc.)
- So we get the following recurrence for the complexity of $\textsc{Mult}(A, B)$:

$$T((n+1)k) = (2n+1)T(k+s) + c\,k$$

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

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

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

How fast is our algorithm?

- Since $\log_b a = \log_{n+1}(2n+1) > 1$, we can choose a small ε such that also $\log_b a - \varepsilon > 1$.
- Consequently, for such an ε we would have $f(N) = c/(n+1)\, N = O\left(N^{\log_b a - \varepsilon}\right)$.
- Thus, with $a = 2n+1$ and $b = n+1$ the first case of the Master Theorem applies;
- so we get:

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

- Note that

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

- Thus, by choosing a sufficiently large n, we can get a run time arbitrarily close to linear time!

- How large does n have to be, in order to to get an algorithm which runs in time $N^{1.1}$?

$$N^{1.1} = N^{1+\frac{1}{\log_2(n+1)}} \quad \rightarrow \quad \frac{1}{\log_2(n+1)} = \frac{1}{10} \quad \rightarrow \quad n+1 = 2^{10}$$

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

- We would have to evaluate polynomials $A(x)$ and $B(x)$ both of degree n at values up to n.
- However, $n = 2^{10}$, so evaluating $A(n) = \mathbf{a}_n n^n + \ldots + \mathbf{a}_0$ involves multiplication of $\mathbf{a}_n$ with $n^n = (2^{10})^{2^{10}} \approx 1.27 \times 10^{3079}$.
- Thus, while evaluations of $A(x)$ and $B(x)$ for $x = -n \ldots n$ can **theoretically** all be done in linear time, $T(n) = c\,n$, the constant c is absolutely **humongous**.
- Consequently, slicing the input numbers in more than just a few slices results in a hopelessly slow algorithm, despite the fact that the asymptotic bounds improve as we increase the number of slices!

Moral of the story

In practice, asymptotic estimates are useless if the size of the constants hidden by the O-notation are not estimated and found to be reasonably small!!!

- **Crucial question:** Are there numbers $x_0, x_1, \ldots, x_n$ such that the size of x_i^n 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 sequence of such values is called **the discrete Fourier transform (DFT)** of the sequence of the coefficients of the polynomial being evaluated.
- We will present a very fast algorithm for computing these values, called **the Fast Fourier Transform**, abbreviated as **FFT**.
- The Fast Fourier Transform is **the most executed algorithm today** 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.
- After we study the FFT we will have a guest lecture by a Dolby engineer to demonstrate to you some cool applications of FFT.

That's All, Folks!!