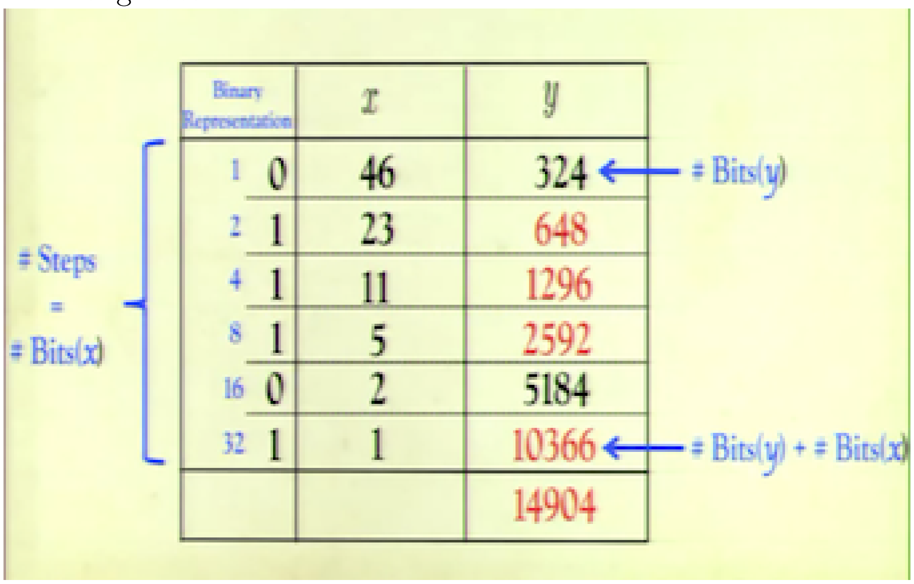
# Lecture 4: Recursive Algorithms: Fast Multiplication; Fast Matrix Multiplication

## 1 Multiplication

- Grade School Multiplication
  - Perform $n^2$ multiplications to multiply two n-digit numbers
    - $\rightarrow$ Long multiplication has running time $\Omega(n^2)$.
- Russian Peasant Multiplication
  Mult(x, y)
  **If** x = 1 **then** output y
  **If** x is odd **then** output y + Mult($[\frac{x}{2}]$,2y)
  **If** x is even **then** output Mult($\frac{x}{2}$,2y)

| Binary Representation | | $x$ | $y$ |
|---|---|---|---|
| 1 | 0 | 46 | 324 |
| 2 | 1 | 23 | 648 |
| 4 | 1 | 11 | 1296 |
| 8 | 1 | 5 | 2592 |
| 16 | 0 | 2 | 5184 |
| 32 | 1 | 1 | 10368 |
| | 46 | | 14904 |

- This method does work. Base case (x = 1) is verified. The step when x is even also works. The only tricky step is that x is odd when you divide x by 2, you actually divide x by 2 and minus a half, so that you need to add one y back.
- How long does it take?



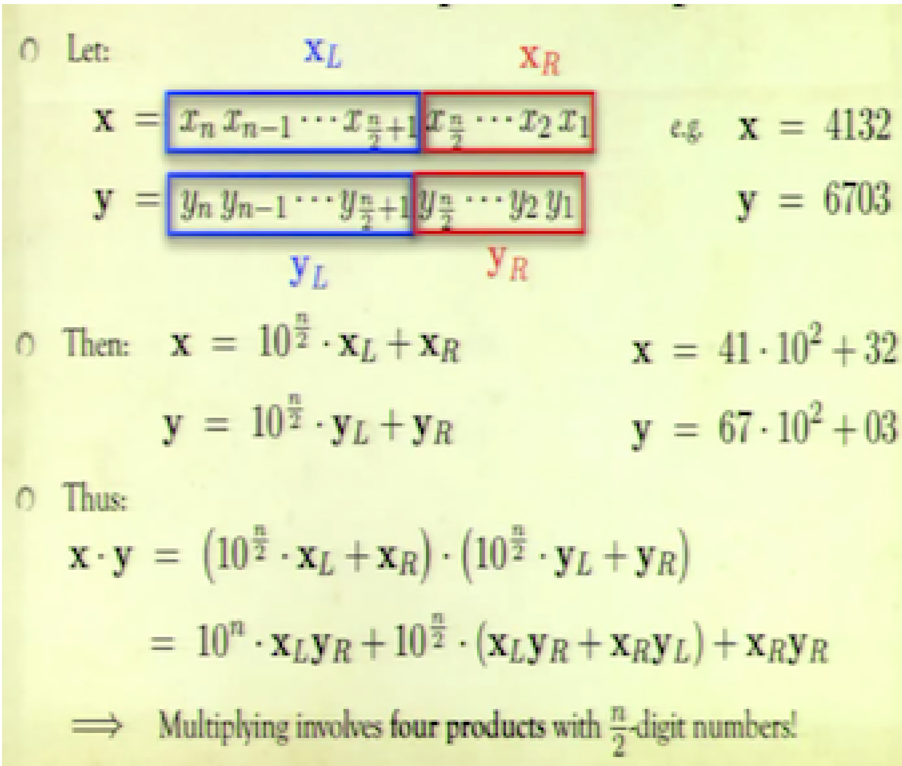
| Binary Representation | | x | y |
|---|---|---|---|
| 1 | 0 | 46 | 324 |
| 2 | 1 | 23 | 648 |
| 4 | 1 | 11 | 1296 |
| 8 | 1 | 5 | 2592 |
| 16 | 0 | 2 | 5184 |
| 32 | 1 | 1 | 10366 |
| | | | 14904 |



  → There are n iterations, so we add up to n numbers with at most 2n digits each.

  → Runtime = $O(n^2)$

## 2 Divide and Conquer Multiplication




- How long does this algorithm take?
  - The recursive formula for the running time is:
    $$T(\mathrm{n}) = 4 \cdot T(\tfrac{n}{2}) + O(\mathrm{n})$$
    Note: by padding some zeroes in front of the number, we can assume n is a power of 2.
  - Thus we have a = 4, b = 2, and d = 1.
  - This is Case 3 of the Master Theorem.
  - Runtime = $O(n^{\log_2 4}) = O(n^2)$

## 3 Gauss's Complex Number Multiplication

- Gauss considered the product of complex numbers:

$$(\mathrm{a} + \mathrm{b} \cdot i) \cdot (\mathrm{c} + \mathrm{d} \cdot i) = \mathrm{ac} - \mathrm{bd} + (\mathrm{bc} + \mathrm{ad}) \cdot i$$

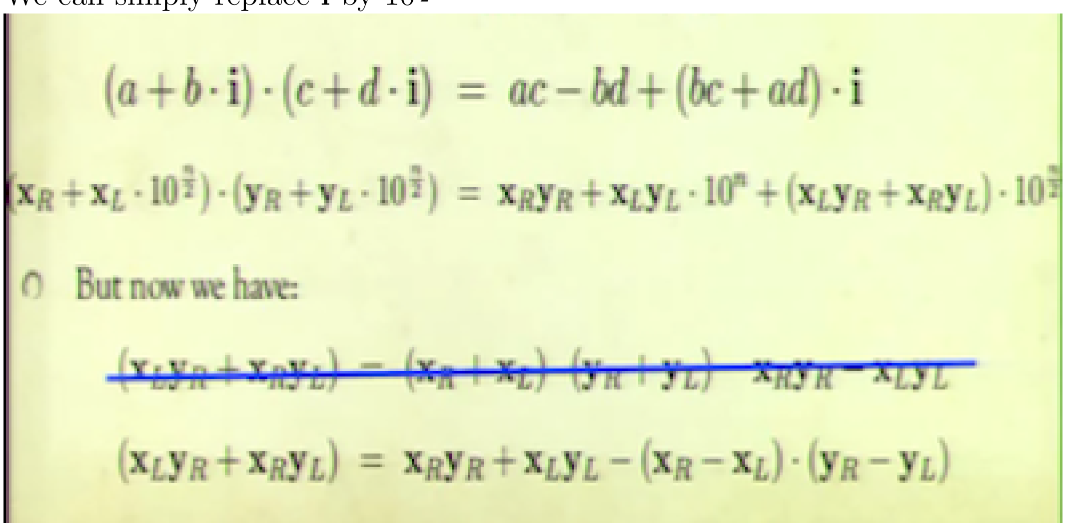
- This seems to require taking **four products**, but he observed that:

$$(\mathrm{bc} + \mathrm{ad}) = (\mathrm{a} + \mathrm{b}) \cdot (\mathrm{c} + \mathrm{d}) - \mathrm{ac} - \mathrm{bd}$$

- So we can calculate ac and bd then we only need to perform only one more product to find (bc + ad), namely (a + b) · (c + d).
    - Multiplying two complex numbers involves only **three products**! We got extra addition here, but addition is much cheaper than multiplication when we think about the running time.

# 4 Application of Gauss's Complex Number Multiplication

- We can simply replace **i** by $10^{\frac{n}{2}}$




    - Multiplying involves only **three products** with $\frac{n}{2}$ digit numbers!

- The recursive formula for the running time is:

$$T(\mathrm{n}) = 3 \cdot T(\tfrac{n}{2}) + O(\mathrm{n})$$

- Thus we have a = 3, b = 2, and d = 1.

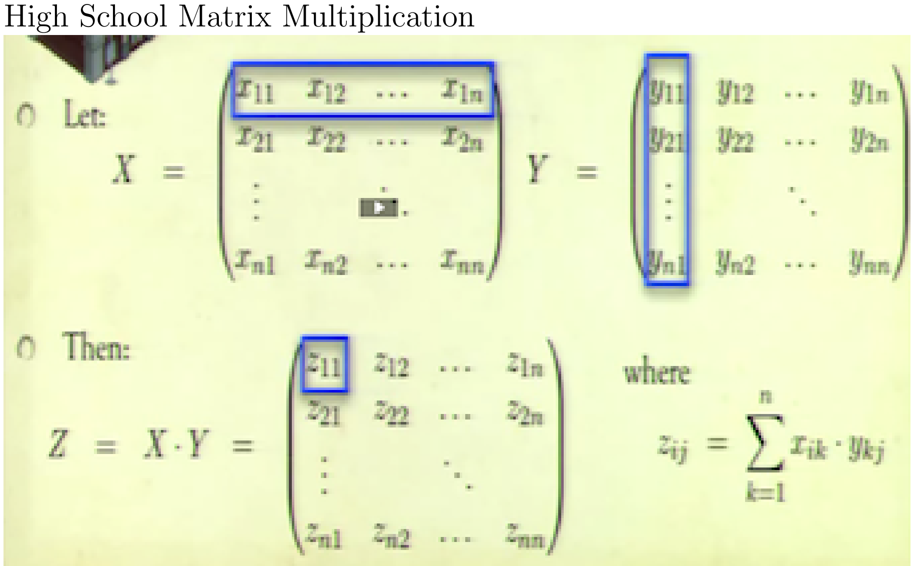
- This is Case 3 of the Master Theorem.
- Runtime $= O(n^{\log_2 3}) = O(n^{1.59})$
- So we can multiply two n-bit numbers using less than $n^2$ operations!

# 5 Fast Fourier Transforms

- We can multiply two n-bit numbers in time $O(\text{n} \cdot \log n)$ using a **Fast Fourier Transform**.
- More generally, FFTs can be used to multiply two **polynomial functions**.
  - This has a vast number of applications, for example in *image compression* and *signal processing*.
  - Indeed, it has been described by Strang as "the most important numerical algorithm of our lifetime."
- We might study FFTs later in the course.

# 6 Matrix Multiplication

- High School Matrix Multiplication




Then we perform n multiplications to calculate each entry of $Z$.

- Runtime = $\Omega(n^3)$

# 7 Divide and Conquer Matrix Multiplication

- We can also multiply matrices using **divide and conquer**.

  - Let: $X = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ $Y = \begin{pmatrix} E & F \\ G & H \end{pmatrix}$
  - Then: $Z = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$

- Multiplying involves **eight products** with $\frac{n}{2} \times \frac{n}{2}$ matrices!
- The recursive formula for the running time is:
$$T(\text{n}) = 8 \cdot T(\tfrac{n}{2}) + O(n^2)$$
- Thus we have a = 8, b = 2, and d = 2.
- This is Case 3 of the Master Theorem.
- Runtime = $O(n^{\log_2 8}) = O(n^3)$
- No improvement!

# 8 An Algebraic Trick for Matrix Multiplication

Let:

$$Z = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \cdot \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$$= \begin{pmatrix} AE+BG & AF+BH \\ CE+DG & CF+DH \end{pmatrix}$$

$$= \begin{pmatrix} S_1+S_2-S_4+S_6 & S_4+S_5 \\ S_6+S_7 & S_2-S_3+S_5-S_7 \end{pmatrix}$$

where
$$\begin{aligned} S_1 &= (B-D)\cdot(G+H) \\ S_2 &= (A+D)\cdot(E+H) \\ S_3 &= (A-C)\cdot(E+F) \\ S_4 &= (A+B)\cdot H \\ S_5 &= A\cdot(F-H) \\ S_6 &= D\cdot(G-E) \\ S_7 &= (C+D)\cdot E \end{aligned}$$

- Multiplying involves **seven products** with $\frac{n}{2} \times \frac{n}{2}$ matrices!
  Note: Even though we now have 18 additions, the runtime is still dominated by the number of multiplications. Having additions would not hurt.
- The recursive formula for the running time is:
  $$T(\text{n}) = 7 \cdot T(\tfrac{n}{2}) + O(n^2)$$
- Thus we have a = 7, b = 2, and d = 2.
- This is Case 3 of the Master Theorem.
- Runtime $= O(n^{\log_2 7}) = O(n^{2.81})$
- This divide and conquer matrix multiplication algorithm was designed by Strasser (1969). Since then faster algorithms (in theory) have been developed:
  - **Theorem**: There is a matrix multiplication algorithm that runs in time $O(n^{2.37})$
  - **Open Problem**: Is matrix multiplication $O(n^{2+\epsilon})$ time for any constant $\epsilon > 0$? Note: There is no way you can beat this because just to read the matrix x you have to do $x^2$ amount of work to read entries in x simply for y. Getting running time like this seems crazy but possible.

# 9 Fast Exponentiation

- Suppose we want to compute $x^n$
  - The slow way $x \cdot x \cdots x$ requires n-1 multiplications.
  - A faster way is to use the fact that $x^n = x^{[\frac{n}{2}]} \cdot x^{[\frac{n}{2}]}$
- The latter method gives the following **recursive algorithm**.

Fast Exponentiation
**FastExp**(x,n)
**If** n = 1 **then** output x
**Else**
  **If** n is even **then** output $\textbf{FastExp}(x, [\frac{n}{2}])^2$
  **If** n is odd **then** output $\mathbf{x} \cdot \textbf{FastExp}(x, [\frac{n}{2}])^2$

- The number of multiplications used in Fast Exponentiation satisfies:
  $T(\text{n}) \leq T([\frac{n}{2}]) + 2 \Rightarrow T(\text{n}) = T(\frac{n}{2}) + O(1)$
- Thus we have a = 1, b = 2, and d = 0.
- This is Case 2 of the Master Theorem.
- Runtime = $O(n^0 \cdot logn) = O(logn)$