DIVIDE AND CONQUER

Divide and Conquer

- Break a problem into similar subproblems
- Solve each subproblem recursively
- Combine

Above its weight class

- Divide and conquer is a very simple idea
- But it has far more than its share of the *miraculous algorithms*
- Examples
  - Strassen Matrix Multiplication
  - Karatsuba multiplication
  - Fast Fourier Transform
  - Linear time select

Multiplying Binomials

- if you want to multiply two binomials
- $(ax + b)(cx + d) = acx^2 + adx + bcx + bd$
- It requires ? 4 multiplications. ac, ad, bc, bd

Multiplying Binomials

- if you want to multiply two binomials
- $(ax + b)(cx + d) = acx^2 + (ad + bc)x + bd$
- It requires 4 multiplications. ac, ad, bc, bd

- If we assume that addition is cheap (has short runtime.) Then we can improve this by only doing 3 multiplications:

$$ac, bd, (a + b)(c + d)$$

Multiplying Binomials

- Reducing the number of multiplications from 4 to 3 may not seem very impressive when calculating asymptotics.
- If this was only a part of a bigger algorithm, it may be an improvement.

Multiplying Binary numbers

$$
\begin{array}{cccccccc l}
 & & & & 1 & 1 & 0 & 1 & \\
 & & & \times & 1 & 0 & 1 & 1 & \\
\hline
 & & & & 1 & 1 & 0 & 1 & \text{(1101 times 1)} \\
 & & & 1 & 1 & 0 & 1 & & \text{(1101 times 1, shifted once)} \\
 & & 0 & 0 & 0 & 0 & & & \text{(1101 times 0, shifted twice)} \\
+ & 1 & 1 & 0 & 1 & & & & \text{(1101 times 1, shifted thrice)} \\
\hline
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & \text{(binary 143)}
\end{array}
$$

Divide and conquer multiply

- Say we want to multiply 10100110 and 10110011

- How can we divide the problem into sub-problems?
- Remember, we want much smaller sub-problems

Multiplying large binary numbers

- 10100110 = 166= 1010 * 2^4 + 0110 = 10*16 + 6
- 10110011 = 179= 1011*$2^4 + 0011 =$ 11*16 + 3
- 10100110*10110011 = (10*16+6)(11*16+3)= 110*256 + 6*11*16 + 3*10*16 + 3*6

Multiplying Binary numbers (DC)

- Suppose we want to multiply two n-bit numbers together where n is a power of 2.
- One way we can do this is by splitting each number into their left and right halves which are each n/2 bits long

- x= x_L x_R
- y= y_L y_R

Multiplying Binary numbers (DC)

- Suppose we want to multiply two n-bit numbers together where n is a power of 2.
- One way we can do this is by splitting each number into their left and right halves which are each n/2 bits long

- $x = 2^{n/2}x_L + x_R$
- $y = 2^{n/2}y_L + y_R$

Multiplying Binary numbers (DC)

$$x = 2^{n/2}x_L + x_R$$
$$y = 2^{n/2}y_L + y_R$$

- $xy = \left(2^{\frac{n}{2}}x_L + x_R\right)\left(2^{\frac{n}{2}}y_L + y_R\right)$
- $xy = 2^n x_L y_L + 2^{\frac{n}{2}}(x_L y_R + x_R y_L) + x_R y_R$

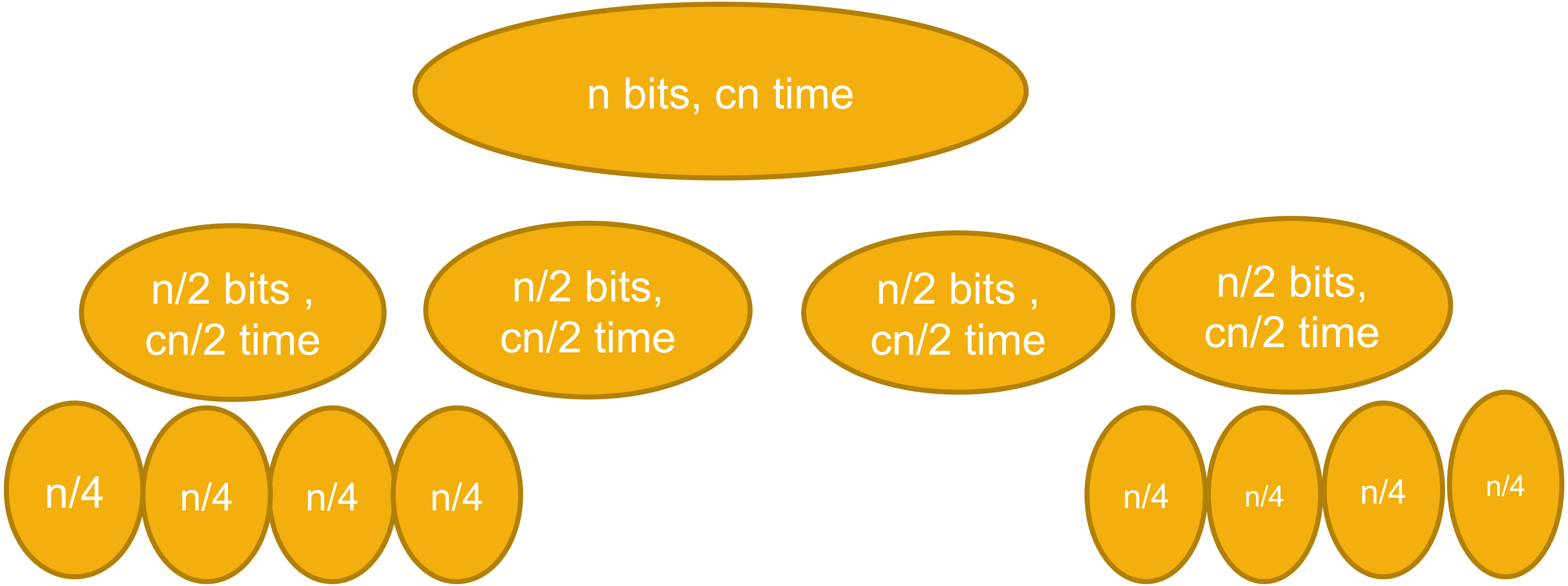
Algorithm multiply

- function **multiply** (x,y):
- *Input*: n-bit integers x and y
- *Output*: the product xy
  - If n=1: return xy
  - x_L, x_R and y_L, y_R are the left-most and right-most n/2 bits of x and y, respectively.
  - $P_1 = \mathbf{multiply}(x_L, y_L)$
  - $P_2 = \mathbf{multiply}(x_L, y_R)$
  - $P_3 = \mathbf{multiply}(x_R, y_L)$
  - $P_4 = \mathbf{multiply}(x_R, y_R)$
  - return $(P_1 * 2^n + (P_2 + P_3) * 2^{\frac{n}{2}} + P_4)$

Algorithm

- Runtime analysis:
  - Let T(n) be the runtime of the multiply algorithm.
  - Then $T(n) = 4T\left(\frac{n}{2}\right) + O(n)$

Total time


n bits, cn time
n/2 bits ,
cn/2 time
n/2 bits,
cn/2 time
n/2 bits ,
cn/2 time
n/2 bits,
cn/2 time
n/4
n/4
n/4
n/4
n/4
n/4
n/4
n/4

Total

- One top level : cn
- 4   depth 1: cn/2 *4
- 16 depth 2: cn/4 * 16
- 64 depth 3: cn/8 * 64

….

- $4^t \; depth \; t : \frac{cn}{2^t} * 4^t = cn * 2^t$

….

- Max level : t= log n, (cn/$2^{\log n}$)*$4^{logn} = c * 2^{logn} * 2^{logn} = cn^2$

Total time

- cn (1+ 2 +4 +8+… $2^{logn}) = O(cn^2)$
- Because in a geometric series with ratio other than 1, largest term dominates order.

Multiplication

Andrey Kolmogorov 1903 - 1987

Anatoly Karatsuba 1937 - 2008

Insight: replace one (of the 4) multiplications by (linear time) subtraction

Algorithm multiply KS

- function **multiplyKS** (x,y)
- *Input*: n-bit integers x and y
- *Output*: the product xy
  - If n=1: return xy
  - x_L, x_R and y_L, y_R are the left-most and right-most n/2 bits of x and y, respectively.
  - $R_1 = \mathbf{multiplyKS}(x_L, y_L)$
  - $R_2 = \mathbf{multiplyKS}(x_R, y_R)$
  - $R_3 = \mathbf{multiplyKS}\big((x_L + x_R), (y_L + y_R)\big)$
  - return $(R_1 * 2^n + (R_3 - R_1 - R_2) * 2^{\frac{n}{2}} + R_2)$

Correctness multiply KS

- Correctness: by strong induction on n, the number of bits of x and y.
- Base Case: $n = 1$ then return xy (could make a table of possibilities.)
- Inductive hypothesis:

Correctness multiply KS

- Correctness: by strong induction on n, the number of bits of x and y.
- Base Case: $n = 1$ then return xy (could make a table of possibilities.)
- Inductive hypothesis: For some $n > 1$, assume that **multiplyKS**(x,y) returns the correct product xy whenever x has k digits and y has k digits for any $1 \leq k < n$.

- Then by the IH: $R_1 = x_L y_L,\ R_2 = x_R y_R,\ R_3 = (x_L + x_R)(y_L + y_R)$

Correctness multiply KS

- Then by the IH:
- $R_1 = x_L y_L,\ R_2 = x_R y_R,\ R_3 = (x_L + x_R)(y_L + y_R) = x_L y_L + x_R y_R + x_L y_R + x_R y_L$

- And the algorithm returns: $R_1 * 2^n + (R_3 - R_1 - R_2) * 2^{\frac{n}{2}} + R_2$

$R_1 * 2^n + (R_3 - R_1 - R_2) * 2^{\frac{n}{2}} + R_2 =$

$x_L y_L * 2^n + (x_L y_R + x_R y_L) * 2^{\frac{n}{2}} + x_R y_R =$

$\left(x_L * 2^{\frac{n}{2}} + x_R\right)\left(y_L * 2^{\frac{n}{2}} + y_R\right) =$

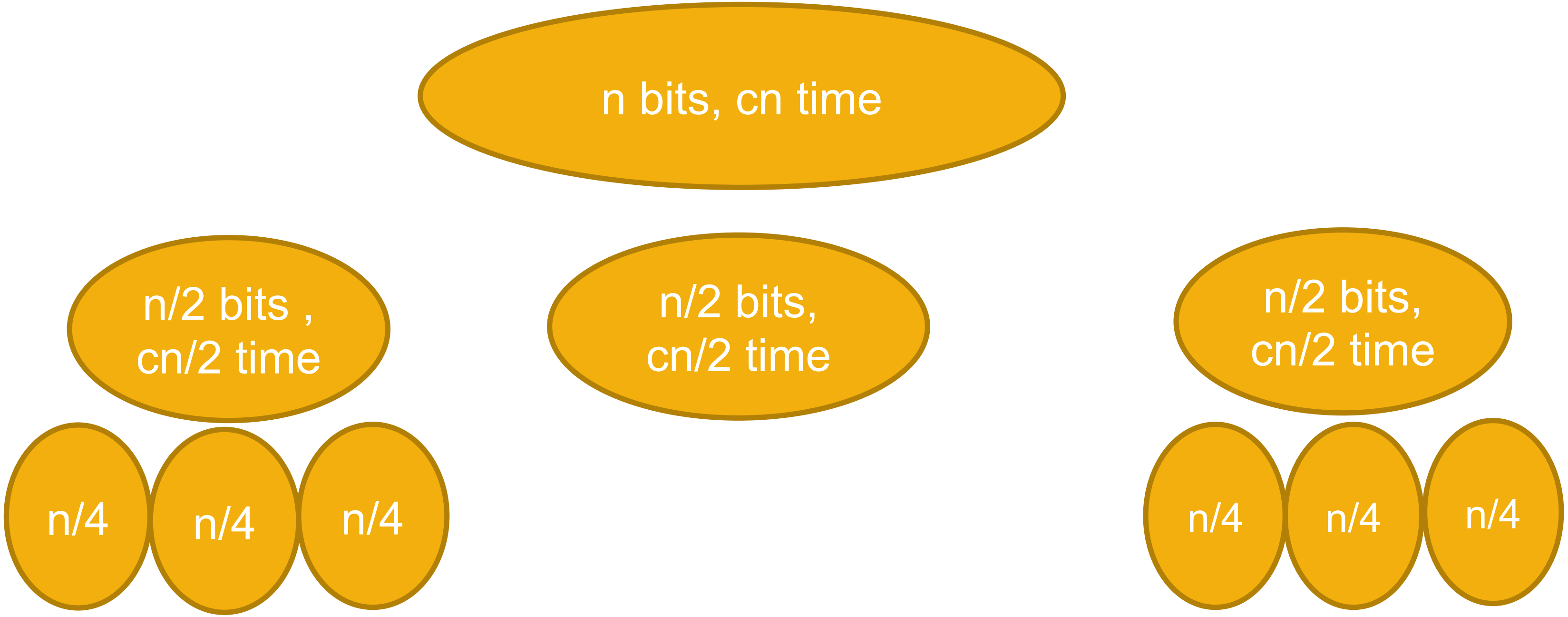
xy

Algorithm multiplyKS

- Runtime
- Let T(n) be the runtime of the multiply algorithm.

- Then
- $T(n) = 3T\left(\frac{n}{2}\right) + O(n)$

Total time


n bits, cn time
n/2 bits ,
cn/2 time
n/2 bits,
cn/2 time
n/2 bits,
cn/2 time
n/4
n/4
n/4
n/4
n/4
n/4

3 vs 4

- Since we are pruning the tree recursively, replacing 4 recursive calls instead of 3 reduces the size of the tree more than a constant factor.

Total

- One top level : cn
- 4   depth 1: cn/2 *3
- 16 depth 2: cn/4 * 9
- 64 depth 3: cn/8 * 27

….

- $4^t \; depth \; t : \frac{cn}{2^t} * 3^t = cn * (1.5)^t$

….

- Max level : t= log n

Total time

- cn (1+ 1.5 +2.25 +…$(1.5)^{logn}) = O\left(3^{logn}\right)$
- Because in a geometric series with ratio other than 1, largest term dominates order.

- But what is 3^{logn}?

Simplifying

- $3^{\log n} = \left(2^{log3}\right)^{logn} = 2^{\{logn*log3\}} = \left(2^{\{logn\}}\right)^{log3} = n^{log3} = n^{\{1.58\dots\}}$

- So total time is $O\left(n^{log3}\right)$

Master Theorem

- How do you solve a recurrence of the form

$$T(n) = aT\left(\frac{n}{b}\right) + O\left(n^d\right)$$

We will use the master theorem.