CS 5112
- Divide & conquer

CORNELL TECH

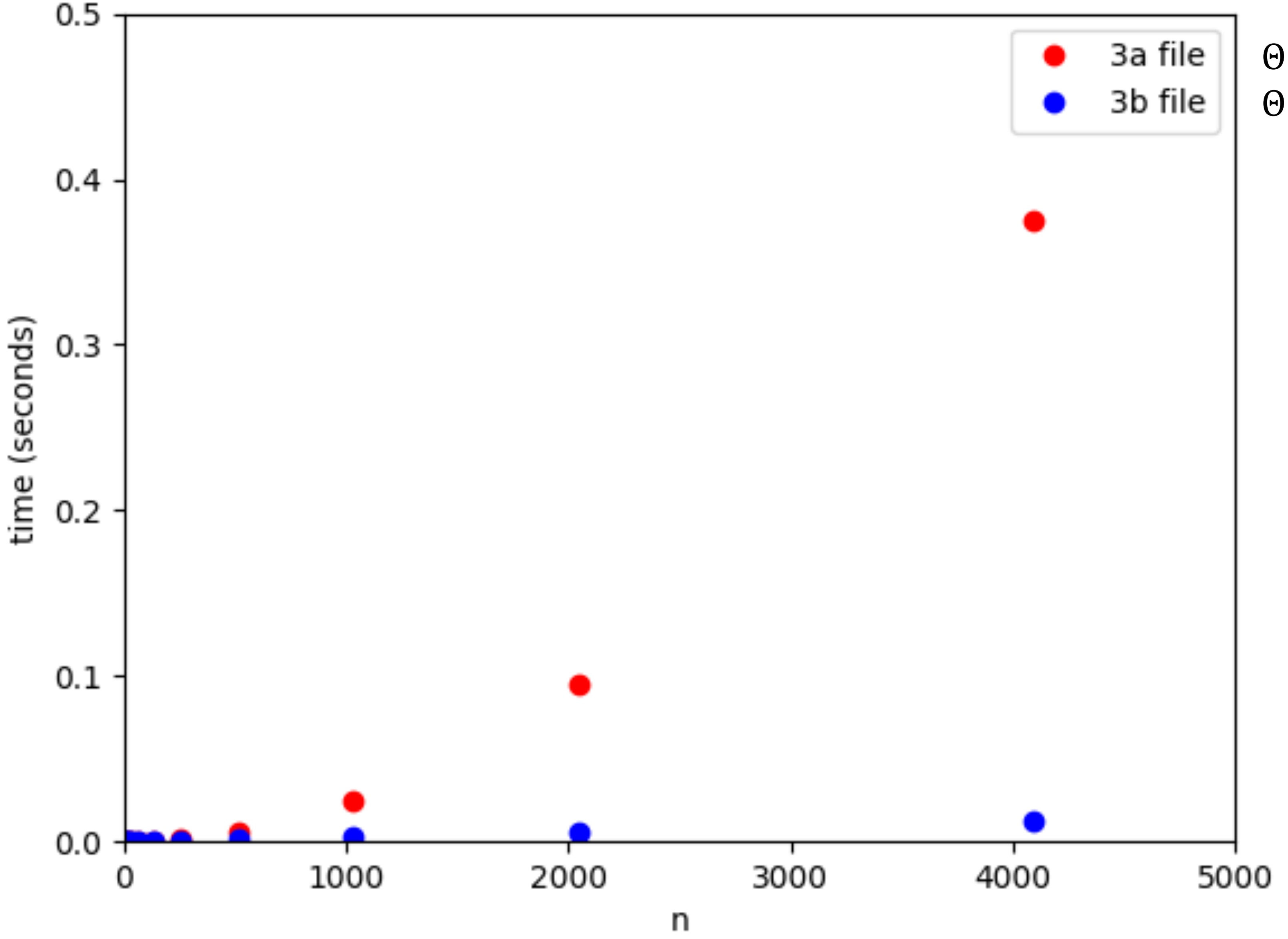
HOME OF THE JACOBS INSTITUTE

Divide and conquer

<u>Class of algorithmic techniques in which:</u>

1. Divide input into sub-problems (most often, two halves each of size n/2)
2. Solve problem on each sub-problem
3. Carefully merge solutions to solve your problem (usually, $O(n)$ time)

Often moves from time $O(n^2)$ solution using brute-force to $O(n \log n)$



$\Theta(n^2)$

$\Theta(n \log n)$

Credit: Carolina Ortega Perez

Sorting a list *L*

We've assumed several times that you can sort a list in *O(n log n)* time
But how can this work?

<u>Merge-Sort(*L*)</u>

If |*L*| = 1 then Return *L*

Split *L* into two halves *A*, *B*
A <- Merge-Sort(*A*)
B <- Merge-Sort(*B*)
L <- Merge(*A*,*B*)

Return *L*

Sorting a list *L*

We've assumed several times that you can sort a list in *O(n log n)* time
But how can this work?

<u>Merge-Sort(*L*)</u>

If |*L*| = 1 then Return *L*

Split *L* into two halves *A*, *B*
A <- Merge-Sort(*A*)
B <- Merge-Sort(*B*)
L <- Merge(*A*,*B*)

Return *L*

Run time? O(n log n)

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + cn \\
&\leq 2c(n/2)\log_2(n/2) + cn \\
&= cn\left((\log_2 n) - 1\right) + cn \\
&= (cn\log_2 n) - cn + cn \\
&= cn\log_2 n
\end{aligned}
$$

Another D&C example: counting inversions

Consider list of distinct numbers $L = x_1, \ldots, x_n$

An inversion is a pair of indices $i < j$ such that $x_i > x_j$

Count the number of inversions in a list L

Another D&C example: counting inversions

Consider list of distinct numbers $L = x_1, \ldots, x_n$

An inversion is a pair of indices $i < j$ such that $x_i > x_j$

Count the number of inversions in a list L

Another D&C example: counting inversions

Consider list of distinct numbers $L = x_1, \ldots, x_n$
An inversion is a pair of indices $i < j$ such that $x_i > x_j$
Count the number of inversions in a list L

<u>Sort-and-Count (L)</u>

If $|L| = 1$ then Return 0
Divide L into two halves A, B
    A has first ceil(n/2) elements
    B has last floor(n/2) elements
(r_A, A) <- Sort-and-Count(A)
(r_B, B) <- Sort-and-Count(B)
(r, L) <- Merge-and-Count(A,B)

Return $(r + r_A + r_A, L)$

Another D&C example: counting inversions

Consider list of distinct numbers $L = x_1, \ldots, x_n$

An inversion is a pair of indices $i < j$ such that $x_i > x_j$

Count the number of inversions in a list L

<u>Sort-and-Count (L)</u>

If $|L| = 1$ then Return 0

Divide L into two halves A, B

    A has first ceil(n/2) elements

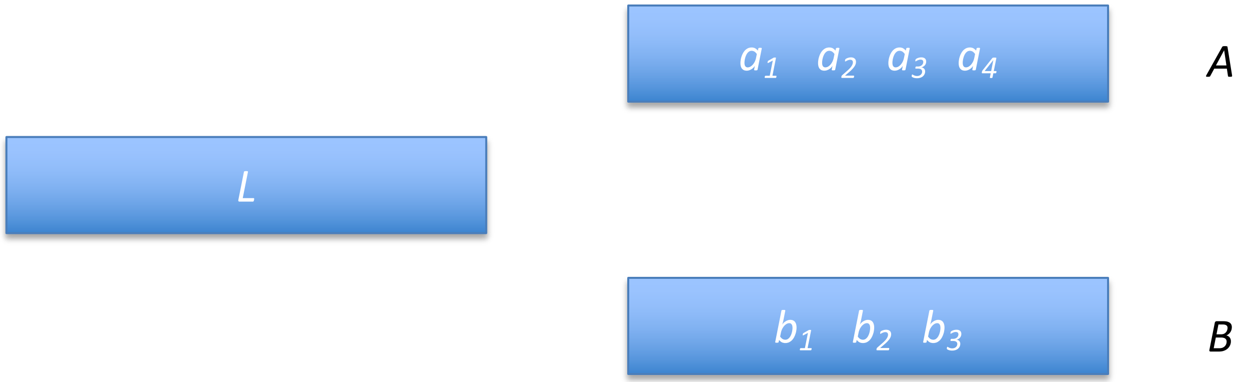
    B has last floor(n/2) elements

(r_A, A) <- Sort-and-Count(A)

(r_B, B) <- Sort-and-Count(B)

(r, L) <- Merge-and-Count(A,B)

Return $(r + r_A + r_A, L)$



Another D&C example: counting inversions

Consider list of distinct numbers $L = x_1, \ldots, x_n$

An inversion is a pair of indices $i < j$ such that $x_i > x_j$

Count the number of inversions in a list L

<u>Sort-and-Count (L)</u>

If $|L| = 1$ then Return 0

Divide L into two halves A, B

    A has first ceil(n/2) elements

    B has last floor(n/2) elements

(r_A, A) <- Sort-and-Count(A)

(r_B, B) <- Sort-and-Count(B)

(r, L) <- Merge-and-Count(A,B)

Return $(r + r_A + r_A, L)$

<u>Merge-and-Count(A,B)</u>

L <- empty list

Count <- 0 ; i <- 1 ; j <- 1

While $i \leq |A|$ and $j \leq |B|$

    If $a_i > b_j$ then

        Append b_j to L

        Count <- Count + ($|A|$ - i)

        j <- j + 1

    else

        Append a_i to L

        i <- i + 1

Return (Count , L)

Run time?

$T(n) = 2T(n/2) + cn$

for some constant c

Thus: O(n log n)

Divide-and-conquer so far

- Merge-Sort algorithm for sorting
  - Trick is that we can merge two sorted lists efficiently (linear)
- Sort-and-Count algorithm for counting inversions
  - Trick is that given two sorted lists we can count inversions efficiently (linear)
- Binary search sometimes called divide-and-conquer

Integer addition (in binary!)

Given two n-bit integers x, y compute $x + y$

Integer multiplication (in binary!)

Given two n-bit integers x, y compute $x \cdot y$

Integer multiplication (in binary!)

Given two n-bit integers *x*, *y* compute $x \cdot y$

Integer addition via grade-school algorithm: $O(n^2)$

Conjecture. [Kolmorogov 1956]
Grade-school algorithm for multiplication is **optimal**

Disproved by [Karatsuba 1960] who showed faster algorithm: $O(n^{1.59})$

Best theoretical result to date:
[Harvey, van Der Hoeven 2019] show $O(n \log n)$

Integer multiplication: D&C try #1

Given two n-bit integers x, y compute $x \cdot y$

Integer multiplication: D&C try #1

Given two n-bit integers x, y compute $x \cdot y$

<u>Multiply1(x,y,n)</u>

If $|L| = 1$ then Return $x \cdot y$
$m = n/2$
$x_1 = x / 2^m$; $x_0 = x \bmod 2^m$
$y_1 = y / 2^m$; $y_0 = y \bmod 2^m$
a <- Multiply1(x_1, y_1, m)
b <- Multiply1(x_1, y_0, m)
c <- Multiply1(x_0, y_1, m)
d <- Multiply1(x_0, y_0, m)

Return $a \cdot 2^n + (b + c) \cdot 2^{n/2} + d$

Split each of x, y into two halves, each of n/2 bits:

$$x = x_1 \cdot 2^{n/2} + x_0$$
$$y = y_1 \cdot 2^{n/2} + y_0$$

Substitute into equation $x \cdot y$

$$\begin{aligned} x \cdot y &= (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \\ &= x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0 \end{aligned}$$

Now we can compute in 4 n/2-bit multiplications!!

Integer multiplication: D&C try #1

Given two n-bit integers x, y compute $x \cdot y$

Multiply1(x,y,n)

If $|L| = 1$ then Return $x \cdot y$
$m = n/2$
$x_1 = x \,/\, 2^m$; $x_0 = x \bmod 2^m$
$y_1 = y \,/\, 2^m$; $y_0 = y \bmod 2^m$
a <- Multiply1(x_1, y_1, m)
b <- Multiply1(x_1, y_0, m)
c <- Multiply1(x_0, y_1, m)
d <- Multiply1(x_0, y_0, m)

Return $a \cdot 2^n + (b + c) \cdot 2^{n/2} + d$

Does this work to beat $O(n^2)$?

Solving the recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4 \cdot T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Master theorem

Theorem. Let $a \geq 1$, $b \geq 2$, and $c \geq 0$ and suppose that $T(n)$ is a function on the non-negative integers that satisfies the recurrence

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta(n^c)$$

with $T(0) = 0$ and $T(1) = \Theta(1)$. Then

$$\begin{aligned}
&\text{Case 1: If } c > \log_b a, \text{ then } T(n) = \Theta(n^c) \\
&\text{Case 2: If } c = \log_b a, \text{ then } T(n) = \Theta(n^c \log n) \\
&\text{Case 3: If } c < \log_b a, \text{ then } T(n) = \Theta(n^{\log_b a})
\end{aligned}$$

Example from Merge-Sort: a = 2, b = 2, c = 1

Back to our recurrence

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 4 \cdot T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Case 1: If $c > \log_b a$, then $T(n) = \Theta(n^c)$
Case 2: If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$
Case 3: If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$

What is run time of our first attempt at multiplication?

Integer multiplication: D&C try #1

Given two n-bit integers x, y compute $x \cdot y$

<u>Multiply1(x,y,n)</u>

If $|L| = 1$ then Return $x \cdot y$
$m = n/2$
$x_1 = x / 2^m$; $x_0 = x \bmod 2^m$
$y_1 = y / 2^m$; $y_0 = y \bmod 2^m$
a <- Multiply1(x_1, y_1, m)
b <- Multiply1(x_1, y_0, m)
c <- Multiply1(x_0, y_1, m)
d <- Multiply1(x_0, y_0, m)

Return $a \cdot 2^n + (b + c) \cdot 2^{n/2} + d$

Split each of x, y into two halves, each of n/2 bits:

$$x = x_1 \cdot 2^{n/2} + x_0 \qquad y = y_1 \cdot 2^{n/2} + y_0$$

$$\begin{aligned} x \cdot y &= (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \\ &= x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0 \end{aligned}$$

Integer multiplication: D&C try #1

Given two n-bit integers *x*, *y* compute $x \cdot y$

Multiply1(*x*,*y*,*n*)

If $|L| = 1$ then Return $x \cdot y$
$m = n/2$
$x_1 = x / 2^m$; $x_0 = x \bmod 2^m$
$y_1 = y / 2^m$; $y_0 = y \bmod 2^m$
a <- Multiply1(x_1, y_1, m)
b <- Multiply1(x_1, y_0, m)
c <- Multiply1(x_0, y_1, m)
d <- Multiply1(x_0, y_0, m)

Return $a \cdot 2^n + (b + c) \cdot 2^{n/2} + d$

Split each of *x*, *y* into two halves, each of n/2 bits:

$$x = x_1 \cdot 2^{n/2} + x_0 \qquad y = y_1 \cdot 2^{n/2} + y_0$$

$$\begin{aligned} x \cdot y &= (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \\ &= x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0 \end{aligned}$$

$$\begin{aligned} z_1 &= x_1 y_0 + x_0 y_1 \\ &= x_1 y_0 + x_0 y_1 + x_1 y_1 - x_1 y_1 + x_0 y_0 - x_0 y_0 \\ &= (x_1 + x_0) y_0 + (x_0 + x_1) y_1 - x_1 y_1 - x_0 y_0 \\ &= (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 \end{aligned}$$

Integer multiplication: D&C try #1

Given two n-bit integers *x*, *y* compute $x \cdot y$

Karatsuba(*x,y,n*)

If $|L| = 1$ then Return $x \cdot y$
$m = n/2$
$x_1 = x / 2^m$; $x_0 = x \bmod 2^m$
$y_1 = y / 2^m$; $y_0 = y \bmod 2^m$
a <- Karatsuba(x_1, y_1, m)
b <- Karatsuba($x_1 + x_0$, $y_1 + y_0$, m)
d <- Karatsuba(x_0, y_0, m)

Return $a \cdot 2^n + (b - a - d) \cdot 2^{n/2} + d$

Split each of *x*, *y* into two halves, each of n/2 bits:

$$x = x_1 \cdot 2^{n/2} + x_0 \qquad y = y_1 \cdot 2^{n/2} + y_0$$

$$\begin{aligned} x \cdot y &= (x_1 \cdot 2^{n/2} + x_0)(y_1 \cdot 2^{n/2} + y_0) \\ &= x_1 y_1 \cdot 2^n + (x_1 y_0 + x_0 y_1) \cdot 2^{n/2} + x_0 y_0 \end{aligned}$$

$$\begin{aligned} z_1 &= x_1 y_0 + x_0 y_1 \\ &= x_1 y_0 + x_0 y_1 + x_1 y_1 - x_1 y_1 + x_0 y_0 - x_0 y_0 \\ &= (x_1 + x_0) y_0 + (x_0 + x_1) y_1 - x_1 y_1 - x_0 y_0 \\ &= (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 \end{aligned}$$

Integer multiplication: D&C try #1

Given two n-bit integers *x*, *y* compute $x \cdot y$

Karatsuba(*x,y,n*)

If $|L| = 1$ then Return $x \cdot y$
$m = n/2$
$x_1 = x / 2^m$; $x_0 = x \bmod 2^m$
$y_1 = y / 2^m$; $y_0 = y \bmod 2^m$
a <- Karatsuba(x_1, y_1, m)
b <- Karatsuba($x_1 + x_0$, $y_1 + y_0$, m)
d <- Karatsuba(x_0, y_0, m)

Return $a \cdot 2^n + (b - a - d) \cdot 2^{n/2} + d$

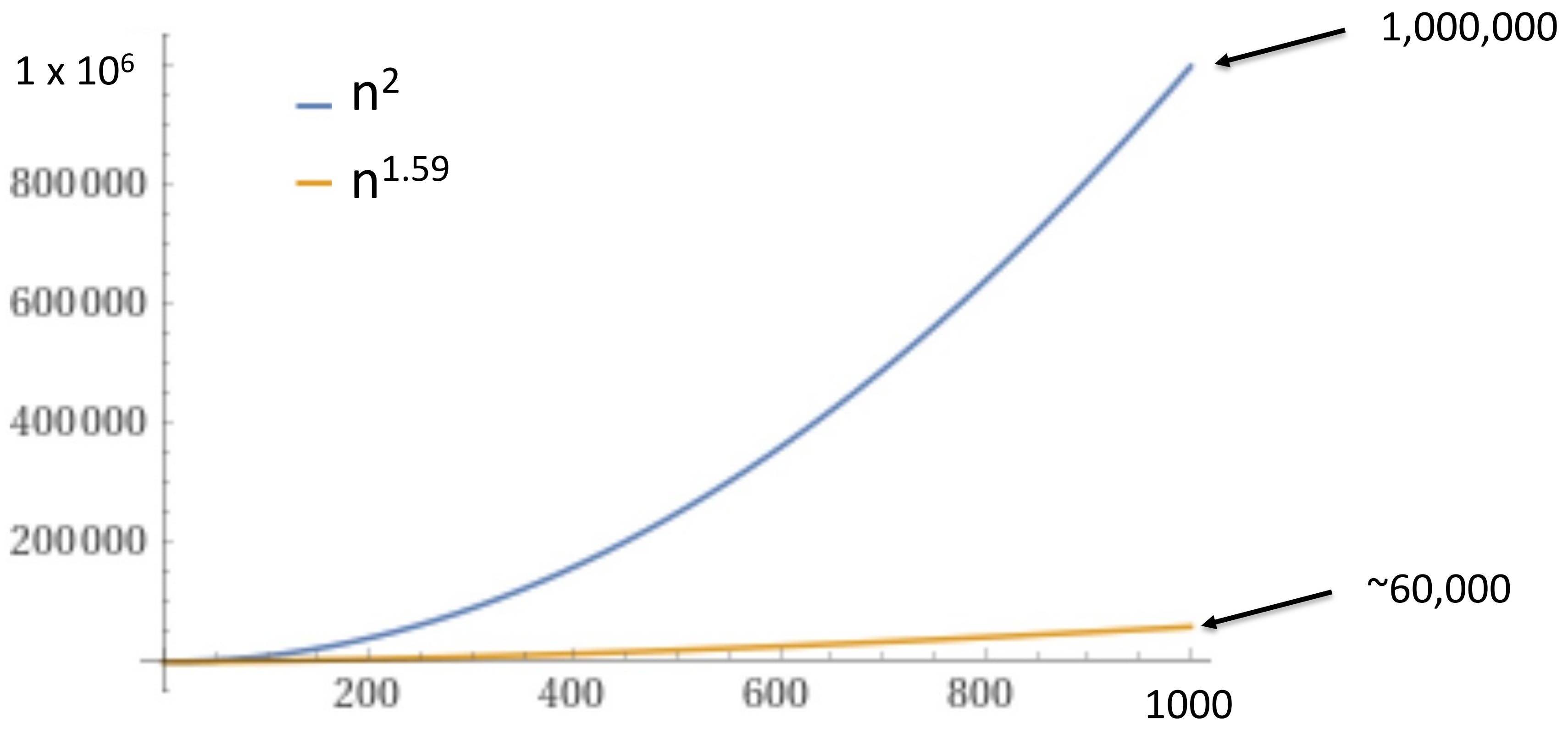
Run time?

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 3 \cdot T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Case 1: If $c > \log_b a$, then $T(n) = \Theta(n^c)$
Case 2: If $c = \log_b a$, then $T(n) = \Theta(n^c \log n)$
Case 3: If $c < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$

$$T(n) = O(n^{\log_2 3}) = O(n^{1.59})$$

Integer multiplication



Courtesy of Wolfram Alpha

Integer multiplication algorithms

year	algorithm	bit operations
12xx	**grade school**	$O(n^2)$
1962	**Karatsuba–Ofman**	$O(n^{1.585})$
1963	**Toom–3, Toom–4**	$O(n^{1.465}),\ O(n^{1.404})$
1966	**Toom–Cook**	$O(n^{1+\varepsilon})$
1971	**Schönhage–Strassen**	$O(n \log n \cdot \log \log n)$
2007	**Fürer**	$n \log n \, 2^{O(\log^* n)}$
2019	**Harvey–van der Hoeven**	$O(n \log n)$
	???	$O(n)$

GNU Multiprecision Arithmetic Library (GMP) uses 7 different algorithms, choosing one based on size of integers:
Karatsuba, variants of Toom and Toom-Cook, Schonhage-Strassen (FFT-based method)

Table replicated form https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/05DivideAndConquerII.pdf

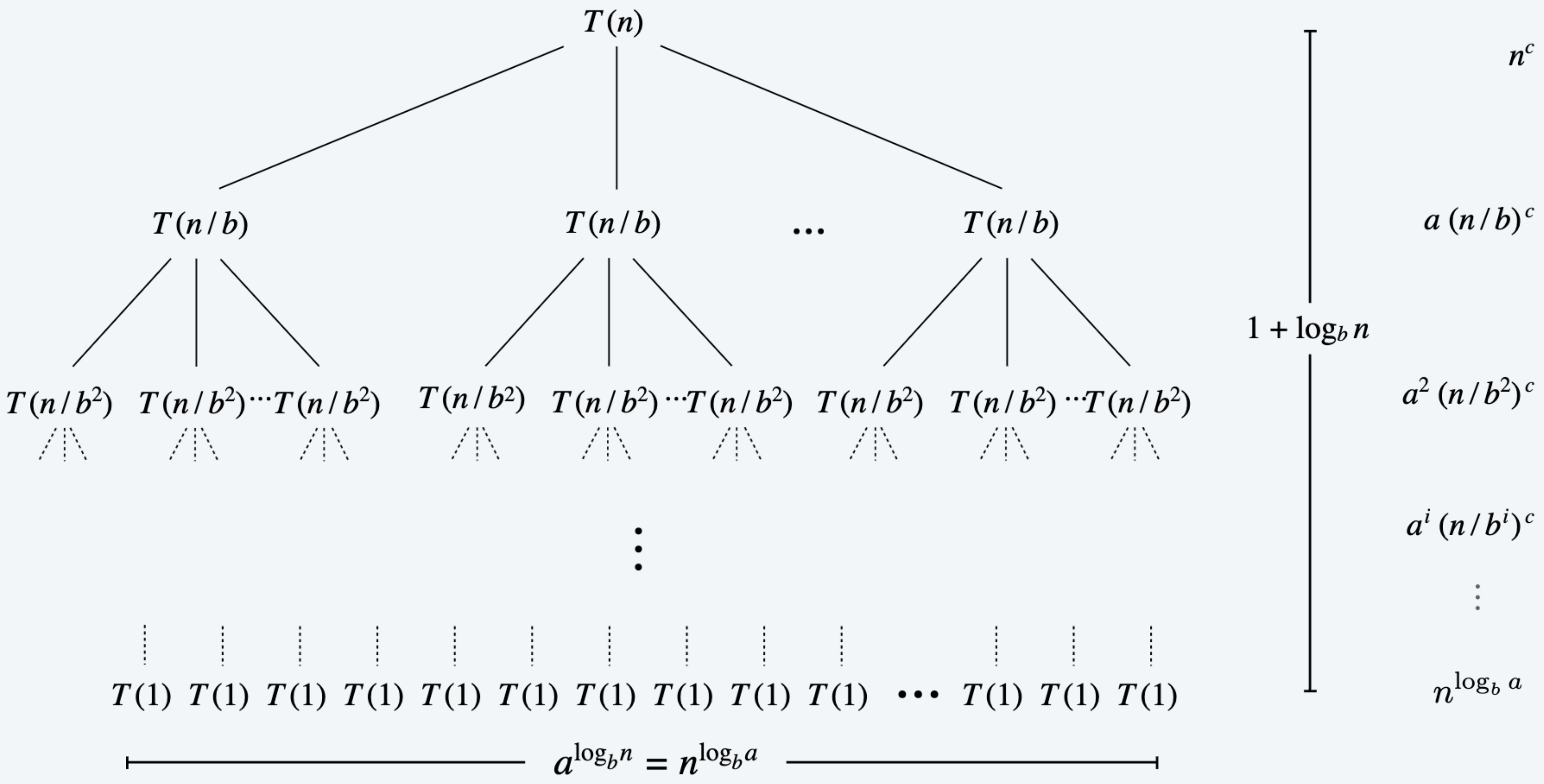
More D&C algorithms

Textbook discusses some more examples

Problem	Algorithm	Approach	Worst case run time
Sorting list	Merge Sort	Divide in two halves, sort each, and then merge	O(n log n)
Counting inversions	Sort-and-Count	Merge sort plus keeping track of number of inversions efficiently as you merge	O(n log n)
Integer multiplication	Karatsuba	Divide numbers into low and high order bits, use three recursive multiplications and combine	$O(n^{1.59})$
Closest pairs of points	Closest-Pair	Divide plane in half and solve on each half, carefully combine by looking at points near division	O(n log n)
Convolutions	Fast Fourier Transform	Treat inputs as coefficients of polynomials, apply FFT to the polynomials	O(n log n)

Divide-and-conquer recurrences: recursion tree

Suppose $T(n)$ satisfies $T(n) = a\,T(n/b) + n^c$ with $T(1) = 1$, for n a power of b.



$r = a / b^c \qquad T(n) = n^c \sum_{i=0}^{\log_b n} r^i$

Replicated from: https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/05DivideAndConquerII.pdf

4

Divide-and-conquer recurrences: recursion tree analysis

Suppose $T(n)$ satisfies $T(n) = a\,T(n/b) + n^c$ with $T(1) = 1$, for n a power of b.

Let $r = a / b^c$. Note that $r < 1$ iff $c > \log_b a$.

$$T(n) = n^c \sum_{i=0}^{\log_b n} r^i = \begin{cases} \Theta(n^c) & \text{if } r < 1 \quad c > \log_b a \longleftarrow \text{cost dominated by cost of root} \\ \Theta(n^c \log n) & \text{if } r = 1 \quad c = \log_b a \longleftarrow \text{cost evenly distributed in tree} \\ \Theta(n^{\log_b a}) & \text{if } r > 1 \quad c < \log_b a \longleftarrow \text{cost dominated by cost of leaves} \end{cases}$$

Geometric series.

- If $0 < r < 1$, then $1 + r + r^2 + r^3 + \ldots + r^k \leq 1 / (1 - r)$.
- If $r = 1$, then $1 + r + r^2 + r^3 + \ldots + r^k = k + 1$.
- If $r > 1$, then $1 + r + r^2 + r^3 + \ldots + r^k = (r^{k+1} - 1) / (r - 1)$.

Replicated from: https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/05DivideAndConquerII.pdf

Claim. ABL(T')=ABL(T)-f_ω

Pf.

$$
\begin{aligned}
\mathrm{ABL}(T) &= \sum_{x\in S} f_x \cdot \mathrm{depth}_T(x) \\
&= f_y \cdot \mathrm{depth}_T(y) + f_z \cdot \mathrm{depth}_T(z) + \sum_{x\in S, x\neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= \left(f_y + f_z\right) \cdot \left(1 + \mathrm{depth}_T(\omega)\right) + \sum_{x\in S, x\neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= f_\omega \cdot \left(1 + \mathrm{depth}_T(\omega)\right) + \sum_{x\in S, x\neq y,z} f_x \cdot \mathrm{depth}_T(x) \\
&= f_\omega + \sum_{x\in S'} f_x \cdot \mathrm{depth}_{T'}(x) \\
&= f_\omega + \mathrm{ABL}(T')
\end{aligned}
$$

[Credit: https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pearson/04Huffman.pdf]