Divide and Conquer

CS 3AC3

Ryszard Janicki

Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Acknowledgments: Material based on *Algorithm Design* by Jon Kleinberg and Éva Tardos (Chapter 5)

Divide-and-conquer paradigm

Divide-and-conquer.

- Divide up problem into several subproblems.
- Solve each subproblem recursively.
- Combine solutions to subproblems into overall solution.

Most common usage.

- Divide problem of size n into two subproblems of size $n/2$ in linear time.
- Solve two subproblems recursively.
- Combine two solutions into overall solution in linear time.

Consequence.

- Other method: $\Theta(n^2)$.
- Divide-and-conquer: $\Theta(n \log n)$.

Divide-and-conquer: an example

Is a given number a power of 2?

Input: a non-negative integer n

Output: YES, if there is k such that $n = 2^k$.
NO, otherwise.

Examples: $32 \implies$ YES, as $32 = 2^6$, $15 \implies$ NO, as $15 = 2^4 + 7$.

Divide-and-conquer Solution:

---

test(m)

1. IF $m = 1$ THEN
2. $test \leftarrow$ YES
3. ELSE IF $(m \mod 2) = 0$ THEN
4. $test \leftarrow test(m/2)$
5. ELSE $test \leftarrow$ NO

---

test(m)

1. IF $m = 1$ THEN
2. $test \leftarrow$ YES
3. ELSE IF (m mod 2) $= 0$ THEN
4. $test \leftarrow test(m/2)$
5. ELSE $test \leftarrow$ NO

Times assign to each line:

1. $1 \rightarrow c_1$
2. $1 \rightarrow c_2$
3. $1 \rightarrow c_1$
4. $1 \rightarrow c_2 + T(m/2)$
5. $1 \rightarrow c_2$

RECURRENCE RELATION

$$T(m) = \begin{cases} c_1 + c_2 & m = 1 \\ 2c_1 + c_2 & m > 1 \text{ and } m \text{ is odd} \\ 2c_1 + c_2 + T(m/2) & m \text{ is even} \end{cases}$$

- $\lceil x \rceil$ is the smallest integer $\geq x$,
  eg. $\lceil 1.5 \rceil = 2$, $\lceil 3.1 \rceil = 4$, $\lceil 3.0 \rceil = 3$,
- Note that: $2^{\lceil \log m \rceil} \geq m$ as $\lceil x \rceil \geq x$ and $m = 2^k \iff \log m = k$.
- If $\lceil \log m \rceil = k$ then $2^{\lceil \log m \rceil} = m$.
- For $m = 6$ we have:
  $\log 4 = 2$ and $\log 8 = 3 \implies 3 < \log 6 < 3 \implies \lceil \log 6 \rceil = 3$
  $\implies 2^{\lceil \log 6 \rceil} > 6$.

Some upper bound of $T(n)$

$$T(m) = \begin{cases} c_1 + c_2 & m = 1 \\ 2c_1 + c_2 & m > 1 \text{ and } m \text{ is odd} \\ 2c_1 + c_2 + T(m/2) & m \text{ is even} \end{cases}$$

We define $T'(m)$ for real numbers as:

$$T'(m) = \begin{cases} c & m \geq 1 \\ c + T'(m/2) & m > 1 \end{cases}$$

where $c = 2c_1 + c_2$.
Note that for all $m > 0$, we have:

$$T(m) \leq T'(m),$$

i.e. $T'(m)$ is an upper bound of $T(m)$.

Proof that $T'(m)$ is $O(\log m)$

$$T'(m) = \begin{cases} c & m \geq 1 \\ c + T'(m/2) & m > 1 \end{cases}$$

$$\begin{aligned} T'(m) &= c + T'(m/2) = c + c + T'(m/2^2) = 2c + T'(m/2^2) \\ &= 3c + T'(m/2^3) \\ &\quad \cdots \\ &= kc + T'(m/2^k) \\ &\quad \cdots \\ &= \lceil \log m \rceil c + T'(\tfrac{m}{2^{\lceil \log m \rceil}}) \end{aligned}$$

{ Note $T'(\frac{m}{2^{\lceil \log m \rceil}})$ is a constant, i.e. $c_0 = T'(\frac{m}{2^{\lceil \log m \rceil}})$.}

$$= C \cdot (\lceil \log m \rceil + 1) = \mathbf{O}(\log \mathbf{m}),$$ where $C = \max(c, c_0)$.

Master method

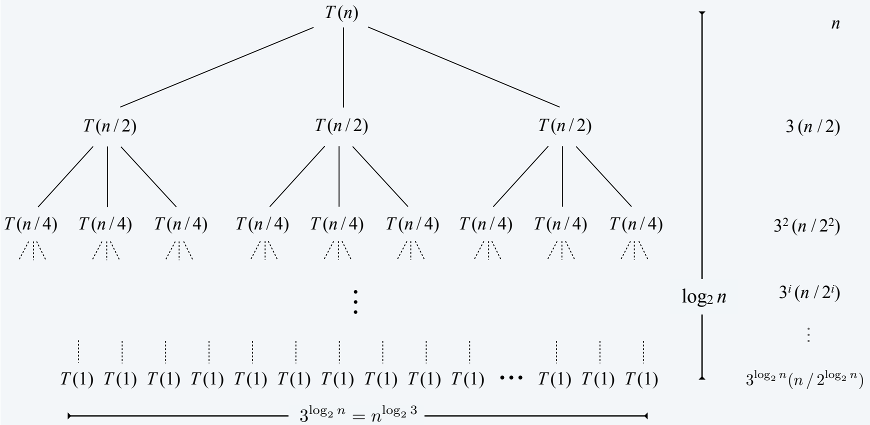
- Goal. Recipe for solving common divide-and-conquer recurrences:

$$T(n) = aT(\frac{n}{b}) + f(n)$$

- Terms.
  - $a \geq 1$ is the number of subproblems.
  - $b > 0$ is the factor by which the subproblem size decreases.
  - $f(n) =$ work to divide/merge subproblems.
- Recursion tree.
  - $k = \log_b n$ levels.
  - $a^i =$ number of subproblems at level i.
  - $n/b^i =$ size of subproblem at level i.

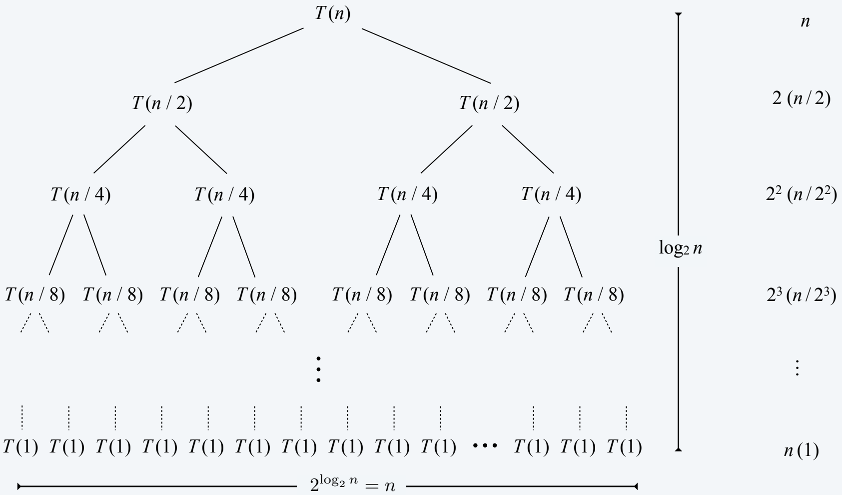
Recursion Tree: total cost is dominated by cost of leaves

Ex 1. If $T(n)$ satisfies $T(n) = 3\,T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n^{\lg 3})$.



$r = 3/2 > 1 \qquad T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_2 n})\,n = \dfrac{r^{1+\log_2 n} - 1}{r - 1}\,n = 3n^{\log_2 3} - 2n$

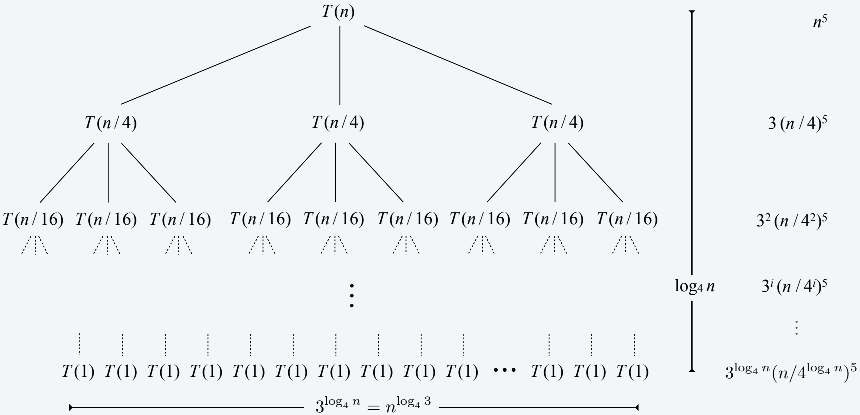
Ex 2. If $T(n)$ satisfies $T(n) = 2\,T(n/2) + n$, with $T(1) = 1$, then $T(n) = \Theta(n \log n)$.



$r = 1$ $$T(n) = (1 + r + r^2 + r^3 + \ldots + r^{\log_2 n})\,n = n\,(\log_2 n + 1)$$

Recursion Tree: cost is dominated by cost of root

Ex 3. If $T(n)$ satisfies $T(n) = 3\,T(n/4) + n^5$, with $T(1) = 1$, then $T(n) = \Theta(n^5)$.



$$r = 3/4^5 < 1 \qquad n^5 \le T(n) \le (1 + r + r^2 + r^3 + \dots)\, n^5 \le \frac{1}{1-r}\, n^5$$

Master Theorem

Theorem

Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = aT(\frac{n}{b}) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. *If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.*

Example

$T(n) = 3T(n/2) + n$.

- $a = 3, b = 2, f(n) = n, k = \log_2 3$.
- $T(n) = \Theta(n^{\log_2 3})$.

Master Theorem

Theorem

Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = aT(\frac{n}{b}) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 2. *If $f(n) = \Theta(n^k)$, then $T(n) = \Theta(n^k \log n)$.*

Example

$T(n) = 2T(n/2) + \Theta(n)$.

- $a = 2, b = 2, f(n) = 17n$ ($f(n) = 175n$, etc.), $k = \log_2 2 = 1$.
- $T(n) = \Theta(n \log n)$.

Master Theorem

Theorem

Suppose that $T(n)$ *is a function on the nonnegative integers that satisfies the recurrence*

$$T(n) = aT(\frac{n}{b}) + f(n)$$

where n/b *means either* $\lfloor n/b \rfloor$ *or* $\lceil n/b \rceil$. *Let* $k = \log_b a$. *Then,*

Case 3. *If* $f(n) = \Omega(n^{k+\varepsilon})$ *for some constant* $\varepsilon > 0$, *and if* $af(n/b) \leq cf(n)$ *for some constant* $c < 1$ *and all sufficiently large* n, *then* $T(n) = \Theta(f(n))$.

Note. The condition $af(n/b) \leq cf(n)$ *holds if* $f(n) = \Theta(n^{k+\varepsilon})$.

Example

$T(n) = 3T(n/4) + n^5$.

- $a = 3, b = 4, f(n) = n^5, k = \log_4 3$.
- $T(n) = \Theta(n^5)$.

Theorem (Master Theorem)

Suppose that $T(n)$ is a function on the nonnegative integers that satisfies the recurrence

$$T(n) = aT(\frac{n}{b}) + f(n)$$

where n/b means either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Let $k = \log_b a$. Then,

Case 1. *If $f(n) = O(n^{k-\varepsilon})$ for some constant $\varepsilon > 0$, then $T(n) = \Theta(n^k)$.*
Case 2. *If $f(n) = \Theta(n^k)$, then $T(n) = \Theta(n^k \log n)$.*
Case 3. *If $f(n) = \Omega(n^{k+\varepsilon})$ for some constant $\varepsilon > 0$, and if $af(n/b) \le cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.*

Proof. (Sketch).

- Use recursion tree to sum up terms (assuming n is a power of b).
- Three cases for geometric series.
- Deal with floors and ceilings.

□

Analysis of Master Theorem

- In each case we compare $f(n)$ with $n^{\log_b a}$.
- The solution is determined by the larger of these two functions.
- Case 1. $n^{\log_b a}$ is larger, hence $T(n) = \Theta(n^{\log_b a})$.
- Case 3. $f(n)$ is larger, hence $T(n) = \Theta(f(n))$.
- Case 2. $f(n)$ and $n^{\log_b a}$ are of the same size, so $T(n) = \Theta(n^{\log_b a} \log n)$.

More subtle analysis.

- Case 1. $f(n)$ is not only smaller but polynomially smaller, i.e. by a factor n^{ε}, $\varepsilon > 0$.
- Case 2. $f(n)$ is not only larger but polynomially smaller, i.e. by a factor n^{ε}, $\varepsilon > 0$.
- There are cases between 1–2 and 2–3!

Applications

- $T(n) = 9T(n/3) + n$
  Hence: $a = 9, b = 3, f(n) = n$, so
  $n^{\log_b a} = n^{\log_3 9} = n^2 = \Theta(n^2)$.
  Since $f(n) = O(n) = O(n^{\log_3 9-\varepsilon})$, where $\varepsilon = 1$, we can apply Case 1, i.e.
  $$T(n) = \Theta(n^2).$$

- $T(n) = T(2n/3) + 1$
  Here: $a = 1, b = 3/2, f(n) = 1$, and
  $n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$.
  So it is Case 2 as $f(n) = \Theta(1) = \Theta(n^{\log_b a})$, i.e.
  $$T(n) = \Theta(\log n).$$

- $T(n) = 3T(n/4) + n\log n$
  Here: $a = 3, b = 4, f(n) = n\log n$, and
  $n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$.
  Since $f(n) = n\log n = \Omega(n^{\log_4 3+\varepsilon})$, where $\varepsilon \approx 0.2$ (we need $\log_4 3 + \varepsilon > 1$), we may try Case 3.
  For sufficiently large n,

  $$af(n/b) = 3(n/4)\log(n/4) \leq (3/4)n\log n = c \cdot f(n)$$

  for $c = 3/4$.
  Hence we can apply Case 3, i.e.

  $$T(n) = \Theta(n\log n).$$

Master Theorem may not work!

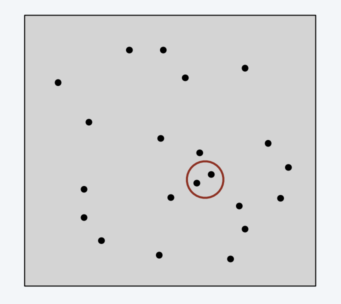
- $T(n) = 2T(n/2) + n \log n$
  $a = 2, b = 2, f(n) = n \log n, n^{\log_b a} = n$.
- Unfortunately, Case 3 doe not work since even though $f(n) = n \log n$ is asymptotically larger that $n^{\log_b a} = n$, it is **not** polynomially larger.
- The ratio $\frac{f(n)}{n^{\log_b a}} = \frac{n \log n}{n} = \log n$ is asymptotically less than n^{ε} for any positive ε.

Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance (i.e. $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$, for points (x_1, y_1) and (x_2, y_2)) between them.

Fundamental geometric primitive.

- Graphics, computer vision, geographic information systems, molecular modeling, air traffic control.
- Special case of nearest neighbor, etc.

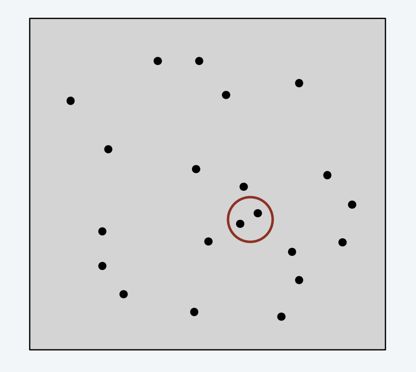


Closest pair of points

Closest pair problem. Given n points in the plane, find a pair of points with the smallest Euclidean distance between them.

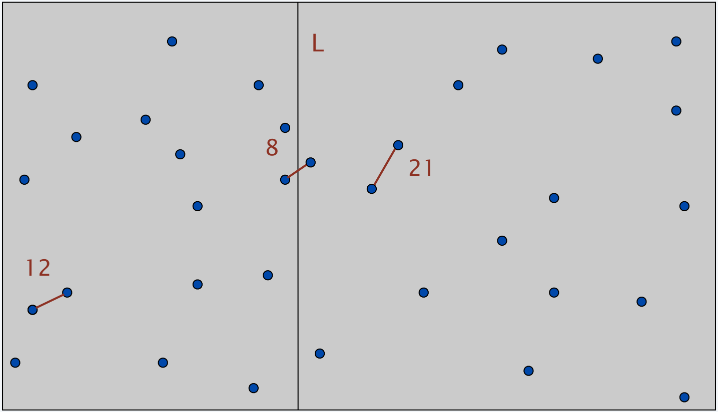
Brute force. Check all pairs with $\Theta(n^2)$ distance calculations.

Nondegeneracy assumption. No two points have the same x-coordinate (this can always be achieved by small plane rotation!).



Closest pair of points: divide-and-conquer algorithm

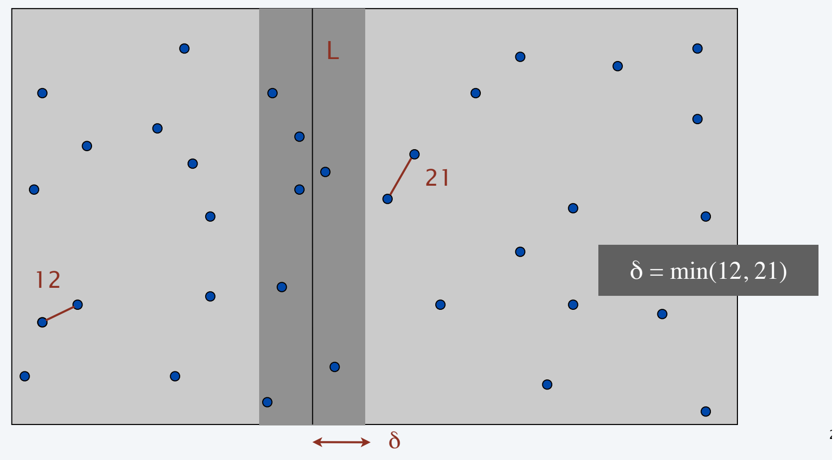
- **Divide:** draw vertical line L so that $n/2$ points on each side.
- **Conquer:** find closest pair in each side recursively.
- **Combine:** find closest pair with one point in each side (seems like $\Theta(n^2)$?).
- Return best of 3 solutions.



How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance δ.

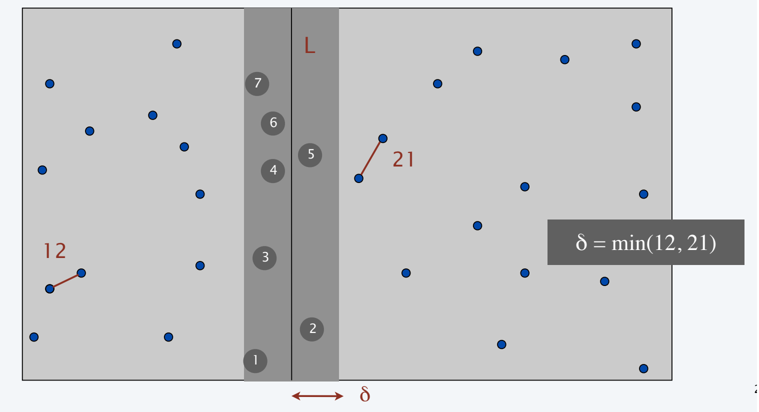
- Observation: only need to consider points within δ of line L.



How to find closest pair with one point in each side?

Find closest pair with one point in each side, assuming that distance δ.

- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y-coordinate.
- Only check distances of those within 11 positions in sorted list! (why 11?)



How to find closest pair with one point in each side?

Definition

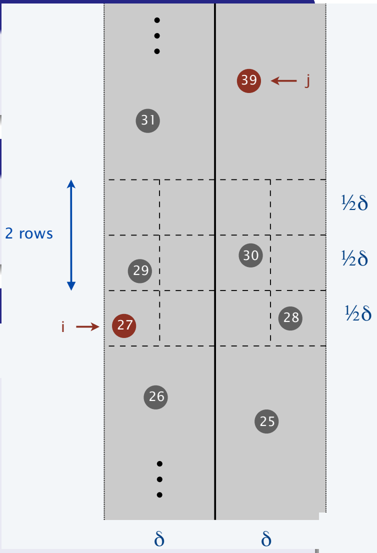
Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate.

Claim

If $|i-j| \geq 12$, then the distance between s_i and s_j is at least δ.

Proof.

- No two points lie in same $\frac{1}{2}\delta$-by-$\frac{1}{2}\delta$ box.
- Two points at least 2 rows apart have distance $\geq 2(\frac{1}{2}\delta)$.



Closest pair of points: divide-and-conquer algorithm

CLOSEST-PAIR $(p_1, p_2, \ldots, p_n)$

Compute separation line L such that half the points are on each side of the line.	← $O(n \log n)$
$\delta_1 \leftarrow$ CLOSEST-PAIR (points in left half).	← $2\,T(n/2)$
$\delta_2 \leftarrow$ CLOSEST-PAIR (points in right half).	
$\delta \leftarrow \min\{\delta_1, \delta_2\}$.	
Delete all points further than δ from line L.	← $O(n)$
Sort remaining points by y-coordinate.	← $O(n \log n)$
Scan points in y-order and compare distance between each point and next 11 neighbors. If any of these distances is less than δ, update δ.	← $O(n)$
RETURN δ.	

Closest pair of points: analysis

Theorem

The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log^2 n)$ time.

Proof.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + O(n \log n) & \text{otherwise} \end{cases}$$

or, just $T(n) = 2T(n/2) + O(n \log n)$.
Hence, by Master Theorem Case 2, $T(n) = \Theta(n \log^2 n)$ time. □

Improved closest pair algorithm

Question. How to improve to $f(n) = O(n \log n)$? Answer. Do not sort points in strip from scratch each time.

- Each recursive returns two lists: all points sorted by x-coordinate, and all points sorted by y-coordinate.
- Sort by merging two pre-sorted lists (merging is $\Theta(n)$).

Theorem

The divide-and-conquer algorithm for finding the closest pair of points in the plane can be implemented in $O(n \log n)$ time.

Proof.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{otherwise} \end{cases}$$

or, just $T(n) = 2T(n/2) + \Theta(n)$.
Hence, by Master Theorem Case 2, $T(n) = \Theta(n \log n)$ time. □

Note. There is a *randomized* closest pair algorithm that run in $O(n)$ time.

Integer addition

- Addition. Given two n-bit integers a and b, compute $a + b$.
- Subtraction. Given two n-bit integers a and b, compute $a - b$.
- Grade-school algorithm. $\Theta(n)$ bit operations.

1	1	1	1	1	1	0	1	
	1	1	0	1	0	1	0	1
+	0	1	1	1	1	1	0	1
1	0	1	0	1	0	0	1	0

- Remark Grade-school addition and subtraction algorithms are asymptotically optimal.

Integer multiplication

- Multiplication. Given two n-bit integers a and b, compute a $\times$ b.
- Grade-school algorithm. $\Theta(n^2)$ bit operations.

								1	1	0	1	0	1	0	1
							×	0	1	1	1	1	1	0	1
								1	1	0	1	0	1	0	1
							0	0	0	0	0	0	0	0	
						1	1	0	1	0	1	0	1		
					1	1	0	1	0	1	0	1			
				1	1	0	1	0	1	0	1				
			1	1	0	1	0	1	0	1					
		1	1	0	1	0	1	0	1						
	0	0	0	0	0	0	0	0							
0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	1

- Question. Is grade-school algorithm optimal?
- No.

Divide-and-conquer multiplication

To multiply two n-bit integers x and y:

- Divide x and y into low- and high-order bits.
  Example. $x = \underbrace{1000}_{a}\underbrace{1101}_{b}\ y = \underbrace{1110}_{c}\underbrace{0001}_{d}$

  $m = \lceil n/2 \rceil$
  $a = \lfloor x/2^m \rfloor \quad b = x \mod 2^m$
  $c = \lfloor y/2^m \rfloor \quad d = y \mod 2^m$
  Bit shifting can be used to compute a, b, c and d.
- Now we have: $x = 2^m a + b$ and $y = 2^m c + d$.
- Multiply four $\frac{1}{2}n$-bit integers, recursively.
- Add and shift to obtain result.

$$xy = (2^m a + b)(2^m c + d) = 2^{2m} \underbrace{ac}_{1} + 2^m(\underbrace{bc}_{2} + \underbrace{ad}_{3}) + \underbrace{bd}_{4}$$

Divide-and-conquer multiplication

MULTIPLY(x, y, n)

IF $(n = 1)$

RETURN $x \times y$.

ELSE

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

$e \leftarrow$ MULTIPLY(a, c, m).

$f \leftarrow$ MULTIPLY(b, d, m).

$g \leftarrow$ MULTIPLY(b, c, m).

$h \leftarrow$ MULTIPLY(a, d, m).

RETURN $2^{2m}\, e + 2^m\, (g + h) + f$.

Divide-and-conquer multiplication analysis

Proposition

The divide-and-conquer multiplication algorithm requires $\Theta(n^2)$ bit operations to multiply two n-bit integers.

Proof.

Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{4T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \implies T(n) = \Theta(n^2)$$

□

Not better than grade-school algorithm!

Karatsuba trick

$x = \underbrace{1000}_{a}\underbrace{1101}_{b} \; y = \underbrace{1110}_{c}\underbrace{0001}_{d}$

$m = \lceil n/2 \rceil$

$a = \lfloor x/2^m \rfloor \quad b = x \mod 2^m$

$c = \lfloor y/2^m \rfloor \quad d = y \mod 2^m$

$$xy = (2^m a + b)(2^m c + d) = 2^{2m}\underbrace{ac}_{1} + 2^m \overbrace{(\underbrace{bc}_{2} + \underbrace{ad}_{3})}^{\text{middle term}} + \underbrace{bd}_{4}$$

- To compute middle term $bc + ad$, use identity:
  $bc + ad = ac + bd - (a - b)(c - d)$

Now we have:

$$xy = 2^{2m}\underbrace{ac}_{1} + 2^m(\underbrace{ac}_{1} + \underbrace{bd}_{2} - \underbrace{(a - b)(c - d)}_{3}) + \underbrace{bd}_{2}$$

Bottom line. Only three multiplications of $\frac{1}{2}$-bit integers!

KARATSUBA-MULTIPLY(x, y, n)

IF $(n = 1)$

RETURN $x \times y$.

ELSE

$m \leftarrow \lceil n / 2 \rceil$.

$a \leftarrow \lfloor x / 2^m \rfloor$; $b \leftarrow x \bmod 2^m$.

$c \leftarrow \lfloor y / 2^m \rfloor$; $d \leftarrow y \bmod 2^m$.

$e \leftarrow$ KARATSUBA-MULTIPLY(a, c, m).

$f \leftarrow$ KARATSUBA-MULTIPLY(b, d, m).

$g \leftarrow$ KARATSUBA-MULTIPLY$(a - b, c - d, m)$.

RETURN $2^{2m} e + 2^m (e + f - g) + f$.

Karatsuba analysis

Proposition

Karatsuba's multiplication algorithm requires $\Theta(n^{1.585})$ bit operations to multiply two n-bit integers.

Proof.

Apply case 1 of the master theorem to the recurrence:

$$T(n) = \underbrace{3T(n/2)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, shift}} \implies T(n) = \Theta(n^{\log_2 3}) = \Theta(n^{1.585})$$

□

Practice. Faster than grade-school algorithm for about 320-640 bits.