Day 8—More Divide and Conquer and Master Method for Solving Recurrences

Neil Rhodes
CSE 101
UC San Diego

MIDTERM April 29, 4-5 PM Center Hall Room 101
NOT HSS

5.4 Closest Pair of Points

Closest Pair of Points

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

Fundamental geometric primitive.

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

fast closest pair inspired fast algorithms for these problems

Brute force. Check all pairs of points p and q with $\Theta(n^2)$ comparisons.

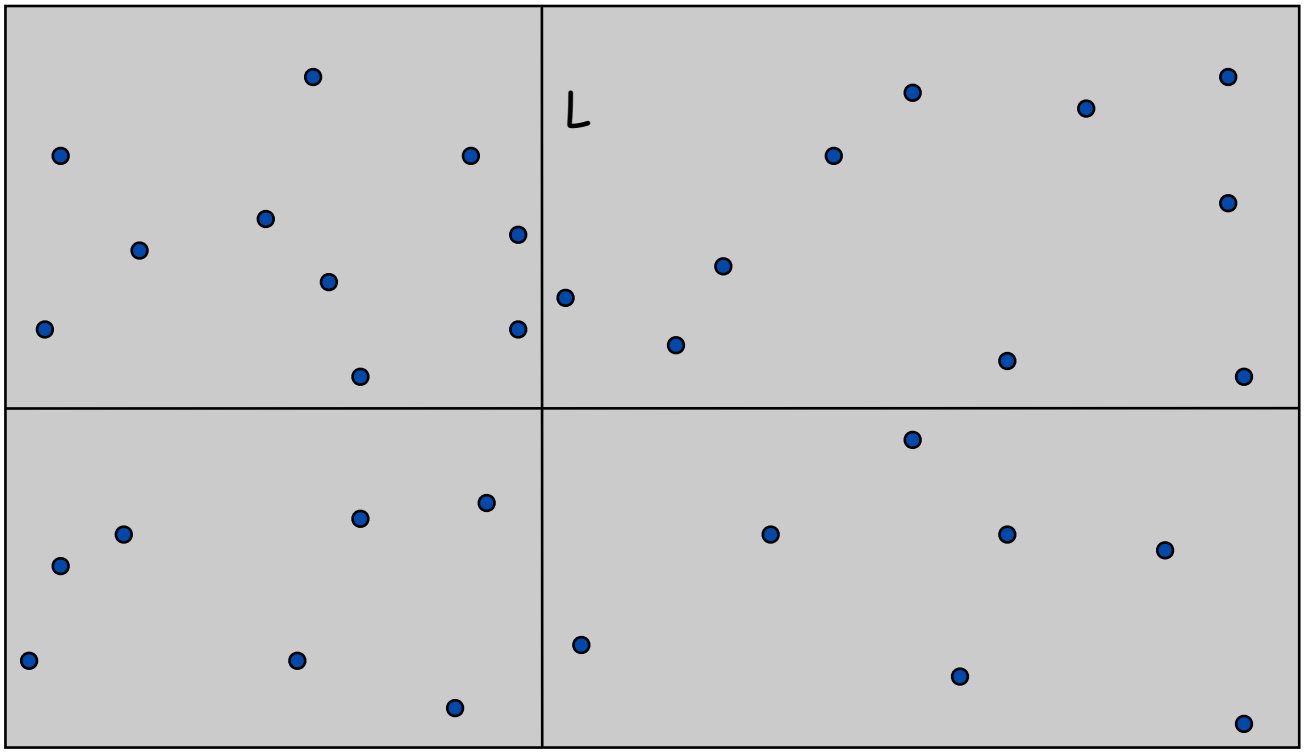
1-D version. O(n log n) easy if points are on a line.

Assumption. No two points have same x coordinate.

to make presentation cleaner

Closest Pair of Points: First Attempt

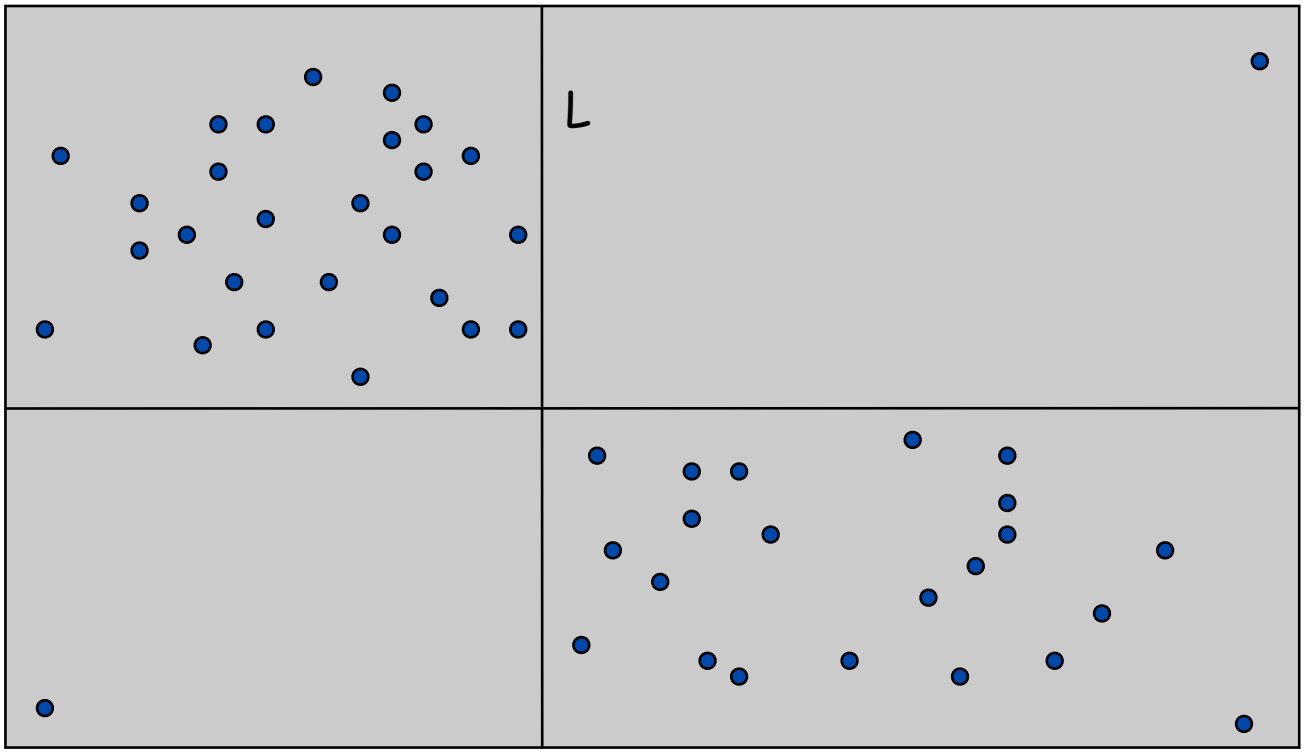
Divide. Sub-divide region into 4 quadrants.



Closest Pair of Points: First Attempt

Divide. Sub-divide region into 4 quadrants.

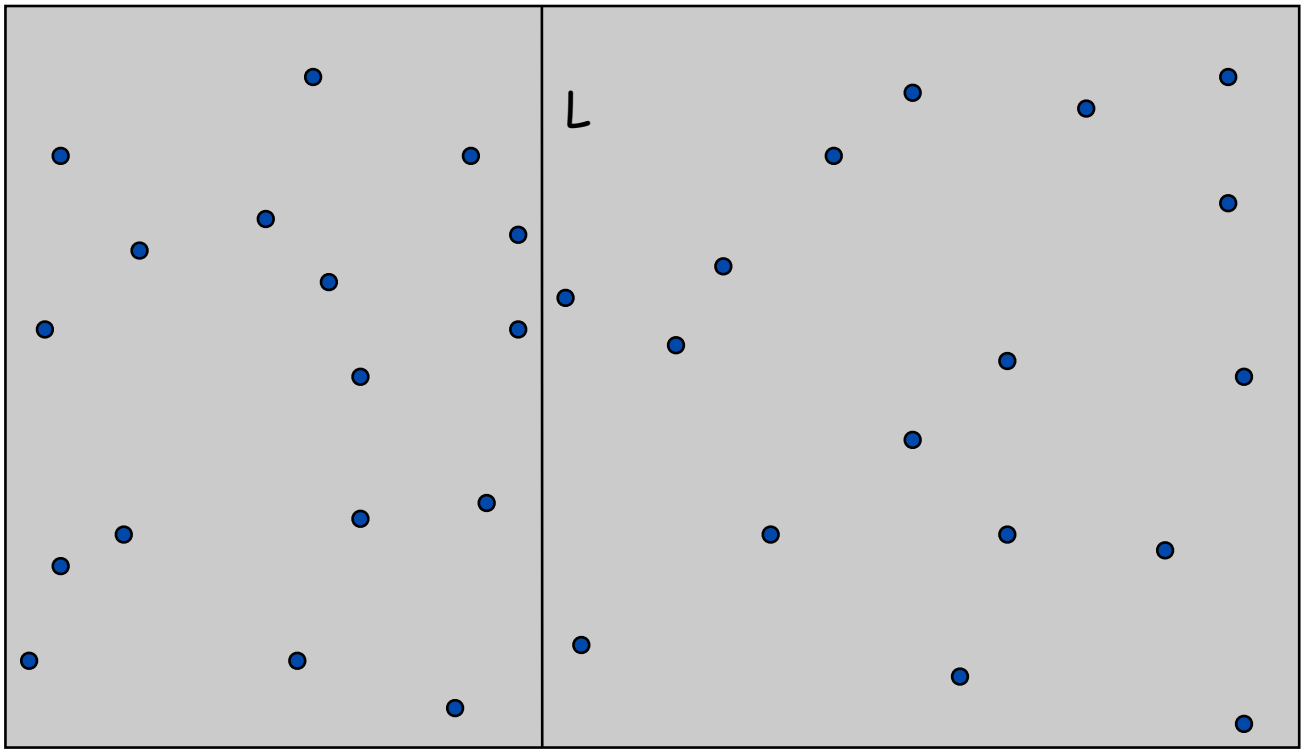
Obstacle. Impossible to ensure n/4 points in each piece.



Closest Pair of Points

Algorithm.

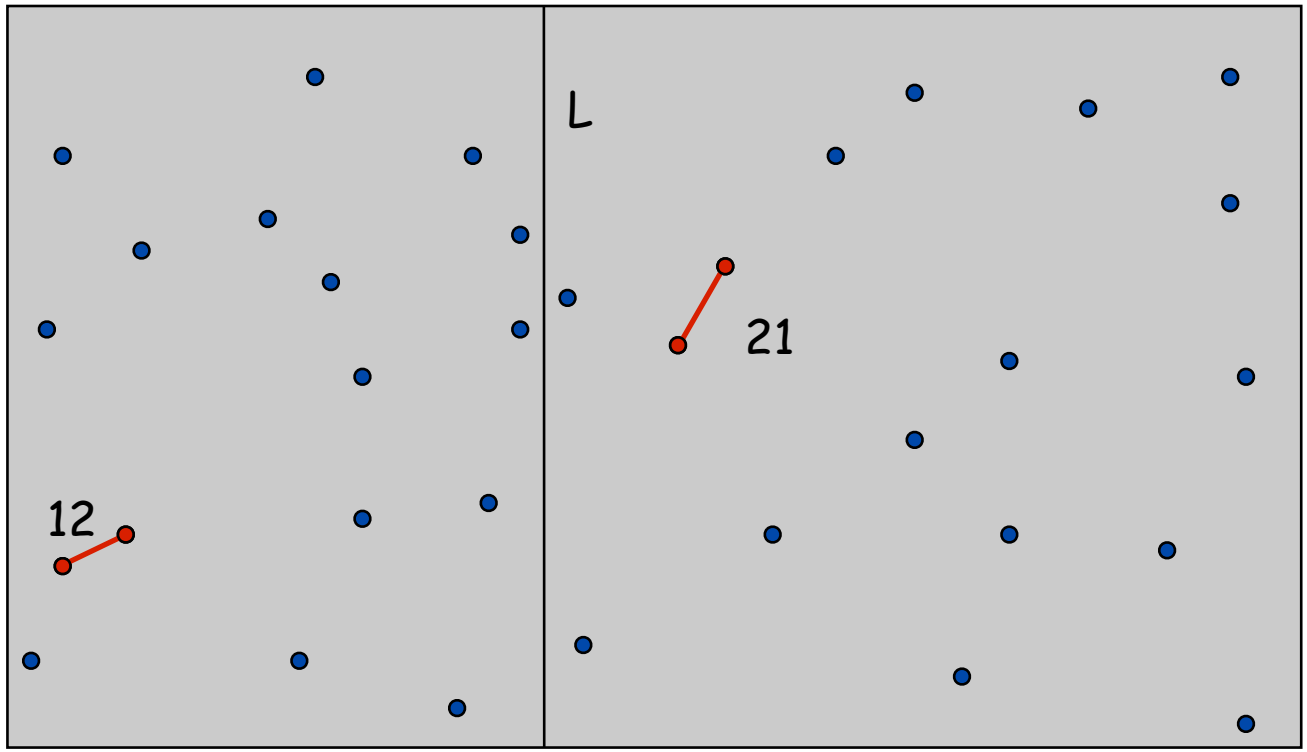
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.



Closest Pair of Points

Algorithm.

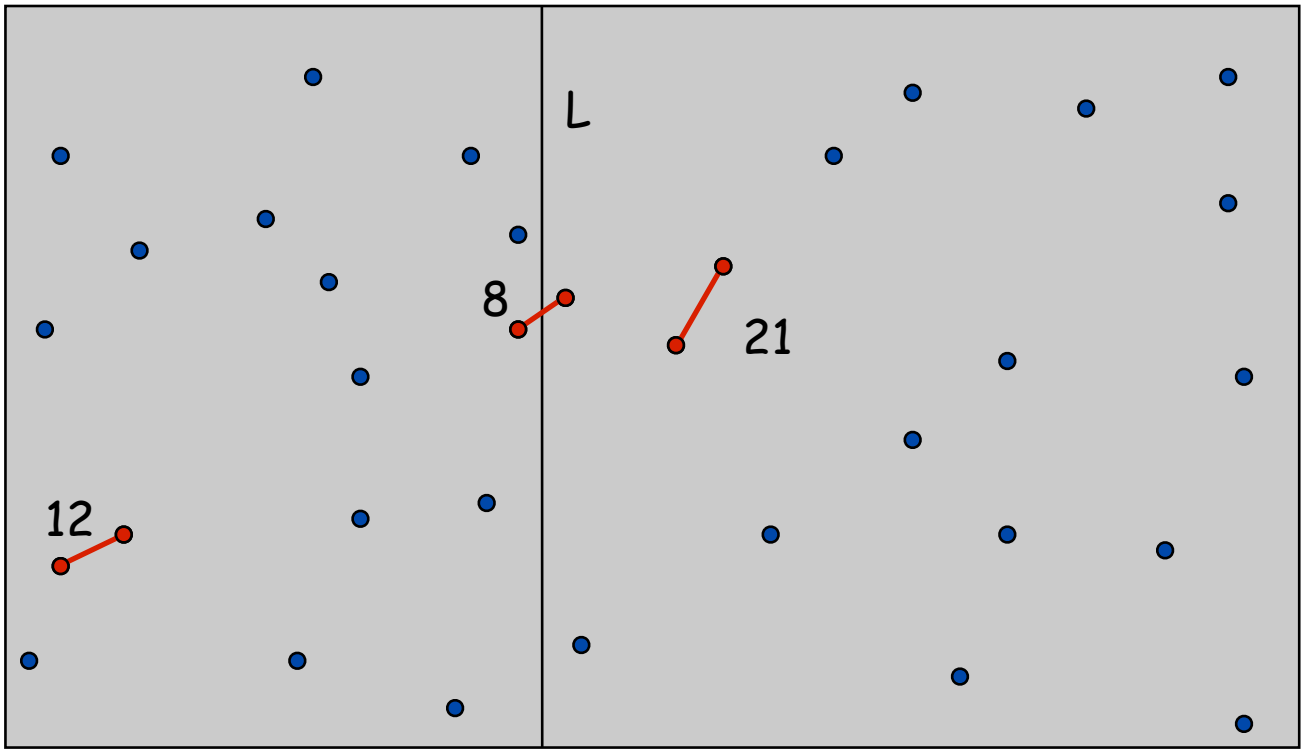
- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ points on each side.
- Conquer: find closest pair in each side recursively.



Closest Pair of Points

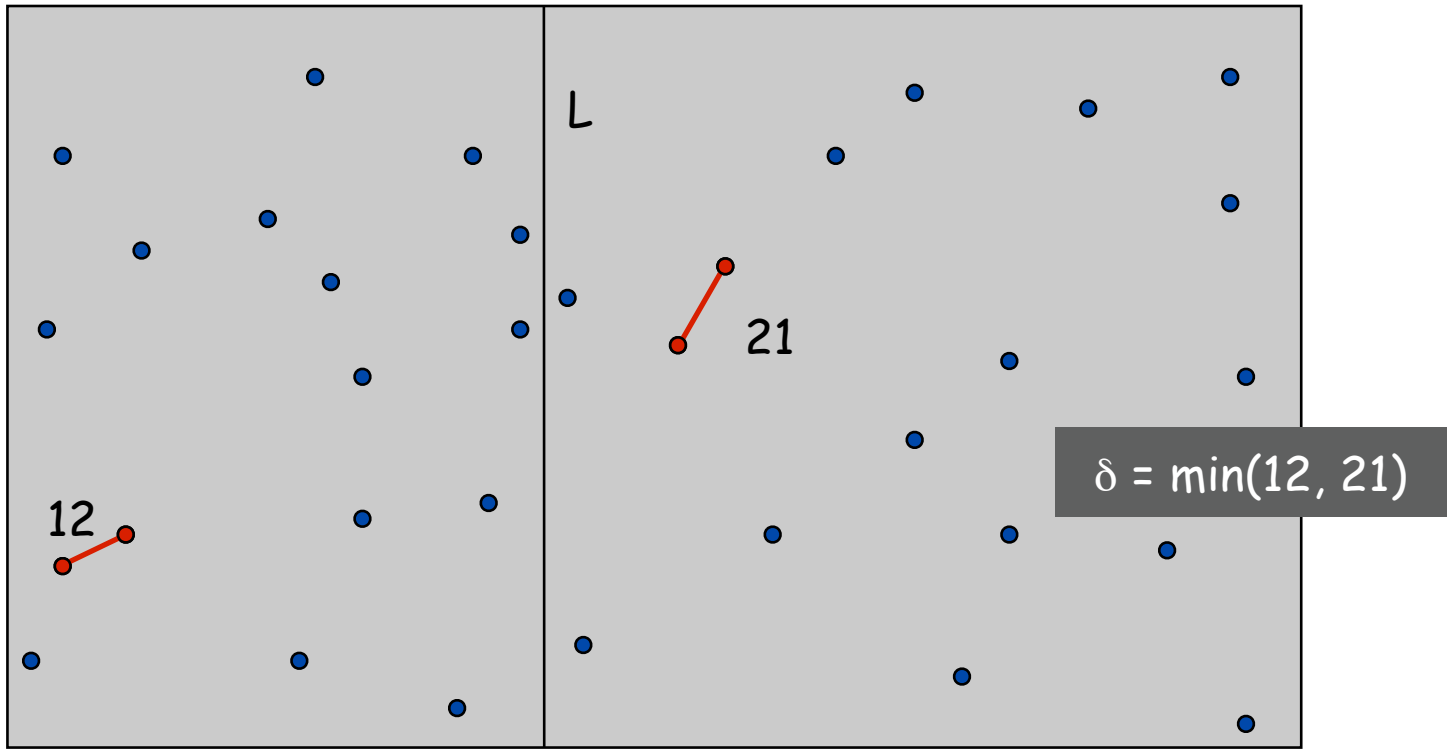
Algorithm.

- Divide: draw vertical line L so that roughly $\frac{1}{2}n$ 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.



Closest Pair of Points

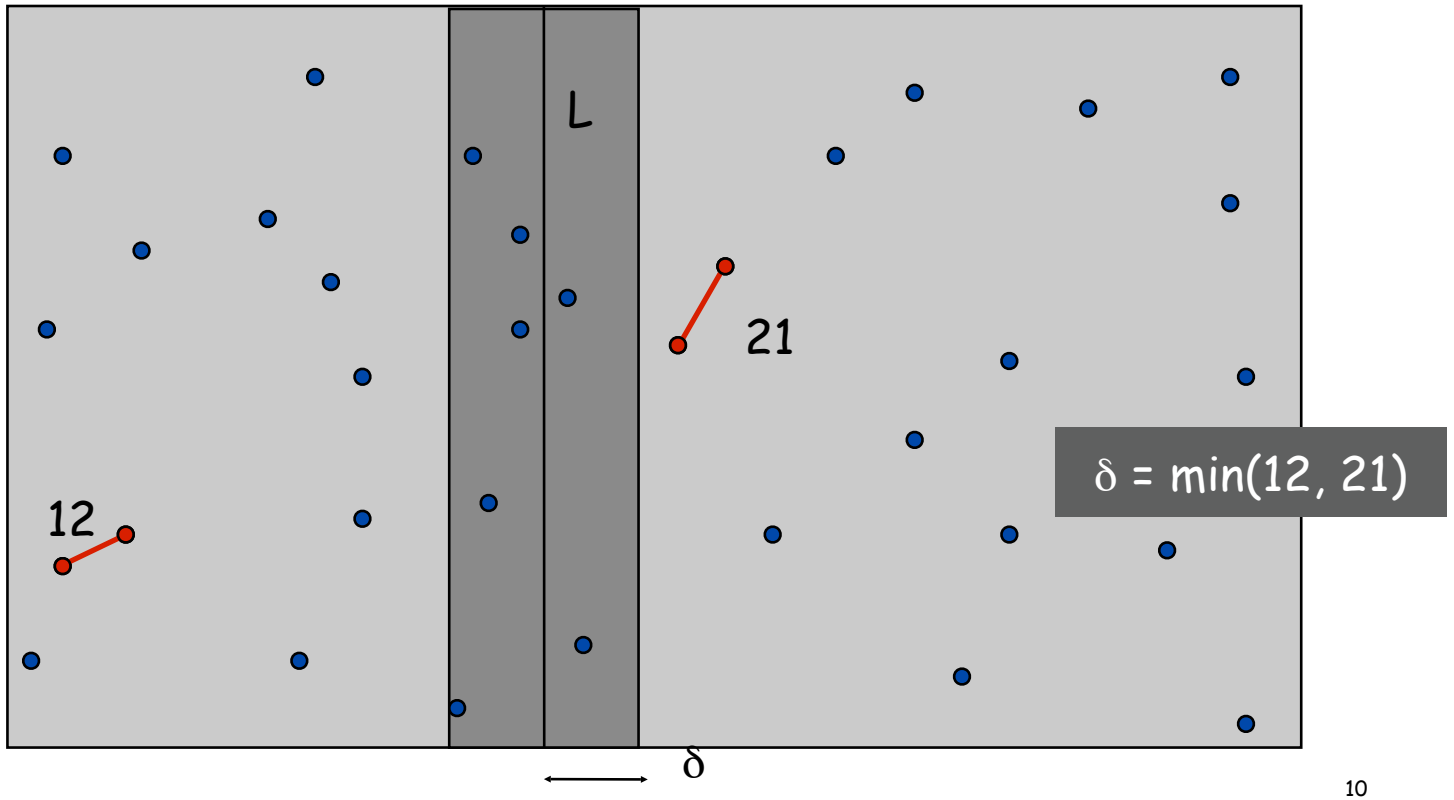
Find closest pair with one point in each side, assuming that distance < δ.



Closest Pair of Points

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

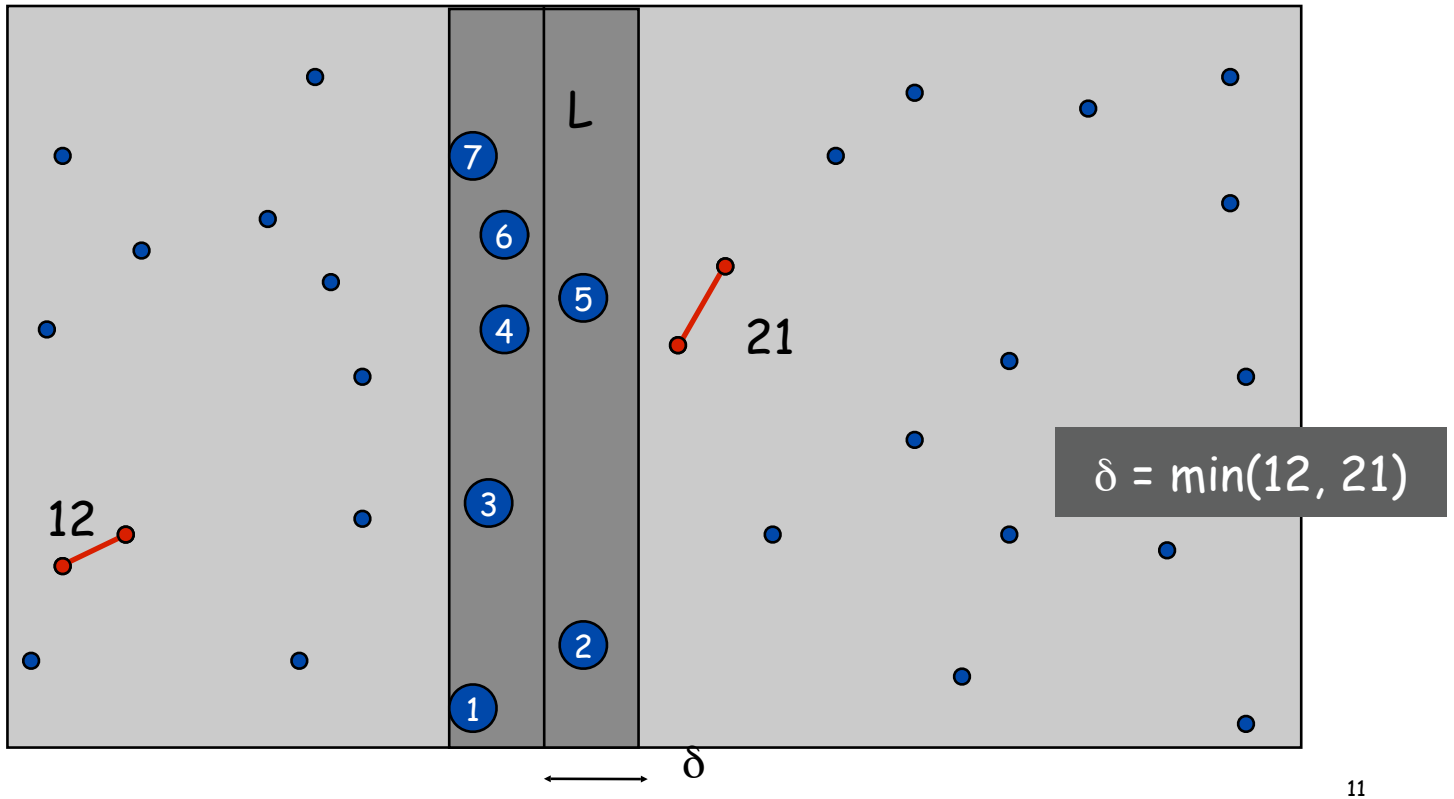
- Observation: only need to consider points within δ of line L.



Closest Pair of Points

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

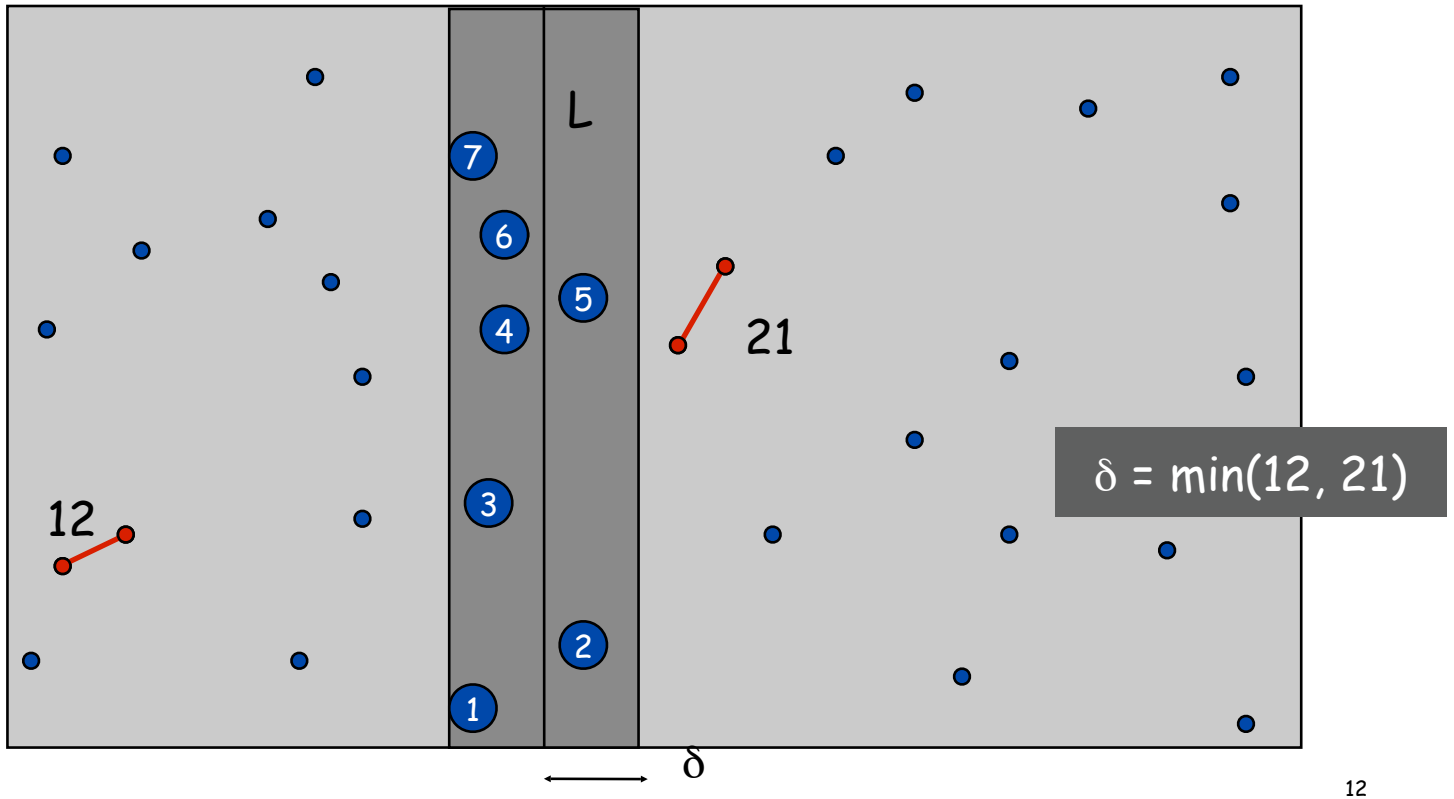
- Observation: only need to consider points within δ of line L.
- Sort points in 2δ-strip by their y coordinate.



Closest Pair of Points

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!



Closest Pair of Points

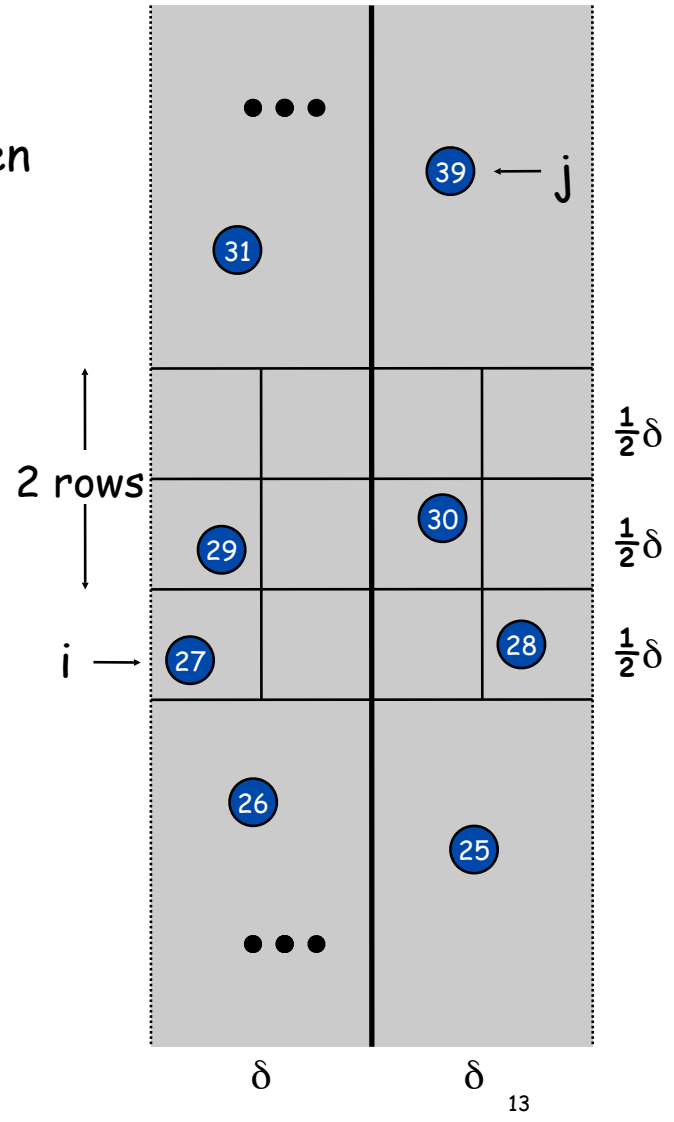
Def. Let s_i be the point in the 2δ-strip, with the i^{th} smallest y-coordinate.

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

Pf.

- 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 $\geq 2(\frac{1}{2}\delta)$. ▪

Fact. Still true if we replace 12 with 7.



Closest Pair Algorithm

```
Closest-Pair(p_1, …, p_n) {
   Compute separation line L such that half the points
   are on one side and half on the other side.

   δ_1 = Closest-Pair(left half)
   δ_2 = Closest-Pair(right half)
   δ   = min(δ_1, δ_2)

   Delete all points further than δ from separation line L

   Sort remaining points by y-coordinate.

   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.

   return δ.
}
```

$O(n \log n)$

$2T(n / 2)$

$O(n)$

$O(n \log n)$

$O(n)$

Closest Pair of Points: Analysis

Running time.

$$T(n) \leq 2T(n/2) + O(n \log n) \Rightarrow T(n) = O(n \log^2 n)$$

Q. Can we achieve O(n log n)?

A. Yes. Don't sort points in strip from scratch each time.

- Each recursive returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.

$$T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$$

O(n log n) Closest Pair Algorithm

```
Closest-Pair(p1, …, pn) {
   Compute separation line L such that half the points
   are on one side and half on the other side.                    O(n)

   (xsorted1, ysorted1, δ1) = Closest-Pair(left half)             2T(n / 2)
   (xsorted2, ysorted2, δ2) = Closest-Pair(right half)
   δ  = min(δ1, δ2)

   Merge xsorted1 and xsorted2 into xsorted                       O(n)
   Merge ysorted1 and ysorted2 into ysorted

   Scan points in y-order and compare distance between
   each point and next 11 neighbors. If any of these              O(n)
   distances is less than δ, update δ.

   return (xsorted, ysorted, δ).
}
```

Master method for solving recurrence relations

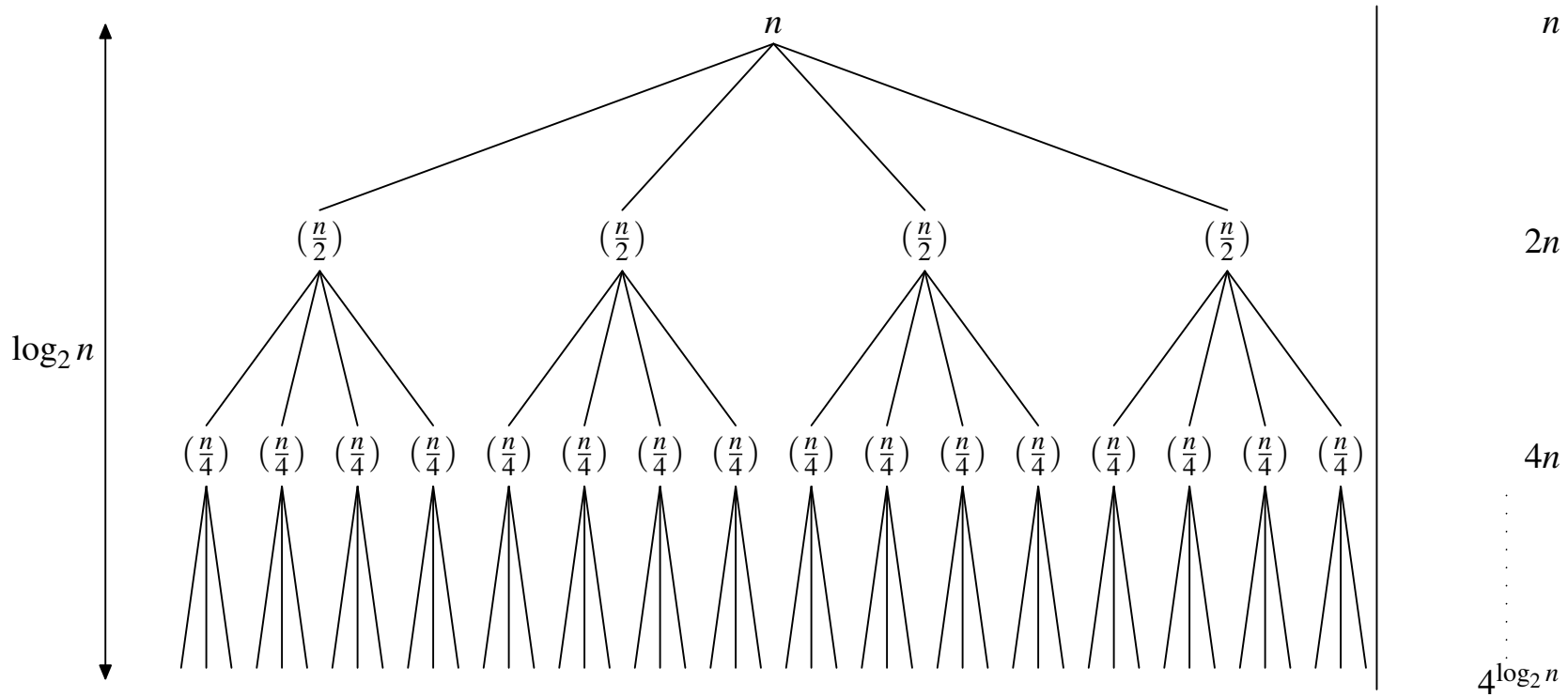
Master method idea

A recurrence tree will, in general, have either

- Time dominated by cost of the leaves
- Time dominated by cost of the root
- Time evenly distributed across the levels of the tree

Time dominated by leaves

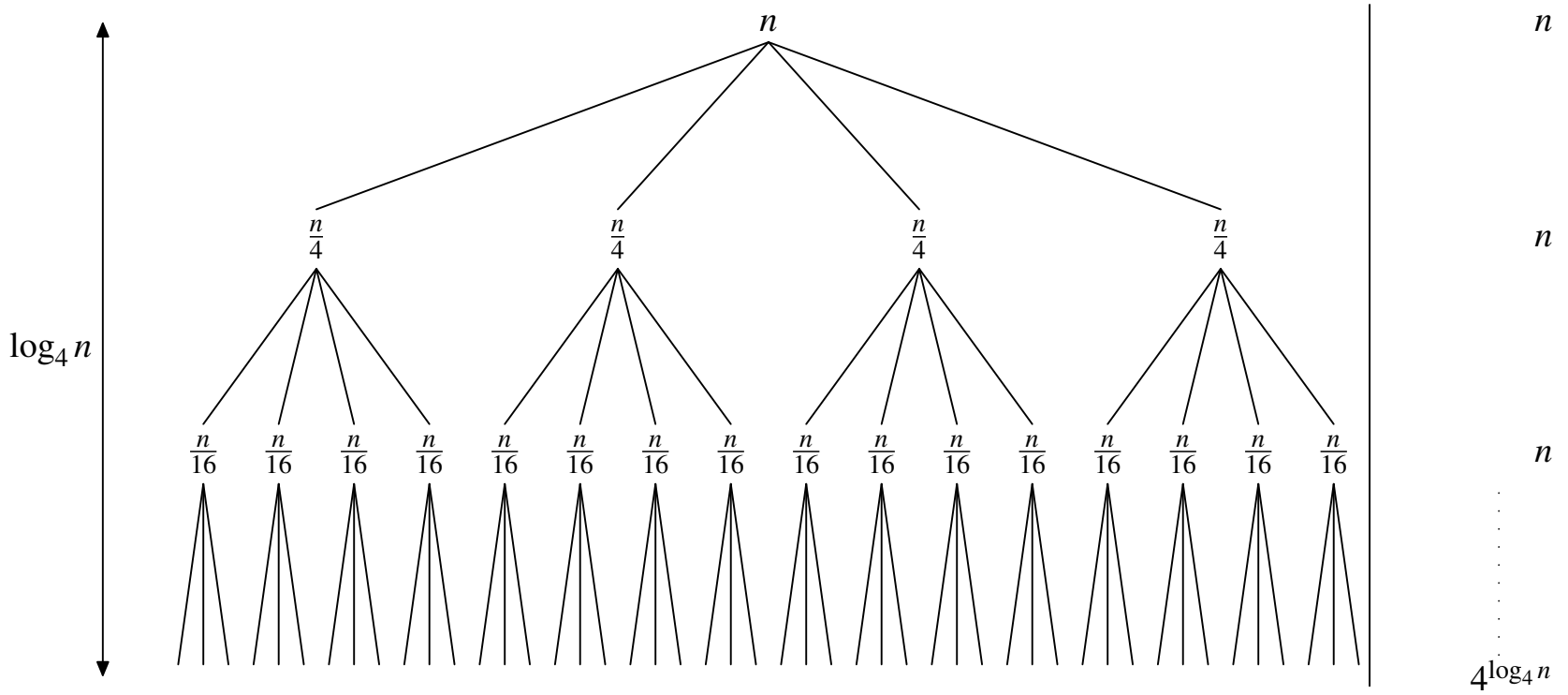
$$T(n) = 4T(\lfloor \tfrac{n}{2} \rfloor) + n$$



$$T(n) = \Theta(n^2)$$

Time evenly distributed across levels

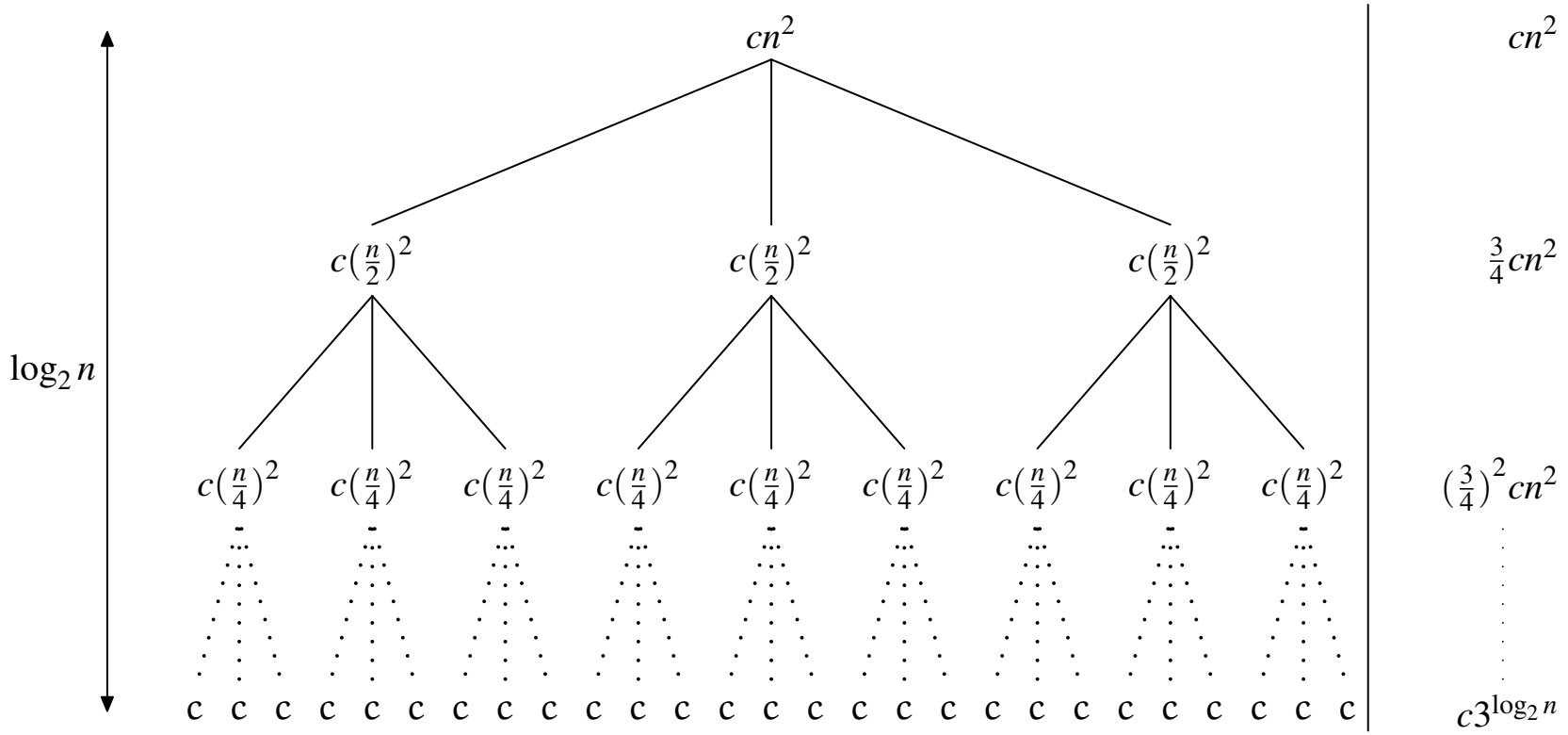
$$T(n) = 4T(\lfloor \tfrac{n}{4} \rfloor) + n$$



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

Time dominated by root

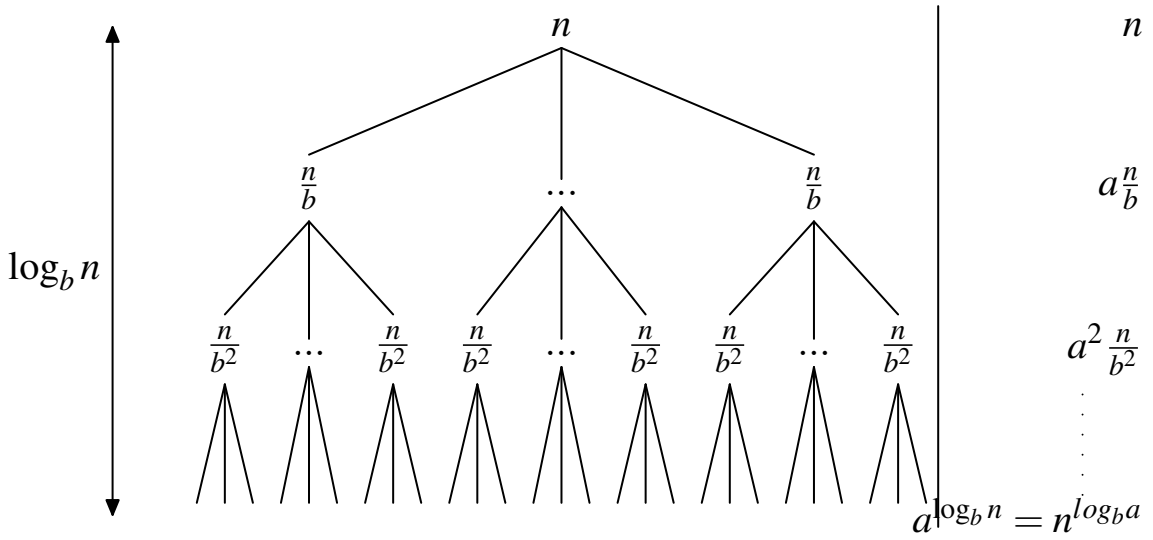
$$T(n) = 3T(\lfloor \tfrac{n}{2} \rfloor) + O(n^2)$$



$$T(n) = \Theta(n^2)$$

Limited master method

Recurrence: $T(n) = aT(n/b) + n$



$$T(n) = n + \frac{a}{b}n + (\frac{a}{b})^2 n + \ldots + a^{\log_b n}$$

$$= \sum_{i=0}^{\log_b n} (\frac{a}{b})^i n = n \sum_{i=0}^{\log_b n} (\frac{a}{b})^i$$

Limited master method

Evaluate: $T(n) = n\sum_{i=0}^{\log_b n}(\frac{a}{b})^i$

1. $\frac{a}{b} > 1$: Time dominated by leaves. $T(n) = n\sum_{i=0}^{\log_b n}(\frac{a}{b})^i = n\frac{(\frac{a}{b})^{\log_b(n+1)}-1}{\frac{a}{b}-1} = O(n(\frac{a}{b})^{\log_b n}) = O(n\frac{a^{\log_b n}}{b^{\log_b n}}) = O(a^{\log_b n}) = O(n^{\log_b a})$
2. $\frac{a}{b} = 1$: Time evenly distributed across all levels. $T(n) = n\log n = O(n\log n)$.
3. $\frac{a}{b} < 1$: Time dominated by root. $n\sum_{i=0}^{\log_b n}(\frac{a}{b})^i < n\frac{1}{1-\frac{a}{b}} = O(n)$.

Limited master method

Examples

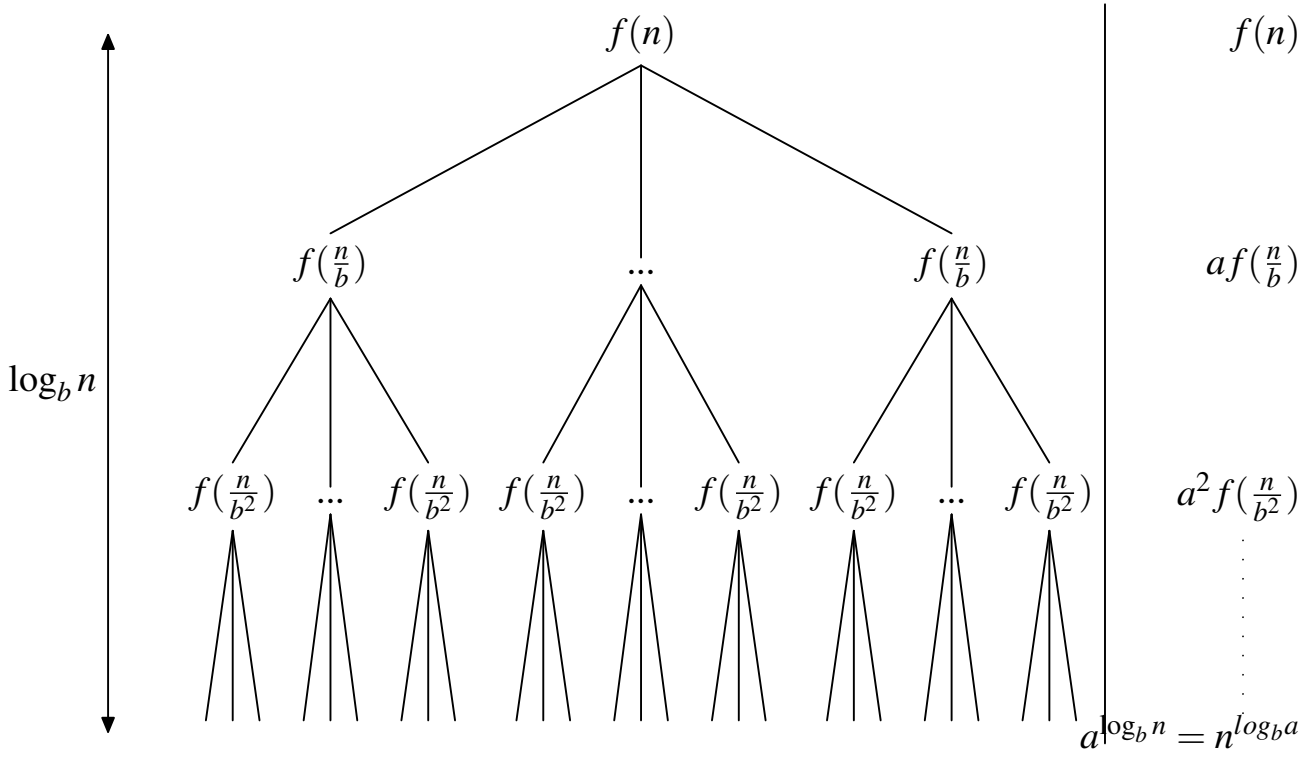
- $T(n) = 3T(\lfloor \frac{n}{2} \rfloor) + n$: $\frac{a}{b} = \frac{3}{2} > 1$.
  $T(n) = O(n^{log_2 3}) = O(n^{1.6})$

- $T(n) = 4T(\lfloor \frac{n}{4} \rfloor) + n$: $\frac{a}{b} = 1$. $T(n) = O(n \log n)$

- $T(n) = 4T(\lfloor \frac{n}{5} \rfloor) + n$. $\frac{a}{b} < 1$. $T(n) = O(n)$

Master Method

Recurrence: $T(n) = aT(n/b) + f(n)$

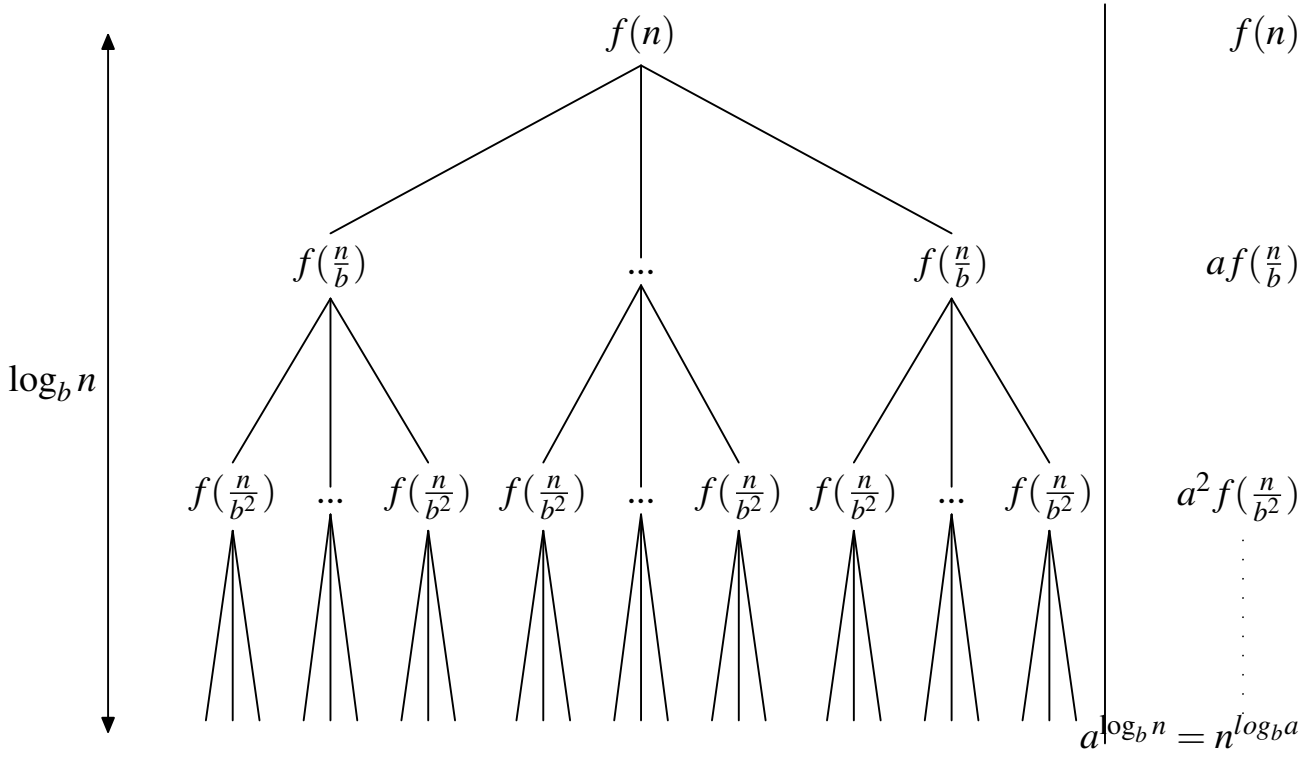


Case 1: f(n) polynomially smaller than $n^{\log_b a}$ (Lots of tiny subproblems) $f(n) = O(n^{\log_b a - \varepsilon})$

$$T(n) = \Theta(a^{\log_b n}) = \Theta(n^{\log_b a}) = \Theta(n^{\frac{\log a}{\log b}})$$

Master Method

Recurrence: $T(n) = aT(n/b) + f(n)$

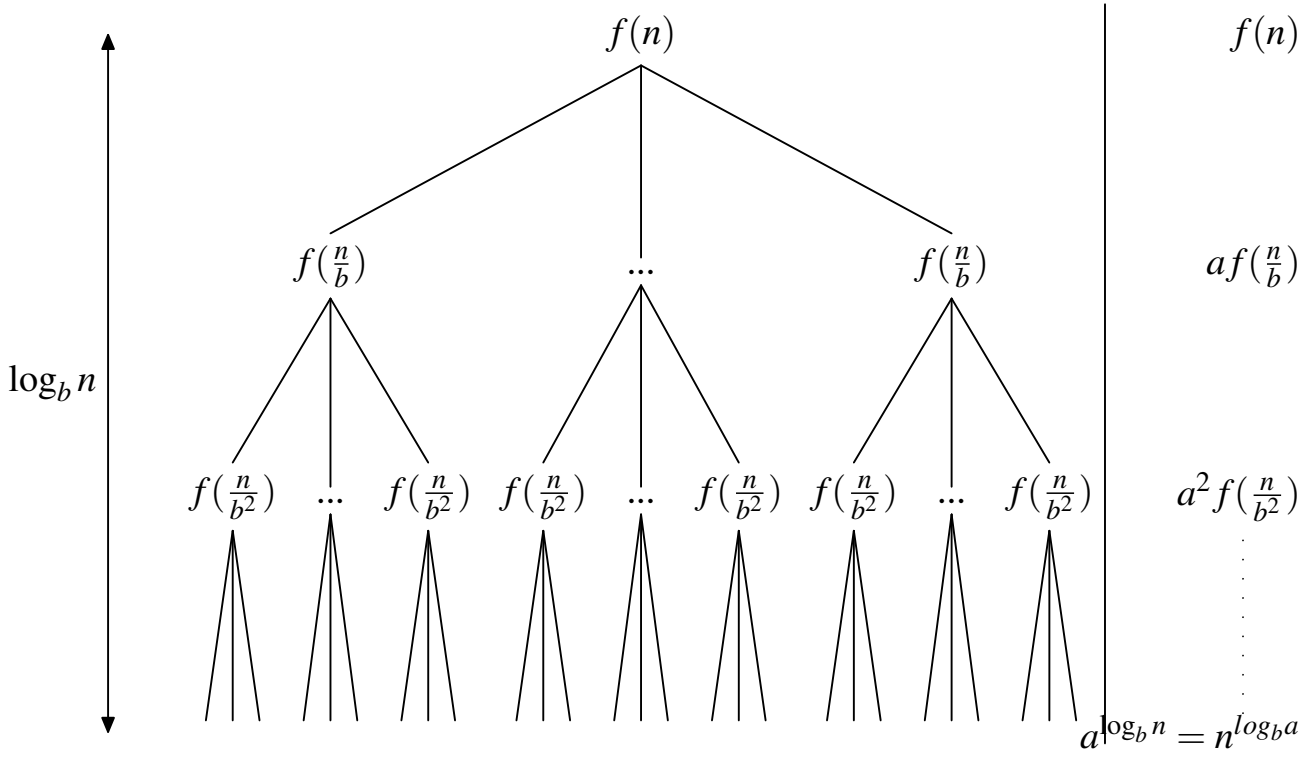


Case 2: $f(n) = \Theta(n^{\log_b a})$ (All levels about the same)

$$T(n) = \log_b n \Theta(n^{\log_b a})$$

Master Method

Recurrence: $T(n) = aT(n/b) + f(n)$



Case 3: f(n) polynomially larger than $n^{\log_b a}$ (Big step is most expensive) $f(n) = \Omega(n^{\log_b a + \varepsilon})$

$$T(n) = \Theta(f(n))$$

Master Method

Additional requirement:

- For case 3, also need regularity: there exists c > 1 such that for sufficiently large n: f(n) ≥ c•af(n/b). That is, each lower level is at least a constant fraction smaller than the higher level. Actually, this regularity implies polynomially larger, so case 3 need meet only this requirement.

  Polynomials or $f(n) = cn^k \log^j n$ are regular.

Master Method

Examples

- $T(n) = 3T(\frac{n}{2}) + O(n^2)$: $f(n) = \Omega(n^{log_2 3 + .2})$
  $T(n) = O(n^2)$

- $T(n) = 3T(\frac{n}{2}) + n^{1.4}$: $f(n) = O(n^{\log_2 3 - .1})$
  $T(n) = O(n^{\log_2 3})$.

- $T(n) = 16T(\frac{n}{4}) + 10n^2 + 3n + 5$:
  $f(n) = \Theta(n^{\log_4 16})$. $T(n) = O(n^2 \log n)$

5.5 Integer Multiplication

Integer Arithmetic

Add. Given two n-digit integers a and b, compute a + b.

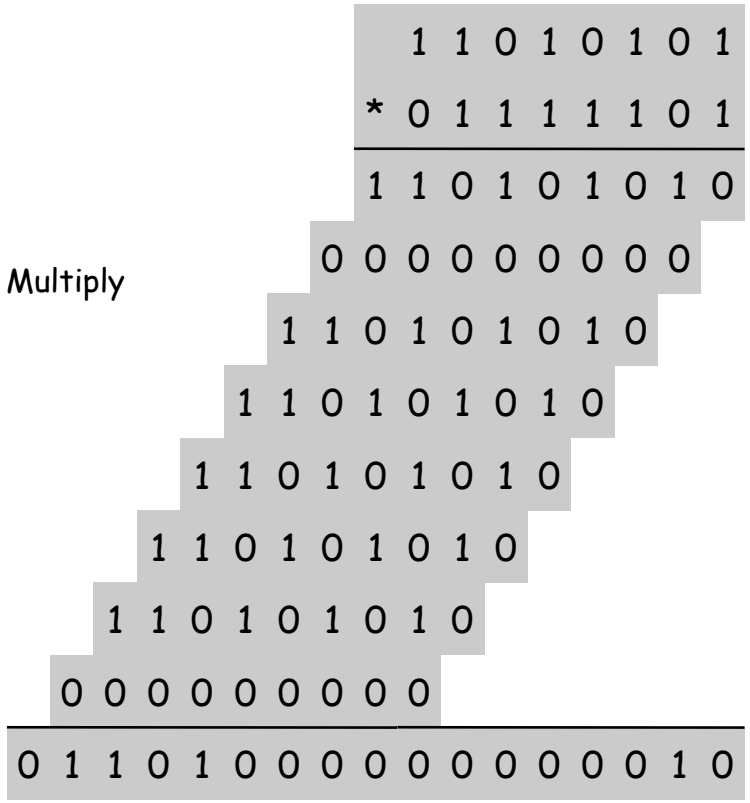
- O(n) bit operations.

Multiply. Given two n-digit integers a and b, compute a × b.

- Brute force solution: $\Theta(n^2)$ 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

Add



Multiply

Divide-and-Conquer Multiplication: Warmup

To multiply two n-digit integers:

- Multiply four ½n-digit integers.
- Add two ½n-digit integers, and shift to obtain result.

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

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

assumes n is a power of 2

Master method: $\log_b a = \log_2 4 = 2$

$f(n) = O(n^{2-1})$. Case 1:

Karatsuba Multiplication

To multiply two n-digit integers:

- Add two $\frac{1}{2}$n digit integers.
- Multiply three $\frac{1}{2}$n-digit integers.
- Add, subtract, and shift $\frac{1}{2}$n-digit integers to obtain result.

$$
\begin{aligned}
x &= 2^{n/2} \cdot x_1 + x_0 \\
y &= 2^{n/2} \cdot y_1 + y_0 \\
xy &= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \\
&= 2^n \cdot x_1 y_1 + 2^{n/2} \cdot \big((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0 \big) + x_0 y_0 \\
&\qquad\quad A \qquad\qquad\quad B \qquad\qquad A \quad C \qquad C
\end{aligned}
$$

Theorem. [Karatsuba-Ofman, 1962] Can multiply two n-digit integers in $O(n^{1.585})$ bit operations.

$$
\mathrm{T}(n) \le \underbrace{T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + T(1 + \lceil n/2 \rceil)}_{\text{recursive calls}} + \underbrace{\Theta(n)}_{\text{add, subtract, shift}}
$$

$$
\Rightarrow \mathrm{T}(n) = O(n^{\log_2 3}) = O(n^{1.585})
$$

Karatsuba: Master Method

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

$\log_b a = \log_2 3 =$ (roughly) 1.585

$f(n) = n$

$f(n) = O(n^{1.585 - .1})$ (polynomially smaller)

$T(n) = \theta(n^{1.585})$

Karatsuba: Recursion Tree

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 3T(n/2) + n & \text{otherwise} \end{cases}$$

$$T(n) = \sum_{k=0}^{\log_2 n} n \left(\tfrac{3}{2}\right)^k = \frac{\left(\frac{3}{2}\right)^{1+\log_2 n} - 1}{\frac{3}{2} - 1} = 3n^{\log_2 3} - 2$$