COMP 355
Advanced Algorithms

Divide and Conquer: Closest Pair & Integer Multiplication
KT: 5.4-5.5
Divide-and-Conquer

• Divide-and-conquer.
  – **Divide**: Break up problem into several parts.
  – **Conquer**: Solve each part recursively.
  – **Combine**: Merge solutions to sub-problems into overall solution.

• Most common usage.
  – Break up problem of size n into two equal parts of size \( \frac{1}{2}n \).
  – Solve two parts recursively.
  – Combine two solutions into overall solution in **linear time**.

• Consequence.
  – Brute force: \( n^2 \).
  – Divide-and-conquer: \( n \log n \).
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.

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.
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.
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.\[ \text{seems like } \Theta(n^2) \]
- Return best of 3 solutions.
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance < $\delta$.

$\delta = \min(12, 21)$
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance $< \delta$.

- Observation: only need to consider points within $\delta$ of line $L$.

$\delta = \min(12, 21)$
Closest Pair of Points

Find closest pair with one point in each side, **assuming that distance < \( \delta \).**
- Observation: only need to consider points within \( \delta \) of line \( L \).
- Sort points in \( 2\delta \)-strip by their \( y \) coordinate.

\[ \delta = \min(12, 21) \]
Closest Pair of Points

Find closest pair with one point in each side, assuming that distance $< \delta$.

- Observation: only need to consider points within $\delta$ of line $L$.
- Sort points in $2\delta$-strip by their y coordinate.
- Only check distances of those within 11 positions in sorted list!

$\delta = \min(12, 21)$
Closest Pair of Points

**Def.** Let $s_i$ be the point in the $2\delta$-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 $\delta$.

**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

Basis: If $|P| \leq 3$, then just solve the problem by brute force in $O(1)$ time.

Divide: Otherwise, partition the points into two subarrays $P_L$ and $P_R$ based on their $x$-coordinates. In particular, imagine a vertical line $\ell$ that splits the points roughly in half.

Conquer: Compute the closest pair within each of the subsets $P_L$ and $P_R$ each, by invoking the algorithm recursively. Let $\delta_L$ and $\delta_R$ be the closest pair distances in each case (see Fig. 35). Let $\delta = \min(\delta_L, \delta_R)$.

Combine: Note that $\delta$ is not necessarily the final answer, because there may be two points that are very close to one another but are on opposite sides of $\ell$. To complete the algorithm, we want to determine the closest pair of points between the sets, that is, the closest points $p \in P_L$ and $q \in P_R$ (see Fig. 35). Since we already have an upper bound $\delta$ on the closest pair, it suffices to solve the following restricted problem: if the closest pair $(p, q)$ are within distance $\delta$, then we will return such a pair, otherwise, we may return any pair. (This restriction is very important to the algorithm’s efficiency.) In the next section, we’ll show how to solve this restricted problem in $O(n)$ time. Given the closest such pair $(p, q)$, let $\delta' = \|pq\|$. We return $\min(\delta, \delta')$ as the final result.
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.

    \[ \delta_1 = \text{Closest-Pair(left half)} \]
    \[ \delta_2 = \text{Closest-Pair(right half)} \]
    \[ \delta = \min(\delta_1, \delta_2) \]

    Delete all points further than \( \delta \) 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 \( \delta \), update \( \delta \).

    return \( \delta \).
}
Closest Pair of Points: Analysis

Running time.

\[ T(n) \leq 2T(n/2) + O(n \log n) \implies T(n) = O(n \log^2 n) \]

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.

\[ T(n) \leq 2T(n/2) + O(n) \implies T(n) = O(n \log n) \]
**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 $a \times b$.
- Brute force solution: $\Theta(n^2)$ bit operations.
To multiply two n-digit integers:

- Multiply four \(\frac{1}{2}n\)-digit integers.
- Add two \(\frac{1}{2}n\)-digit integers, and shift to obtain result.

\[
x = 2^{n/2} \cdot x_1 + x_0 \\
y = 2^{n/2} \cdot y_1 + y_0 \\
xy = (2^{n/2} \cdot x_1 + x_0)(2^{n/2} \cdot y_1 + y_0) = 2^n \cdot x_1y_1 + 2^{n/2} \cdot (x_1y_0 + x_0y_1) + x_0y_0
\]

\[
T(n) = 4T\left(\frac{n}{2}\right) + \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n^2)
\]

assumes \(n\) is a power of 2
To multiply two n-digit integers:
- Add two ½n digit integers.
- Multiply three ½n-digit integers.
- Add, subtract, and shift ½n-digit integers to obtain result.

\[ x = 2^{n/2} \cdot x_1 + x_0 \]
\[ y = 2^{n/2} \cdot y_1 + y_0 \]
\[ xy = 2^n \cdot x_A y_1 + 2^{n/2} \cdot (x_1 y_0 + x_0 y_1) + x_0 y_0 \]
\[ = 2^n \cdot x_1 y_1 + 2^{n/2} \cdot ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0) + x_0 y_0 \]

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

\[ T(n) \leq T\left(\lfloor n/2 \rfloor \right) + T\left(\lceil n/2 \rceil \right) + T\left(1+\lceil n/2 \rceil \right) + \Theta(n) \]
\[ \Rightarrow T(n) = O(n^{\log_2 3}) = O(n^{1.585}) \]
Karatsuba: Recursion Tree

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

\[ T(n) = \sum_{k=0}^{\log_2 n} n \left( \frac{3}{2} \right)^k = \frac{ \left( \frac{3}{2} \right)^{1+\log_2 n} - 1 }{ \frac{3}{2} - 1 } = 3n^{\log_2 3} - 2 \]
Next Time

Dynamic Programming: Weighted Interval Scheduling