**CS 580: Algorithm Design and Analysis**

**Recap: Divide and Conquer**

**Recursive Approach:**
1. Divide input into smaller parts (Divide)
2. Solve each smaller instance (Conquer)
3. Combine solutions from each smaller instance (Merge)

**Example:** Merge Sort (Sort list of n items in $O(n \log n)$ time)
1. Divide array into two equal size parts ($n/2$)
2. Sort each sub-array (Conquer)
3. Merge the each sub-array to obtain the sorted list

**Recurrence Relationships**
- Useful to express the running time of recursive algorithms
- Analyzing a Recurrence: Unrolling, Telescoping, Induction,...
- Master's Theorem ($T(n)=a T(n/b) + n^c$)

**Applications**
- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the web.
- Nonparametric statistics (e.g., Kendall's Tau distance).

**5.3 Counting Inversions**

**Recall:**
- A pair $(i,j)$ is an inversion if $i < j$ but $S[i] > S[j]$.

**Counting Inversions:**

- **Brute force:** Check all $n(n-1)/2$ pairs $(i,j)$.
- **Divide-and-Conquer:**

**Applications**
- Voting theory.
- Collaborative filtering.
- Measuring the "sortedness" of an array.
- Sensitivity analysis of Google's ranking function.
- Rank aggregation for meta-searching on the web.
- Nonparametric statistics (e.g., Kendall's Tau distance).
Counting Inversions: Divide-and-Conquer

**Divide-and-Conquer**
- **Divide**: separate list into two pieces.
  - Example:
    - Original list: $1, 5, 4, 8, 10, 2, 6, 9, 12, 11, 3, 7$
    - Divide: $O(1)$
- **Conquer**: recursively count inversions in each half.
  - Conquer: $2T(n/2)$
  - Example:
    - Divide:
      - List: $1, 5, 4, 8, 10, 2, 6, 9, 12, 11, 3, 7$
      - Divide: $O(1)$
    - Conquer:
      - Split into two halves:
        - $1, 5, 4, 8, 10, 2$ and $6, 9, 12, 11, 3, 7$
      - Recursively count inversions in each half:
        - $T(5), T(6), T(4), T(8), T(2), T(10), T(6), T(9), T(11), T(3), T(7)$
      - Combine:
        - $r = r_1 + r_2 + r_3$
        - $r_1$: $5$ blue-blue inversions
        - $r_2$: $8$ green-green inversions
        - $r_3$: $9$ blue-green inversions
        - Total: $5 + 8 + 9 = 22$
- **Combine**: count inversions where $a_i$ and $a_j$ are in different halves, and return sum of three quantities.
  - Combine: $O(n)$

Counting Inversions: Implementation

**Pre-condition**: Merge-and-Count $A$ and $B$ are sorted.
**Post-condition**: Sort-and-Count $L$ is sorted.

**Sort-and-Count(L)**
- if list $L$ has one element
  - return $0$ and the list $L$
- Divide the list into two halves $A$ and $B$
  - $(r_A, A) \leftarrow \text{Sort-and-Count}(A)$
  - $(r_B, B) \leftarrow \text{Sort-and-Count}(B)$
- $(r_B, L) \leftarrow \text{Merge-and-Count}(A, B)$
- return $r = r_A + r_B + r$ and the sorted list $L$

5.4 Closest Pair of Points
Closest Pair of Points

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 \( O(n^2) \) 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.

**Algorithm.**
- Draw vertical line \( L \) so that roughly \( n/4 \) points on each side.
- Find closest pair in each side recursively.
- Find closest pair with one point in each side. Return best of 3 solutions.
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!

Fact. Still true if we replace 12 with 7.
5.5 Integer Multiplication

Motivation: Complex Multiplication

Complex multiplication: 
\((a + bi)(c + di) = x + yi\).
Grade-school: 
\[ x = ac - bd, \quad y = bc + ad. \]
4 multiplications, 2 additions

Q. Is it possible to do with fewer multiplications?

A. Yes. [Gauss]

\[ x = ac - bd, \quad y = (a + b)(c + d) - ac - bd. \]
3 multiplications, 5 additions ($\$305$)

Remark. Improvement if no hardware multiply.

Integer Addition

Addition. Given two $n$-bit integers $x$ and $y$, compute $x + y$.
Grade-school: $\mathcal{O}(n)$ bit operations.

\[
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Remark. Grade-school addition algorithm is optimal.

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.

\[
T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)
\]
**Integer Multiplication**

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

**Grade-school**

6($n$) bit operations.

---

**Divide-and-Conquer Multiplication: Warmup**

To multiply two $n$-bit integers $x$ and $y$:

- Multiply four $\frac{n}{2}$-bit integers, recursively.
- Add and shift to obtain result.

Ex. $x = 10001101$, $y = 11100001$

$$T(n) = 4T(\frac{n}{2}) + n$$

**Master’s Theorem:**

$a = 4$, $b = 2$, $c = 1$

$(2 \log_2 n - 1) > 1$, $O(n^{\log_2(4)}) = O(n^2)$

---

**Karatsuba Multiplication**

To multiply two $n$-bit integers $x$ and $y$:

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

Ex. $x = 10001101$, $y = 11100001$

$$xy = 2^2 \cdot x_3 y_3 + 2^1 \cdot (x_2 y_3 + x_3 y_2) + 2^0 \cdot (x_2 y_2 + x_3 y_3)$$

Theorem. (Karatsuba-Ofman 1962) Can multiply two $n$-bit integers in $O(n^{\log_2(3)})$ bit operations.

**Recursion Tree**

---

Master’s Theorem: $a = 3$, $b = 2$, $c = 1$

$2^\log_2(3) - 1 > 1$, $O(n^{\log_2(3)}) = O(n^1.585)$
Karatsuba: Recursion Tree

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

\[
T(n/2^k) \quad \ldots
\]

Fast Integer Division Too (!)

**Integer division.** Given two \(n\)-bit (or less) integers \(a\) and \(b\), compute quotient \(q = \lfloor a / b \rfloor\) and remainder \(r = a \mod b\) (such that \(a = bq + r\)).

**Fact.** Complexity of integer division is (almost) same as integer multiplication.

To compute quotient \(q = \lfloor a / b \rfloor\)
- **Approximate** \(a = 1 / y\) using Newton's method
  - After \(i\) iterations, either \(y_i \approx a / x\) or \(\lfloor a / x \rfloor < y_i\)
  - **Otherwise** \(y_i = \lfloor a / x \rfloor\) (1 multiplication)
  - **Total:** \(O(\log n)\) multiplications and subtractions

**Remark.** Grade-school dot product algorithm is optimal.

Matrix Multiplication

**Dot product.** Given two length \(n\) vectors \(a\) and \(b\), compute
\[
c = a \cdot b = \sum_{i=1}^{n} a_i b_i
\]

**Grade-school.** \(\Theta(n^2)\) arithmetic operations.

**Schönhage–Strassen algorithm**
\[
O(n \log n \log \log n)
\]

Only used for really big numbers: \(a \gg 2^{215}\)

State of the Art: \(O(n \log n \log \log n)\) for increasing small
\(g(n) = \log \log n\)

**Remark.** Grade-school dot product algorithm is optimal.
Fast Matrix Multiplication

To multiply two $n$-by-$n$ matrices $A$ and $B$, compute $C = AB$.

Divide: partition $A$ and $B$ into $\lceil n/2 \rceil$-by-$\lceil n/2 \rceil$ blocks.

Conquer: multiply $7$ pairs of $\lceil n/2 \rceil$-by-$\lceil n/2 \rceil$ matrices, recursively.

Combine: add appropriate products using $6$ matrix additions.

\[
C = \left( \begin{array}{ccc}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{array} \right) = 
\left( \begin{array}{ccc}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{array} \right) \left( \begin{array}{ccc}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{array} \right)
\]

Apply Master Theorem ($a = 7, b = 2, c = 2$)

\[
T(n) = 7T(n/2) + \Theta(n^2)
\]

Numerical stability.

Matrix Multiplication: Practice

Implementation issues.
- Sparsity.
- Caching effects.
- Numerical stability.
- Odd matrix dimensions.
- Crossover to classical algorithm around $n = 128$.

Common misconception. "Strassen is only a theoretical curiosity."

Apple reports 4x speedup on G4 Velocity Engine when
\[ n = 2,550 \]

Range of instances where it’s useful is a subject of controversy.

Remark. Can “Strassenize” $A = b$, determinant, eigenvalues, SVD, ….
Fast Matrix Multiplication: Theory

Multiply two 2-by-2 matrices with 7 scalar multiplications?
A. Yes! [Strassen 1969]

Multiply two 2-by-2 matrices with 6 scalar multiplications?
A. Impossible. [Hopcroft and Kerr 1971]

Two 2-by-2 matrices with 21 scalar multiplications?
A. Also impossible.

Begun, the decimal wars have. [Pan, Bini et al, Schönhage, ...]
- Two 2n-by-2n matrices with 4.60n scalar multiplications.
- Two 4n-by-4n matrices with 4.217n scalar multiplications.
- A year later.

Best known. \( O(n^{2.376}) \) [Coppersmith-Winograd, 1987]

Conjecture. \( O(n^{2+\epsilon}) \) for any \( \epsilon > 0 \).

Caveat. Theoretical improvements to Strassen are progressively less practical.

Best known. \( O(n^{2.373}) \) [Williams, 2014]

Conjecture. \( O(n^{2+\epsilon}) \) for any \( \epsilon > 0 \).

Caveat. Theoretical improvements to Strassen are progressively less practical.

Best known. \( O(n^{2.3729}) \) [Le Gall, 2014]

Conjecture. \( O(n^{2+\epsilon}) \) for any \( \epsilon > 0 \).

Caveat. Theoretical improvements to Strassen are progressively less practical.