Counting inversions

**Similarity metric:** number of inversions between two rankings

- Ranking $R_1$: 1, 2, ..., $n$
- Ranking $R_2$: $a_1, a_2, ..., a_n$.
- Items $i$ and $j$ inverted if $i < j$, but $a_i > a_j$

**Goal**

Count the number of inversions given two rankings

- Brute-force algorithm has complexity $O(n^2)$

Applications in data mining, voting theory, web search, ...

- For example, rankings could represent musical preferences or voting preferences...
Divide and conquer

- Divide the array into two halves: left and right
  - Takes $O(1)$ time
- Count the number of inversions $n_1$ and $n_2$ in the two halves
  - $T(n/2)$ time
- Return $n_1 + n_2 + n'$, where $n'$ is the number of inversions where one item is in the left subarray, and the other is in the right subarray
Divide and conquer

- Divide the array into two halves: left and right
  - Takes $O(1)$ time
- Count the number of inversions $n_1$ and $n_2$ in the two halves
  - $T(n/2)$ time
- Apply the *merge* routine used in merge sort to count inversions across subarrays
  - Assumes the subarrays are sorted
procedure sort_and_count(L) =
    if L.Length = 1
        then return (0, L)
    else
        (A1, B1) := split(L)
        (p, A) := sort_and_count(A1)
        (q, B) := sort_and_count(B1)
        (r, L) := merge_and_count(A, B)
        return (p + q + r, L)
procedure merge_and_count(A, B) =
    a := List.head A
    b := List.head B
    if (a > b)
        (p, C) := merge_and_count (A, List.tail B)
        return (p + A.length, b :: C)
    else
        (p, C) := merge_and_count (List.tail A, B)
        return (p, a :: C)

- **Precondition for** Merge-and-Count: A and B are sorted.
- **Postcondition for** Sort-and-Count: L is sorted
- **Recurrence**: $T(n) = 2T(n/2) + n$
Exercise: Finding modes

You are given an array $A$ with $n$ entries. Each entry is a distinct number.

You are told that the sequence $A[1], \ldots, A[n]$ is unimodal. That is, for some index $p$ between 1 and $n$, values in the array increase up to position $p$ in $A$, and then decrease the rest of the way up to position $n$.

Give a $O(\log n)$-time algorithm to find the “peak entry” of the array.
Closest pair of points

Goal

Given $n$ points in the 2-D plane, find a pair with smallest Euclidean distance between them.

- Fundamental geometric primitive
Attempt 1

- Divide the plane into 4 quadrants
- Impossible to guarantee that points are spread equally
Closest pair of points

- **Divide:** Draw vertical line $L$ so that roughly $n/2$ points are on each side
Closest pair of points

- **Divide:** Draw vertical line $L$ so that roughly $n/2$ points are on each side.
- **Conquer:** Find closest pair in each side, recursively.

Closest pair of points

- **Divide:** Draw vertical line $L$ so that roughly $n/2$ points are on each side
- **Conquer:** Find closest pair in each side, recursively
- **Combine:** Find closest pair with one point in each side
  - A quadratic step?
- Return best of 3 solutions
Let $\delta$ be the smaller of the shortest distances computed by the recursive calls.

Find closest pair $(p, q)$ where $p$ and $q$ are on opposite sides of $L$, assuming that the distance between $p$ and $q$ is less than $\delta$. 

Let $\delta$ be the smaller of the shortest distances computed by the recursive calls.

Find closest pair $(p, q)$ where $p$ and $q$ are on opposite sides of $L$, assuming that the distance between $p$ and $q$ is less than $\delta$.

- Sort points in the $2\delta$-strip by their $y$-coordinate.
• Let $\delta$ be the smaller of the shortest distances computed by the recursive calls
• Find closest pair $(p, q)$ where $p$ and $q$ are on opposite sides of $L$, assuming that the distance between $p$ and $q$ is less than $\delta$
  • Sort points in the $2\delta$-strip by their $y$-coordinate
  • Only check distances of those within 11 positions in the sorted list!
Let $s_i$ be the point in the $2\delta$ strip with the $i$-th smallest coordinate.

If $|i - j| < 12$, then the distance between $i$ and $j$ is at least $12$.

- No two points lie in the same $(\delta/2) \times (\delta/2)$ box.
- Two points that are at least two rows apart have distance $> 2(\delta/2)$.
method Closest-Pair(p₁, ..., pₙ) {
    Compute separation line \( L \) such that there are \( n/2 \) points on each 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 \( L \)
    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 \( \delta \)
}
\[ T(n) = 2T(n/2) + n \log n \quad \Rightarrow \quad T(n) = O(n(\log n)^2) \]
Where do you achieve $O(n \log n)$?

- Don’t sort points in strip from scratch each time.
- Each recursive call returns two lists: all points sorted by y coordinate, and all points sorted by x coordinate.
- Sort by merging two pre-sorted lists.
Addition

- **Goal:** Given two $n$-digit integers $a$ and $b$, compute $a + b$.
- $O(n)$ bitwise operations
### Addition
- **Goal:** Given two $n$-digit integers $a$ and $b$, compute $a + b$.
- $O(n)$ bitwise operations

### Multiplication
- **Goal:** Given two $n$-digit integers $a$ and $b$, compute $ab$.
- $O(n^2)$ bitwise operations?
To multiply two $n$-digit numbers:

- Multiply four $n/2$-digit numbers
- Add two $n/2$-digit numbers, and shift to obtain result

\[
x = 2^{n/2}x_1 + x_0
\]
\[
y = 2^{n/2}y_1 + y_0
\]
\[
xy = (2^{n/2}x_1 + x_0)(2^{n/2}y_1 + y_0)
\]
\[
= 2^n x_1 y_1 + 2^{n/2}(x_1 y_0 + x_0 y_1) + x_0 y_0
\]
Divide and conquer: Attempt 1

To multiply two \( n \)-digit numbers:

- Multiply four \( n/2 \)-digit numbers
- Add two \( n/2 \)-digit numbers, and shift to obtain result

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

\[
T(n) = 4T(n/2) + \Theta(n)
\]
To multiply two $n$-digit numbers:

- Multiply four $n/2$-digit numbers
- Add two $n/2$-digit numbers, and shift to obtain result

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

\[
T(n) = 4T(n/2) + \Theta(n) \\
T(n) = \Theta(n^2)
\]
Karatsuba’s multiplication

To multiply two \( n \)-digit numbers:

- Add two \( n/2 \)-digit numbers
- Multiply \textit{three} \( n/2 \)-digit numbers
- Add, subtract, and shift two \( n/2 \)-digit numbers

\[
\begin{align*}
x &= 2^{n/2}x_1 + x_0 \\
y &= 2^{n/2}y_1 + y_0 \\
xy &= (2^{n/2}x_1 + x_0)(2^{n/2}y_1 + y_0) \\
    &= 2^n x_1 y_1 + 2^{n/2}(x_1 y_0 + x_0 y_1) + x_0 y_0 \\
    &= 2^n x_1 y_1 + 2^{n/2} ((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0) + x_0 y_0
\end{align*}
\]
Karatsuba’s multiplication

To multiply two $n$-digit numbers:

- Add two $n/2$-digit numbers
- Multiply three $n/2$-digit numbers
- Add, subtract, and shift two $n/2$-digit numbers

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

\[
T(n) = 3T(n/2) + \Theta(n)
\]
Karatsuba’s multiplication

To multiply two \( n \)-digit numbers:

- Add two \( n/2 \)-digit numbers
- Multiply three \( n/2 \)-digit numbers
- Add, subtract, and shift two \( n/2 \)-digit numbers

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

\[
T(n) = 3T(n/2) + \Theta(n) \\
T(n) = \Theta(n^{\lg 3}) = \Theta(n^{1.585})
\]
Fast matrix multiplication

Goal

Given two $n \times n$ matrices $A$ and $B$, compute $C = AB$

$$
\begin{pmatrix}
  c_{11} & c_{12} & \ldots & c_{1n} \\
  c_{21} & c_{22} & \ldots & c_{2n} \\
  \vdots  & \vdots  & \ddots & \vdots  \\
  c_{n1} & c_{n2} & \ldots & c_{nn}
\end{pmatrix} =
\begin{pmatrix}
  a_{11} & a_{12} & \ldots & a_{1n} \\
  a_{21} & a_{22} & \ldots & a_{2n} \\
  \vdots  & \vdots  & \ddots & \vdots  \\
  a_{n1} & a_{n2} & \ldots & a_{nn}
\end{pmatrix}
\begin{pmatrix}
  b_{11} & b_{12} & \ldots & b_{1n} \\
  b_{21} & b_{22} & \ldots & b_{2n} \\
  \vdots  & \vdots  & \ddots & \vdots  \\
  b_{n1} & b_{n2} & \ldots & b_{nn}
\end{pmatrix}
$$

$$
c_{ij} = \sum_{i=1}^{n} a_{ik} b_{kj}
$$

Brute-force algorithm takes $O(n^3)$ time
Matrix multiplication: Attempt 1

Divide and conquer:

- Partition $A$ and $B$ into $n/2 \times n/2$ blocks
- Multiply eight $n/2 \times n/2$ recursively
- Add appropriate products using four matrix additions

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = (A_{11}B_{11}) + (A_{12}B_{21}) \quad C_{12} = (A_{11}B_{12}) + (A_{12}B_{22})
\]
\[
C_{21} = (A_{21}B_{11}) + (A_{22}B_{21}) \quad C_{22} = (A_{21}B_{12}) + (A_{22}B_{22})
\]
Matrix multiplication: Attempt 1

Divide and conquer:
- Partition \( A \) and \( B \) into \( n/2 \times n/2 \) blocks
- Multiply eight \( n/2 \times n/2 \) recursively
- Add appropriate products using four matrix additions

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix} =
\begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = (A_{11}B_{11}) + (A_{12}B_{21}) \quad C_{12} = (A_{11}B_{12}) + (A_{12}B_{22})
\]
\[
C_{21} = (A_{21}B_{11}) + (A_{22}B_{21}) \quad C_{22} = (A_{21}B_{12}) + (A_{22}B_{22})
\]

\[
T(n) = 8T(n/2) + \Theta(n^2)
\]

\[
T(n) = \Theta(n^3)
\]
Strassen’s algorithm

Key idea: Multiply block matrices using seven multiplications

\[
\begin{bmatrix}
  C_{11} & C_{12} \\
  C_{21} & C_{22}
\end{bmatrix} = \begin{bmatrix}
  A_{11} & A_{12} \\
  A_{21} & A_{22}
\end{bmatrix} \begin{bmatrix}
  B_{11} & B_{12} \\
  B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = (A_{11}B_{11}) + (A_{12}B_{21}) \\
C_{21} = (A_{21}B_{11}) + (A_{22}B_{21}) \\
C_{12} = (A_{11}B_{12}) + (A_{12}B_{22}) \\
C_{22} = (A_{21}B_{12}) + (A_{22}B_{22})
\]
Strassen’s algorithm

**Key idea:** Multiply block matrices using *seven* multiplications

\[
\begin{bmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{bmatrix}
= \begin{bmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{bmatrix}
\begin{bmatrix}
B_{11} & B_{12} \\
B_{21} & B_{22}
\end{bmatrix}
\]

\[
C_{11} = (A_{11}B_{11}) + (A_{12}B_{21})
\]

\[
C_{21} = (A_{21}B_{11}) + (A_{22}B_{21})
\]

\[
C_{11} = P_5 + P_4 - P_2 + P_6
\]

\[
C_{21} = P_3 + P_4
\]

\[
P_1 = A_{11}(B_{12} - B_{22})
\]

\[
P_3 = (A_{21} + A_{22})B_{11}
\]

\[
P_5 = (A_{11} + A_{22})(B_{11} + B_{22})
\]

\[
P_7 = (A_{11} - A_{21})(B_{11} + B_{12})
\]

\[
P_2 = (A_{11} + A_{12})B_{22}
\]

\[
P_4 = A_{22}(B_{21} - B_{11})
\]

\[
P_6 = (A_{12} - A_{22})(B_{21} + B_{22})
\]
\[ T(n) = 7T(n/2) + \Theta(n^2) \]
\[ T(n) = 7T(n/2) + \Theta(n^2) \]
\[ T(n) = \Theta(n^{\lg 7}) = O(n^{2.81}) \]
\[ T(n) = 7T(n/2) + \Theta(n^2) \]

\[ T(n) = \Theta(n^{\log_2 7}) = O(n^{2.81}) \]

**Practical issues:**

- Sparsity
- Caching effects
- Numerical stability
- Typically useful only for larger values of \( n \)
Complexity

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

\[
T(n) = \Theta(n^{\lg 7}) = O(n^{2.81})
\]

**Practical issues:**

- Sparsity
- Caching effects
- Numerical stability
- Typically useful only for larger values of \( n \)

**Best current bound**

\[ O(n^{2.3727}) \text{ [Williams 2011]} \]
Exercise: Exponentiation

Can you compute $a^n$, for given $a$ and $n$, in $o(n)$ steps?