CS 341: ALGORITHMS
Lecture 5: divide & conquer II
Readings: see website
Trevor Brown (co-taught with Anna Lubiw)
https://www.student.cs.uwaterloo.ca/~cs341
  trevor.brown@uwaterloo.ca

Problem: Non-Dominated Points
- A point dominates everything to the southwest

More Formally
- Given two points \((x_1, y_1)\) and \((x_2, y_2)\), we say \((x_1, y_1)\) dominates \((x_2, y_2)\) if \(x_1 > x_2\) and \(y_1 > y_2\)
- Input: a set \(S\) of \(n\) points
- Output: all non-dominated points in \(S\), i.e., all points in \(S\) that are not dominated by any point in \(S\)

Brute Force Algorithm
```python
1: def non_dominated_points(S):
2:     dominated[p] = false
3:     for q in S:
4:         if q.x > p.x and q.y > p.y
5:             dominated[p] = true
6:         if not dominated[p]
7:             print p
```

Running time: \(O(n^2)\)
Let's come up with a better algorithm

Problem Decomposition
- Suppose we sort the points in \(S\) with respect to their x-coordinates. This takes time \(O(n \log n)\).
PROBLEM DECOMPOSITION

Divide: Let the first $n/2$ points be denoted $S_1$ and the last $n/2$ points be denoted $S_2$.

Combine: Given the non-dominated points in $S_1$ and the non-dominated points in $S_2$, how do we find the non-dominated points in $S$?

Observe that no point in $S_1$ dominates a point in $S_2$.
Therefore we only need to eliminate the points in $S_1$ that are dominated by a point in $S_2$. It turns out that this can be done in time $O(n)$.

COMBINING TO GET NON-DOMINATED POINTS

- Let $Q_1, Q_2, \ldots, Q_k$ be the non-dominated points in $S_1$.
- Let $R_1, R_2, \ldots, R_m$ be the non-dominated points in $S_2$.

Just need to find rightmost $Q_i$ that is not dominated (that has $y$-coordinate $\geq R_j.y$).

Running time complexity?

Assume $n = 2^j$ for simplicity

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

Same as merge sort recurrence: $T(n) = \Theta(n \log n)$

So total for sort & recursion is $T(n) = \Theta(n \log n)$

Regardless of data structures…

either way doesn't matter
MULTIPRECISION MULTIPLICATION

Input: two \( k \)-bit positive integers \( X \) and \( Y \)
- With binary representations:
  \[ X = [X[k-1], \ldots, X[0]] \]
  \[ Y = [Y[k-1], \ldots, Y[0]] \]
Output: The \( 2k \)-bit positive integer \( Z = XY \)
- With binary representation:
  \[ Z = [Z[2k-1], \ldots, Z[0]] \]

Here, we are interested in the bit complexity of algorithms that solve Multiprecision Multiplication, which means that the complexity is expressed as a function of \( k \) (the size of the problem instance is \( 2k \) bits).

A DIVIDE-AND-CONQUER APPROACH

Let \( X_L \) be the integer formed by the \( k/2 \) high-order bits of \( X \) and let \( X_R \) be the integer formed by the \( k/2 \) low-order bits of \( X \).
Similarly for \( Y \).

\[ X = [X_L, X_R] \]
\[ Y = [Y_L, Y_R] \]

Thus \( X = 2^{k/2}X_L + X_R \) and \( Y = 2^{k/2}Y_L + Y_R \).

\[ Z = X \cdot Y = 2^{k/2}X_L \cdot Y_L + 2^{k/2}X_L \cdot Y_R + X_R \cdot Y_L + X_R \cdot Y_R \]

Suggests a D&C approach...
- Divide into four \( k/2 \)-bit multiplication subproblems
- Conquer with recursive calls
- Combine with \( k \)-bit addition and bit shifting

EXPRESSING \( k \)-BIT MULT. AS \( k/2 \)-BIT MULT.

\[ X = 2^{k/2}X_L + X_R \] and \( Y = 2^{k/2}Y_L + Y_R \)

So \( XY = (2^{k/2}X_L + X_R)(2^{k/2}Y_L + Y_R) \)

\[ = 2^kX_LY_L + 2^{k/2}(X_LY_R + X_RY_L) + X_RY_R \]

Assume \( k = 2^j \) for ease
\[ T(2^j) = 4T(2^{j-2}) + \Theta(k) \]

Master theorem says \( T(2^j) \in \Theta((k^{2(j-1)}) = \Theta(k^2)) \)

Recall: \( XY = 2^kX_LY_L + 2^{k/2}(X_LY_R + X_RY_L) + X_RY_R \)

Brute Force Algorithm

One row per digit of \( Y \)
- For each row
  - copy the \( k \) bits of \( X \)
- Add the \( k \) rows together
  - \( \Theta(k) \) binary additions of \( \Theta(k) \) bit numbers
- Total runtime is \( \Theta(k^2) \) bit operations
KARATSUBA’S ALGORITHM
- Let’s optimize from four subproblems to three
  Recall: \( X = X_1X_2 + X_2X_1 \)
  Idea: compute \( X_1Y_2 + X_2Y_1 \) with only one multiplication
  Note \( X_1Y_2 + X_2Y_1 \) appears in \( (X_1 + X_2)(Y_1 + Y_2) \)

\[
(X_1 + X_2)(Y_1 + Y_2) = X_1Y_1 + X_1Y_2 + X_2Y_1 + X_2Y_2
\]

Let \( X_1 = X_1 + X_2 \) and \( Y_1 = Y_1 + Y_2 \)
Then \( X_1Y_2 + X_2Y_1 = X_1Y_1 + X_1Y_2 + X_2Y_1 + X_2Y_2 \)
And the other two terms \( X_2Y_2 + X_1Y_2 \) are already in \( XY \)
So \( XY = 2^3X_1Y_1 + 2^2(X_1Y_2 - X_2Y_1 - X_2Y_2) + X_2Y_2 \)

Only three unique multiplications

Running time complexity?
\( T(k) = 3T(\frac{k}{2}) + o(k) \)
Assume \( k = 2^l \) for ease
Master theorem:
- \( a = 3, b = 2, y = 1 \)
- \( x = \log_2 a = \log_2 3 \)
- \( T(k) \in \Theta(k^{\log_2 3}) \)
- \( \approx \Theta(k^{1.585}) \)

Input size increase by \( 10x \) causes runtime to \( 38x \)
Compare to \( \Theta(k^2) \) algo:
\( 10x \) input causes \( 100x \) time

ATTEMPTING A BETTER SOLUTION
What if we first partition the matrix into sub-matrices
Then divide and conquer on the sub-matrices
Example of partitioning: \( 4x4 \) matrix into four \( 2x2 \) matrices

\[
\begin{bmatrix}
  a & b \\
  c & d \\
\end{bmatrix} = \begin{bmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  c_{11} & c_{12} & c_{13} & c_{14} \\
  c_{21} & c_{22} & c_{23} & c_{24} \\
\end{bmatrix}
\]

MULTIPLYING PARTITIONED MATRICES
Let \( A = \begin{bmatrix}
  a & b \\
  c & d \\
\end{bmatrix} \)
Then
\[
\begin{bmatrix}
  a_{11} & a_{12} & a_{13} & a_{14} \\
  a_{21} & a_{22} & a_{23} & a_{24} \\
  c_{11} & c_{12} & c_{13} & c_{14} \\
  c_{21} & c_{22} & c_{23} & c_{24} \\
\end{bmatrix} \times \begin{bmatrix}
  b_{11} & b_{12} & b_{13} & b_{14} \\
  b_{21} & b_{22} & b_{23} & b_{24} \\
  d_{11} & d_{12} & d_{13} & d_{14} \\
  d_{21} & d_{22} & d_{23} & d_{24} \\
\end{bmatrix}
\]

Note \( C = AB = \begin{bmatrix}
  a & b \\
  c & d \\
\end{bmatrix} \begin{bmatrix}
  e & f \\
  g & h \\
\end{bmatrix} \) where \( a, b, ..., k \) are matrices

Note that \( X_2 + Y_2 \) and \( Y_2 + Y_2 \) could be \((k/2 + 1)\)-bit integers.
However, computation of \( X_2 \) can be accomplished by multiplying \((k/2)\)-bit integers and adding to carry bits for extra additions.
Various techniques can be used to handle the case when \( k \) is not a power of two. One possible solution is to pad with zeros on the left. So let \( m \) be the smallest power of two that is \( \geq k \).
The complexity is \( O(m^2 \log_2 m) \).
Since \( m \leq 3k \), the complexity is \( O(3k^2 \log_2 m) \).
Accordingly, the complexity is \( O(3k^2 \log_2 m) \).

There are further improvements known:
- The \( Four-Cost \) algorithm splits \( X \) and \( Y \) into three equal parts and uses five multiplications of \((k/3)\)-bit integers. The recurrence is \( T(k) = 3T(k/3) + O(k) \), and thus \( T(k) \sim O(k^2 \log_2 k) \).
- The 1071 Schönhage-Strassen algorithm (based on FFT) has complexity \( O(n \log \log n \log n) \).
- The 2007 \( Fast \) algorithm has complexity \( O(n \log n \log \log n) \).
IDENTIFYING SUBPROBLEMS TO SOLVE

\[ C = AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} \]

\[ C = AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} \]

Recall \( ae, bg, \) etc., each represent matrix multiplication!

Can compute \( C \) using 8 matrix multiplications!

SIZE OF SUBPROBLEMS & SUBSOLUTIONS

\[ AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} = C = \begin{bmatrix} r & s \\ t & u \end{bmatrix} \]

Suppose \( A, B \) are \( n \times n \) matrices

For simplicity assume \( n \) is a power of 2

Then \( a, b, c, d, e, f, g, h, r, s, t, u \) are \( 2 \times 2 \) matrices

So we compute \( C \) with 8 multiplications of \( 2 \times 2 \) matrices

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

Master theorem

\[ a = 8, b = 2, y = 2 \]

\[ x > y \rightarrow T(n) \in \Theta(n^x) \]

\[ T(n) \in \Theta(n^3) \]

\[ T(n) \in \Theta(n^3) \]

\[ \Theta(1) \]

\[ \Theta(n^2) \]

\[ 8T(n/2) \]

\[ \Theta(n^2) \]

\[ \text{Recall} A, B, \text{have } n^2 \text{ entries} \]

Intuition: to get speedup, must reduce the number of subproblems or their size

STRASSEN FAST MATRIX MULTIPLICATION ALGORITHM

\[ AB = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix} = C = \begin{bmatrix} r & s \\ t & u \end{bmatrix} \]

Define

\[ P_1 = (e - h) \\ P_2 = (a + b) \\ P_3 = (c + d)(e + h) \\ P_4 = (c + d)(g + h) \]

Claim

\[ t = P_3 + P_4 - P_5 - P_6 \]

As an example, according to Strassen, \( t = P_3 + P_4 \)

Plugging in \( P_5, P_6 \) we get \( t = (c + d)(g - e) \)

This simplifies to \( t = ce + de + dg - de = ce + dg \)

 Define each \( P_i \) requires one multiplication

Can combine these \( P_i \) terms with +/- to compute \( r, s, t, u \)
Running time complexity:

- $T(n) = 7T(n^2) + \Theta(n^3)$
  - Master theorem
    - $a = 7, b = 2, y = 2$
    - $x = \log_2 7$
    - $x > y$ so $T(n) \in \Theta(n^x)$
    - $T(n) \in \Theta(n^{\log_2 7}) \approx \Theta(n^{2.81})$

Strassen's algorithm was improved in 1990 by Coppersmith-Winograd. Their algorithm has complexity $O(n^{2.376})$. Some slight improvements have been found more recently:

<table>
<thead>
<tr>
<th>How much better is $\Theta(n^{2.376})$ than $\Theta(n^2)$?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $n=10000$</td>
</tr>
<tr>
<td>$n^{2.376} = 174$ billion</td>
</tr>
<tr>
<td>$n^2 = 1$ trillion</td>
</tr>
<tr>
<td>(6x more)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How much better is $\Theta(n^{2.376})$ than $\Theta(n^3)$?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $n=10000$</td>
</tr>
<tr>
<td>$n^{2.376} = 3.1$ billion</td>
</tr>
<tr>
<td>$n^3 = 1$ trillion</td>
</tr>
<tr>
<td>(312x)</td>
</tr>
</tbody>
</table>