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 with distinct \(x\) values

• Output: all non-dominated points in \(S\), i.e., all points in \(S\) that are not dominated by any point in \(S\).

What’s an easy (brute force) algorithm for this?

PROBLEM DECOMPOSITION

Suppose we pre-sort the points in \(S\) with respect to their \(x\)-coordinates. This takes time \(\Theta(n \log n)\).
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 \( \geq R_k.y \))

Running time complexity? (unit cost model)

Assume \( n = 2^j \) for simplicity

Same as merge sort recurrence \( (\text{RSeq}) \)

\( T(n) = 2T(n/2) + ROO \)

To solve for \( T(n) \), use recursion \( \text{RSeq} \) + \( \text{POSS} \) + \( \text{RSeq} \)
BONUS SLIDE: WHAT IF X VALUES ARE NOT DISTINCT?

- It might contain multiple points with the same x value but with different y values.
- If there are points in Q with the same x as R[1], and a lower y, then the algorithm would say they are dominated by R[1]. Wrong!
- We can find all of the points with the same x as R[1] in linear time.
- If there are multiple such points, and some are in Q, then they are not dominated by R[1], but might be dominated by the next element R[i] of R that has a different x.
- So, we compare them with R[i].y (in linear time) instead of R[1].y.
- All of the other points in Q with x different from R[1].x are compared with R[1].y as usual (in linear time).

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:
  \[ Z = [Z[2k-1], \ldots, Z[0]] \]

BRUTE FORCE ALGORITHM

- One row per digit of Y.
- For each row, copy the k bits of X.
- Add the k rows together.
- \( \Theta(2^k) \) binary additions of \( \Theta(k) \) bit numbers.
- Total runtime is \( \Theta(k^2) \) bit operations.

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

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

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

Recall: \( X = X_L + X^{k/2} X_R + X_R Y_L + X_R Y_R \)

EXPRESSING k-BIT MULT. AS k/2-BIT MULT.

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

Runtime? (bit complexity model)
KARATSUBA’S ALGORITHM

Let’s optimize from four subproblems to three.

Recall: \(X \cdot Y = X \cdot (2^{\frac{n}{2}} \cdot Y + X \cdot Y) + X \cdot Y\)

- Idea: compute \(X_1Y_1, X_1Y_2, X_2Y_1, \text{ and } X_2Y_2\) with only one multiplication
- Note \(X_1 + X_2\) and \(Y_1 + Y_2\) appear in \((X_1 + X_2)(Y_1 + Y_2)\)
- Let \(X' = X_1 + X_2\) and \(Y' = Y_1 + Y_2\)
- Then \(X'Y_1 + X_2Y' = X_1Y_1 - X_1Y_2 - X_2Y_1 + X_2Y_2\)
- And the other two terms \(X_1Y_2\) and \(X_2Y_1\) are already in \(XY\)
- So \(XY = 2X_1Y_1 + 2X_1Y_2 + X_2Y_2\)

For millennia it was widely thought that \(O(n^2)\) multiplication was optimal.

Then in 1960, the 23-year-old Russian mathematician Anatoly Karatsuba took a seminar led by Andrey Kolmogorov, one of the great mathematicians of the 20th century.

Kolmogorov asserted that there was no general procedure for doing multiplication that required fewer than \(n^2\) steps.

Karatsuba thought there was—and after a week of searching, he found it.

Quoting Füredi, author of the \(O(n \log n 2^{O(\log^* n)})\) algorithm:

“It was kind of a general consensus that multiplication is such an important basic operation that, just from an aesthetic point of view, such an important operation requires a nice complexity bound... From general experience the mathematics of basic things at the end always turns out to be elegant.”
And Harvey and van der Hoeven achieved $O(n \log n)$ in November 2020!
(https://hal.archives-ouvertes.fr/hal-02070778/document)

Lower bound of $\Omega(n \log n)$ is conjectured.

A conditional proof is known… it holds if a central conjecture in the area of network coding turns out to be true. [https://arxiv.org/abs/1902.10935]

Unfortunately, simple complexity doesn't always mean simple algorithm…

**MATRIX MULTIPLICATION**

- **Input:** A and B
- **Output:** their product C = AB
- **Naïve algorithm for $n \times n$ matrices:**
  
  - For each output cell $C_{ij}$
  
  
  $C_{ij} = \text{DotProd}(row(A), col(B)^T)$

  $= \sum_{k=1}^{n} A_{ik}B_{kj}$

  - Running time (unit cost)?

- **Size of subproblems & subsolutions**

  $AB = \begin{bmatrix} a & b \end{bmatrix} \begin{bmatrix} e & f \end{bmatrix} = \begin{bmatrix} ae + bg & af + bh \end{bmatrix}$

  $= \begin{bmatrix} e \end{bmatrix} \begin{bmatrix} s \end{bmatrix}$

  - 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 $\frac{n}{2} \times \frac{n}{2}$ matrices
  - So we compute $C$ with 8 multiplications of $\frac{n}{2} \times \frac{n}{2}$ matrices
  - (and 4 additions of such matrices)

**MULTIPLYING PARTITIONED MATRICES**

Let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$

Let $B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$

Note $C = AB = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$

**IDENTIFYING SUBPROBLEMS TO SOLVE**

$C = AB = \begin{bmatrix} ae + bg & af + bh \\ ce + dg & cf + dh \end{bmatrix}$

Recall $ae, bg, etc., each represent matrix multiplication!

Can compute $C$ using 8 matrix multiplications.
**Time complexity (unit cost)?**

\[ T(n) = 8T(\frac{n}{2}) + \Theta(n^2) \]

- **Master theorem**
  - \( a = 8 \), \( b = 2 \), \( y = 2 \)
  - \( x = \log_2 8 = 3 \)
  - \( x > y \) so \( T(n) = \Theta(n^y) \) or \( \Theta(n^2) \)

**Same time as brute force!**

\[ \Theta(1) \text{ or } \Theta(n^2) \]

---

**STRASSEN FAST MATRIX MULTIPLICATION ALGORITHM**

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

**Key idea:** get rid of one multiplication!

**Define**

- \( P_1 = a(f - h) \)
- \( P_2 = (a + b)h \)
- \( P_3 = (a + d)(c + e) \)
- \( P_4 = (b - d)(g + h) \)
- \( P_5 = (a - c)(e + f) \)

Each \( P_i \) requires one multiplication

Can combine these \( P_i \) terms with +/- to compute \( r, s, t, u \)!

---

**Running time complexity?**

\[ T(n) = 7T(\frac{n}{2}) + \Theta(n^2) \]

- **Master theorem**
  - \( a = 7 \), \( b = 2 \), \( y = 2 \)
  - \( x = \log_2 7 \approx 2.81 \)
  - \( x > y \) so \( T(n) = \Theta(n^y) \) or \( \Theta(n^{2.81}) \)
How much better is $Θ(n^{2.81})$ than $Θ(n^3)$?

Let $n=10,000$

$n^{2.81} \approx 174$ billion

$n^3 = 1$ trillion (~6x more)

How much better is $Θ(n^{2.376})$ than $Θ(n^3)$?

Let $n=10,000$

$n^{2.376} \approx 3.2$ billion

$n^3 = 1$ trillion (~312x)