Integer Multiplication

Motivation: multiply two 30-digit integers?

$$153819617987625488624070712657 \times 925421863832406144537293648227$$

- Multiply two 300-digit integers?
- Cannot do this in Java with built-in data types
- 64-bit unsigned integer can only represent integers up to ~20 digits ($2^{64} \approx 10^{20}$)

Warm-Up: Addition

**Input:** two $n$-digit binary integers $x$ and $y$
**Goal:** compute $x + y$

Let's do everything in base-10 instead of binary to make examples more familiar.

Grade-school algorithm:

```
1854
+ 3242
----
5096
```

Running time? $\Theta(n)$

Grade-School Algorithm (Long Multiplication)

**Example:** $n = 3$

```
287
x 132
----
574
861
287
-----
37884
```

Running time? $\Theta(n^2)$
But $xy$ has at most $2n$ digits. Can we do better?

Divide and Conquer

Idea: split $x$ and $y$ in half (assume $n$ is a power of 2)

$$x = \overline{33802367}$
$$y = \overline{45081854}$$

Then use distributive law

$$xy = (10^{n/2}x_1 + x_0) \times (10^{n/2}y_1 + y_0)$$

$$= 10^n x_1 y_1 + 10^{n/2}(x_1y_0 + x_0 y_1) + x_0 y_0$$

Have reduced the problem to multiplications of $n/2$-digit integers and additions of $n$-digit numbers
Divide and Conquer: First Try

Recursive algorithm:

\[ xy = 10^n x_1 y_1 + 10^n/2 (x_1 y_0 + x_0 y_1) + x_0 y_0 \]

Running time? Four multiplications of \( n/2 \) digit numbers plus three additions of at most \( 2n \) digit numbers

\[ T(n) \leq 4T\left(\frac{n}{2}\right) + cn \]
\[ = O(n^{\log_2 4}) \]
\[ = O(n^2) \]

We did not beat the grade-school algorithm. :(

Better Divide and Conquer

Total: three multiplications of \( n/2 \) digit integers, six additions of at most \( 2n \) digit numbers

\[ xy = 10^n x_1 y_1 + 10^n/2 (x_1 y_0 + x_0 y_1) + x_0 y_0 \]

Trick: use three multiplications to compute the following:

\[ A = (x_1 + x_0)(y_1 + y_0) = x_1 y_1 + x_1 y_0 + x_0 y_1 + x_0 y_0 \]
\[ B = x_1 y_1 \]
\[ C = x_0 y_0 \]

Then

\[ xy = 10^n B + 10^n/2 (A - B - C) + C \]

Total: three multiplications of \( n/2 \) digit integers, six additions

Finding Minimum Distance between Points on a Plane

- **Problem**: Given \( n \) distinct points \( p_1, \ldots, p_n \in \mathbb{R}^2 \), find minimum distance between any two points \( d(p_i, p_j) \)

\[ d(p, q) = \sqrt{(p[1] - q[1])^2 + (p[2] - q[2])^2} \]

How long does naive algorithm take? \( O(n^2) \)

We’ll do it in \( O(n \log n) \) steps.

Minimum Distance Algorithm

- Divide points \( P \) with a vertical line into \( P_L \) and \( P_R \) where \( |P_L| = |P_R| = n/2 \)
- Recursively find minimum distance within \( P_L \) and \( P_R \):
  \[ \delta_L = \min_{p \in P_L, q \in P_R} d(p, q) \]
  \[ \delta_R = \min_{p \in P_L, q \in P_R} d(p, q) \]
- Compute \( \delta_M = \min_{p \in P_L, q \in P_R} d(p, q) \) and return \( \min(\delta_L, \delta_R, \delta_M) \)
- If Step 3 takes \( \Omega(n^2) \) time, we get
  \[ T(n) \leq 2T(n/2) + \Omega(n^2) \implies T(n) = \Omega(n^2) \]
- If we can do Step 3 in \( \Theta(n) \) time, we get \( T(n) = O(n \log n) \).

Making Step 3 Efficient

- Need to find \( \min(\delta_L, \delta_R, \delta_M) \) where \( \delta_M = \min_{p \in P_L, q \in P_R} d(p, q) \)
- Suppose that the dividing line is \( x = m \) and \( \delta = \min(\delta_L, \delta_R) \)
Making Step 3 Efficient

- Need to find $\min(\delta_L, \delta_R, \delta_M)$ where $\delta_M = \min_{p \in P_L, q \in P_R} d(p, q)$
- Suppose that the dividing line is $x = m$ and $\delta = \min(\delta_L, \delta_R)$
- Once we know $\delta$, only need $O(n)$ comparisons to find $\min(\delta, \delta_M)$
  - Only compare $(p_1, p_2) \in P_L, (q_1, q_2) \in P_R$ if
    \[ m - \delta < p_1 \leq q_1 < m + \delta \quad \text{and} \quad |p_2 - q_2| < \delta \]
- Each point $p \in P_L$ only gets compared with $O(1)$ points in $P_R$
- Need to identify the relevant comparisons in $O(n)$ time
  - Make two copies of points sorted by each coordinate
  - Ensure both lists are passed to each recursion sorted
  - Given sorted lists, it’s easy to find the relevant points

Merge step pseudocode

- Assume $P_L, P_R$ sorted in increasing by second coordinate.
- Assume they only contain the points within $\delta$ of the boundary.

\[
\text{Lw} = [], \text{Rw} = [], \delta_M = \infty
\]

while $P_L.next(), P_R.next() \neq \text{None}$ do
  if $P_L.next().y < P_R.next().y$ then
    Append $\text{next} = P_L.pop()$ to $\text{Lwindow}$;
    Remove points in $\text{Lw}$, $\text{Rw}$ with $y$-distance $> \delta$ from $\text{next}$
    Compare distances between $\text{Lw}$, $\text{Rw}$, update $\delta_M$.
  else
    Same thing but for $P_R.next()$
  end if
end while

- Fact. $\text{Lw}$, $\text{Rw}$ always of $O(1)$ size!
- Runtime. $O(n \log n)$. 