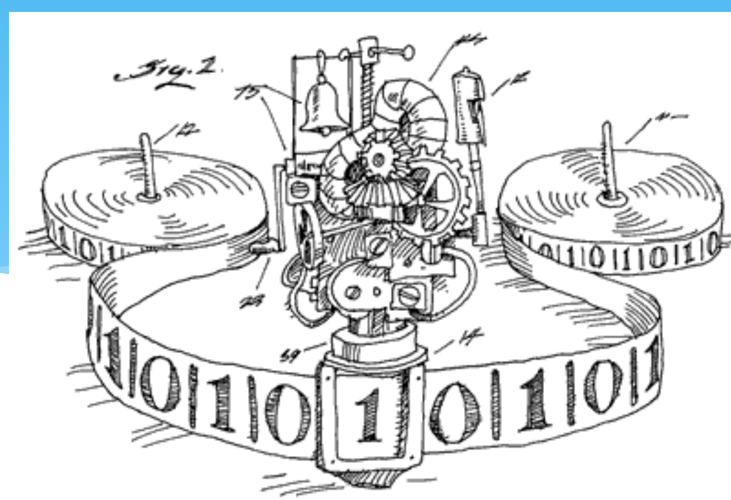
# EECS 376: Foundations of Computer Science

**Chris Peikert**
**11 Jan 2023**



# Divide-and-Conquer Algorithms

**Main Idea:**

1. Divide the input into smaller sub-inputs
2. Solve each sub-input recursively
3. *Combine these solutions in a "meaningful" way*

**Designing, Proving Correctness: an "art"**

* Depends on problem structure, ad-hoc, creative

**Runtime Analysis: "mechanical"**

* Express runtime using a recurrence
* Can often solve using the "Master Theorem"

# Recall: MergeSort

**Algorithm:** *MergeSort*$(A[1..n]$ : array of $n$ integers)

- **if** $n = 1$ **return**
- $m := \lfloor n/2 \rfloor$ — find mid point
- *MergeSort*$(A[1..m])$ — sort first half recursively
- *MergeSort*$(A[m+1..n])$ — sort second half recursively
- **return merge**$(A[1..m], A[m+1..n])$ — combine two **sorted** lists

**Runtime Analysis:**

* $T(n) =$ runtime of MergeSort on input arrays of size $n$.
* Runtime of combining two **sorted** arrays of size $n/2$ is $O(n)$.
* **So: $T(n) = 2T(n/2) + O(n)$. (Same for ClosestPair.)**

**Question:** How do we get a "closed form" for $T(n)$ ?

# Solving Recurrences via Recurrence Trees

* (Follow hand-written notes…)

# The "Master" Theorem

**Template:** Suppose a D&C algorithm breaks size-$n$ input into:

- a *constant* number $k \geq 1$ sub-inputs (solved recursively),
- each of size $n/b$ (for some *constant* $b \geq 1$),
- with cost of $O(n^d)$ to combine the results together.

So its runtime $T(n) = k \cdot T(n/b) + O(n^d)$. Then:

$$T(n) = \begin{cases} O(n^d) & \text{if } k/b^d < 1 \text{ ("root dominates")} \\ O(n^d \log n) & \text{if } k/b^d = 1 \text{ ("levels equal")} \\ O(n^{\log_b k}) & \text{if } k/b^d > 1 \text{ ("leaves dominate")} \end{cases}$$

# Integer Arithmetic

- Many programming languages support "big" integers with a *non-constant* number of digits, and basic operations on them, e.g., $+$, $-$, $*$, /, $\ll$, etc.
  - We can represent a "big" integer as an *array* of digits.
- Q: How does the runtime of arithmetic operations scale with the input size ($n = \#$ digits)?
  - **Addition/Subtraction:** $\mathrm{O}(n)$
  - **Multiplication:** $O(n \log n)$ [Harvey-Hoeven 2019]

# Integer Addition

* Given $n$-digit integers $x$ and $y$
* **Goal:** compute $x + y$ (or $x - y$)
* **Easy:** add digits one at a time and keep a "carry" digit
* **Q:** What's the runtime (in terms of the input size $n$)?
  * $O(n)$

$$
\begin{array}{ccccc}
 & 1 & 1 & 1 & \\
 & & 9 & 4 & 6 \\
+ & & 9 & 8 & 5 \\
\hline
 & 1 & 9 & 3 & 1
\end{array}
$$

# Integer Shift

* Given an $n$-digit integer $x$ and a (small) positive integer $k$
* **Goal:** compute $x \ll k := x \cdot 10^k$ and $x \gg k := \lfloor x \cdot 10^{-k} \rfloor$
* **Easy:** "shift" the array forward or backward by $k$ positions, padding by zeros / dropping digits.
* **Q:** What's the runtime?
  * $\mathrm{O}(n + k)$

# Integer Multiplication

- **Goal:** Given $n$-digit positive integers $x$ and $y$, compute $x \cdot y$.
- **Easy:** use "grade-school" method (works in base 10, 2, etc.)
- **Q:** What's the runtime?
  - $O(n^2)$ (not great!)

| | | | |
|---|---|---|---|
| | | 3 | 4 |
| * | | 3 | 9 |
| | 3 | 0 | 6 |
| 1 | 0 | 2 | |
| 1 | 3 | 2 | 6 |

# Splitting a Number

- $376280 = 376 \cdot 10^3 + 280$
- **Observation 1:** if $X$ is an $n$-digit number, (wlog $n$ is even, by padding)
- $X$ can be split into $n/2$-digit ‘high’ and ‘low’ values :
  - $X = H \cdot 10^{n/2} + L$

| digits | digits |
|---|---|
| H | L |

- **Observation 2:** Splitting works the same way in binary
  - $101010 = 101 \cdot 2^3 + 010$
- An $n$-bit number can be split into $n/2$-bit ‘high’ and ‘low’ values:
  - $X = H \cdot 2^{n/2} + L$

| bits | bits |
|---|---|
| H | L |

# Divide-and-Conquer Multiplication

* **Input:** $X$ and $Y$, two $n$-digit numbers ($n$ is a power of 2)
* Split them into $n/2$-digit 'high' and 'low' values:
  * $X = A \cdot 10^{n/2} + B$
  * $Y = C \cdot 10^{n/2} + D$

| digits | digits |
|---|---|
| A | B |
| C | D |

* Compute $X \cdot Y = (AC) \cdot 10^{n} + (AD + BC) \cdot 10^{n/2} + BD$.
* **Runtime Analysis:**
  * 4 recursive multiplications of $n/2$-digit numbers
  * 2 left shifts (by $n/2$, $n$): $O(n)$ time
  * 3 additions: $O(n)$ time

$$T(n) = \begin{cases} O(n^d) & \text{if } k/b^d < 1 \\ O(n^d \log n) & \text{if } k/b^d = 1 \\ O(n^{\log_b k}) & \text{if } k/b^d > 1 \end{cases}$$

* $T(n) = 4T(n/2) + O(n)$. So $k = 4, b = 2, d = 1 \Longrightarrow k/b^d = 2 > 1$
* **Conclusion:** $T(n) = O(n^{\log_2 4}) = O(n^2)$.

# Divide-and-Conquer Multiplication

- **Conclusion:**
  - Simple, well-known long-multiplication algorithm: $O(n^2)$
  - Complicated, scary divide-and-conquer algorithm: $O(n^2)$

# Karatsuba (1962) Multiplication

- **Input:** $X$ and $Y$, two $n$-digit numbers ($n$ is a power of 2)
- Split them into $n/2$-digit 'high' and 'low' values:
  - $X = A \cdot 10^{n/2} + B$
  - $Y = C \cdot 10^{n/2} + D$
- *Cleverly* compute $X \cdot Y = (AC) \cdot 10^n + (AD + BC) \cdot 10^{n/2} + BD$:
  1. Compute $H = AC$, $L = BD$, and $M = (A + B) \cdot (C + D)$.
  2. Return $H \cdot 10^{n/2} + (M - H - L) \cdot 10^{n/2} + L$.
- **Runtime Analysis:**
  - Just **3** recursive multiplications of $n/2$-digit numbers!
  - Shifts, additions: $O(n)$ time
- Now $T(n) = 3T(n/2) + O(n)$.
- **Conclusion:** $T(n) = O(n^{\log_2 3}) = O(n^{1.585})$. Much better than $O(n^2)$!

# Divide and Conquer Multiplication

* **Conclusions & Remarks:**
  * Karatsuba’s algorithm (1962) was the first known multiplication algorithm that is *asymptotically faster* than “grade school” multiplication.
  * After many improvements, an $O(n \log n)$-time multiplication algorithm was devised by Harvey and van der Hoeven in 2019!
  * We still don’t know if this is the best possible. Perhaps $O(n)$ is attainable, just like for addition/shifts!