Lecture 9

Multiplication algorithms

Chinmay Nirkhe | CSE 421 Winter 2026

Analysis divide and conquer runtimes

The master theorem

- For solving recursive equations of the form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \text{ and } T(<b) = O(1)$$

- Different cases based on how $f(n)$, a, and b compare:

Analysis divide and conquer runtimes

The master theorem

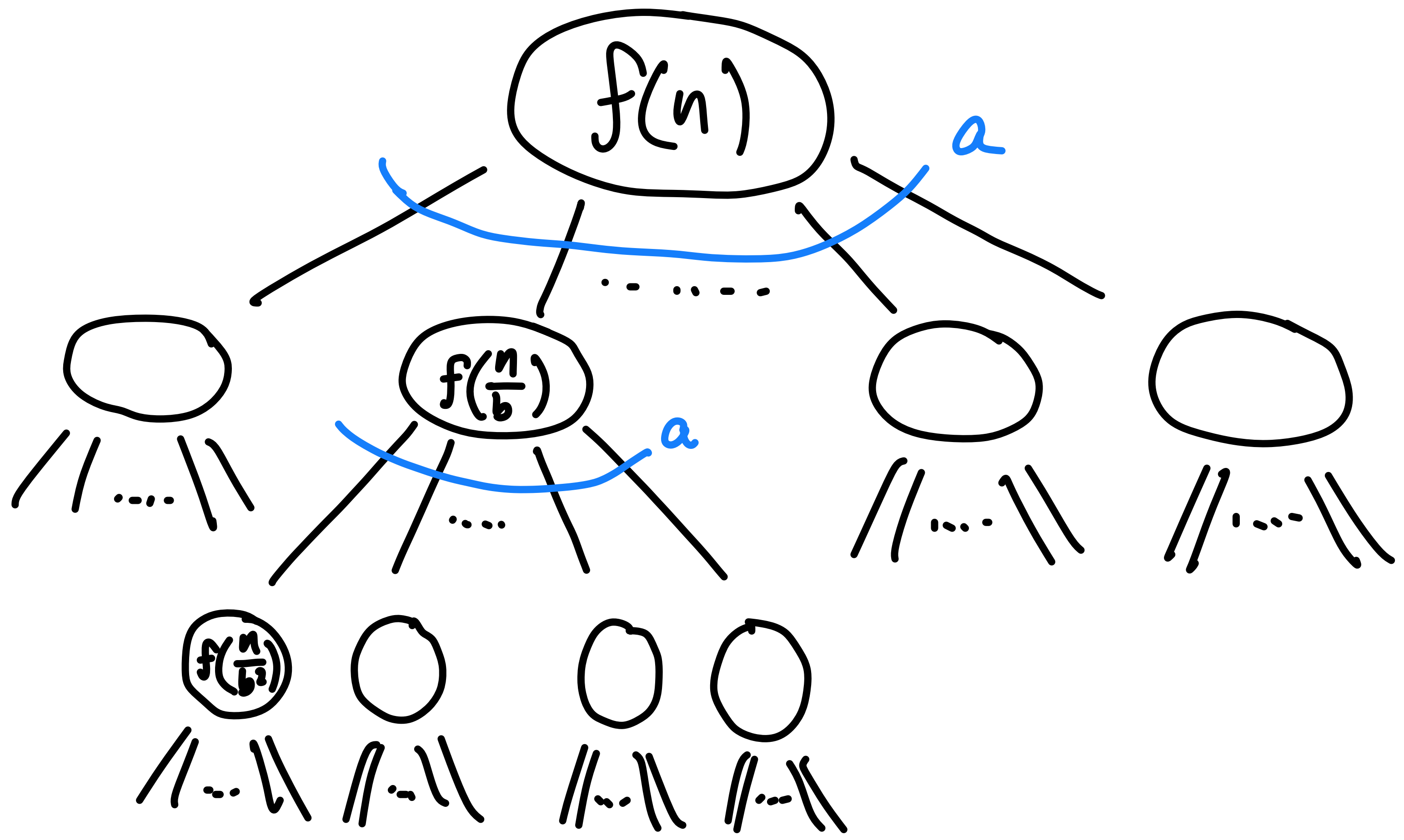
- For solving recursive equations of the form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n^k) \text{ and } T(<b) = O(1)$$

- Different cases based on how $f(n)$, a, and b compare:
  - If $a < b^k$, then $T(n) = O(n^k)$
  - If $a = b^k$, then $T(n) = O(n^k \log n)$
  - If $a > b^k$, then $T(n) = O(n^{\log_b a})$

Proof of the master theorem

- **Proof strategy**:
  - Due to recursion, the problem has a tree like structure



  - Calculate the amount of work done by the “conquer” step at each level
  - Count how many levels of computation there are

Proof the master theorem

- Let $d = \lceil \log_b n \rceil$ so $n \le b^d$

level	# of problems	compute per conquer	total compute at level
d	1	n^k	n^k
$d-1$	a	$(n/b)^k$	$a(n/b)^k = (a/b^k) \cdot n^k$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$d-j$	a^j	$(n/b^j)^k$	$a^j(n/b^j)^k = (a/b^k)^j \cdot n^k$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	a^d	1	a^d

Proof the master theorem

- Let $d = \lceil \log_b n \rceil$ so $n \leq b^d$

$$\text{Total compute} = \sum_{j=0}^{d} \left(\frac{a}{b^k}\right)^j \cdot n^k$$

– If $a < b^k$, then $\sum_{j=0}^{d} \left(\frac{a}{b^k}\right)^j \leq \sum_{j=0}^{\infty} \left(\frac{a}{b^k}\right)^j = \left(1 - \frac{a}{b^k}\right)^{-1} \Rightarrow O(n^k)$.

– If $a = b^k$, then $\sum_{j=0}^{d} \left(\frac{a}{b^k}\right)^j = \sum_{j=0}^{d} 1 = d + 1 \Rightarrow O(n^k \log n)$.

last row of prev table.

– If $a > b^k$, then $\sum_{j=0}^{d} \left(\frac{a}{b^k}\right)^j = \frac{\left(\frac{a}{b^k}\right)^d - 1}{\left(\frac{a}{b^k}\right) - 1} \Rightarrow O\left(\left(\frac{a}{b^k}\right)^d \cdot n^k\right) = O(a^d) = O(n^{\log_b a})$

Analysis divide and conquer runtimes

The master theorem

- For solving recursive equations of the form

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + {\color{blue}O(n^k)} \text{ and } T(<b) = O(1)$$

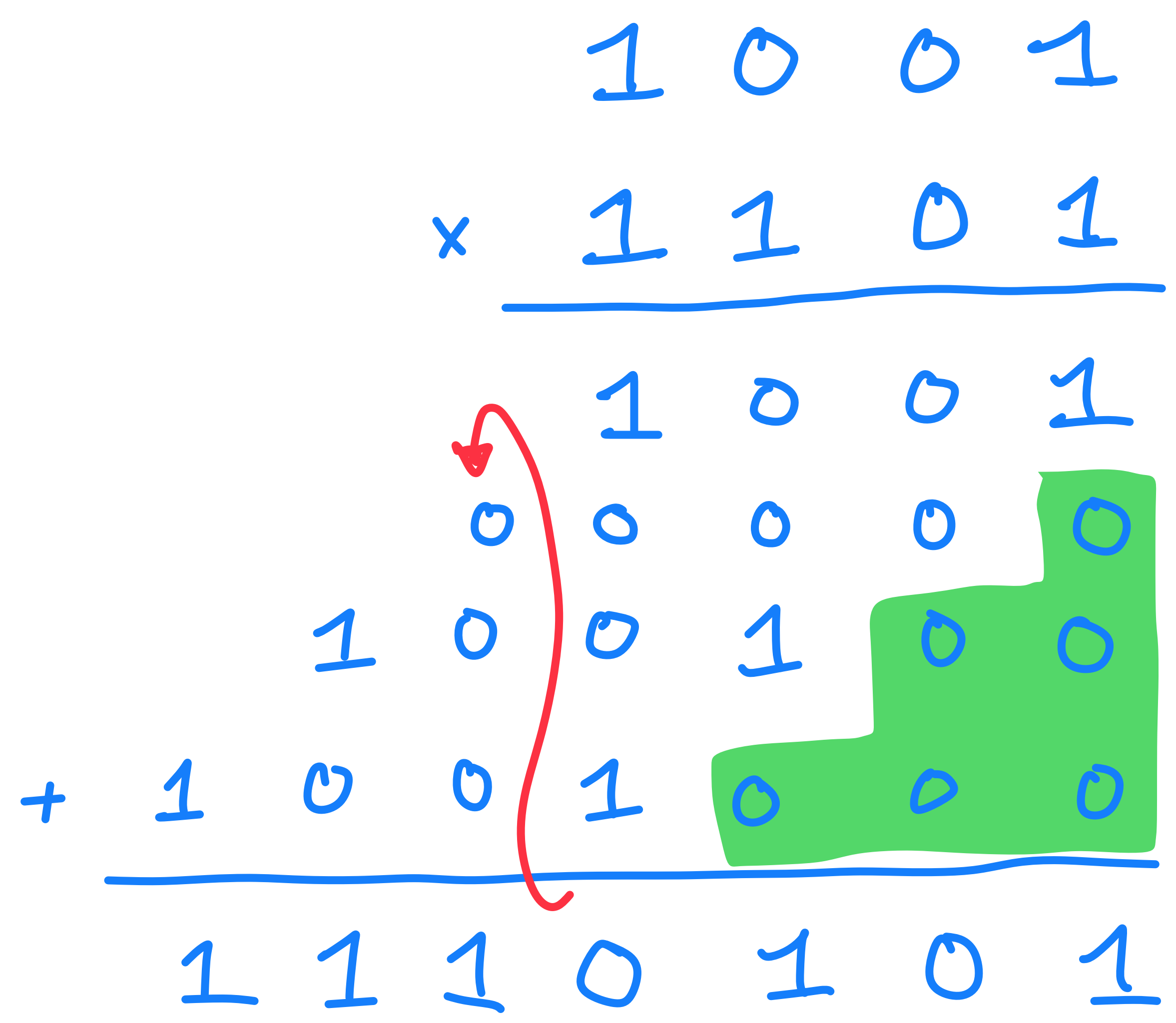
- Different cases based on how $f(n)$, a, and b compare:
  - If $a < b^k$, then $T(n) = O(n^k)$ ← most of the compute is in the largest conquer step
  - If $a = b^k$, then $T(n) = O(n^k \log n)$ ← each level has a commensurate ammount of compute
  - If $a > b^k$, then $T(n) = O(n^{\log_b a})$ ← the number of leaves dominates the computation

Matrix, integer, and (some) polynomial multiplication

Integer multiplication

- **Input**: Two n-bit binary numbers $x, y \in \{0,\ldots,2^n - 1\}$
- **Output**: A $2n$-bit binary number

- Complexity is not measured in RAM model?
- Instead by number of binary operations required.



- Gradeschool multiplication algorithm takes $O(n^2)$ time

The Karatsuba method

$$\boxed{x_1 \quad x_0} \times \boxed{y_1 \quad y_0}$$

$$= \left(2^{n/2} x_1 + x_0\right)\left(2^{n/2} y_1 + y_0\right)$$

$$= 2^{n} x_1 y_1 + 2^{n/2}\left(x_1 y_0 + x_0 y_1\right) + x_0 y_0 .$$

$$= 2^{n}\left(\boxed{x_1} \times \boxed{y_1}\right) + 2^{n/2}\left(\boxed{x_1} \times \boxed{y_0} + \boxed{x_0} \times \boxed{y_1}\right) + \boxed{x_0} \times \boxed{y_0} .$$

left shifts

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n) \implies T(n) = O(n^{\log_2 4}) = O(n^2)$$

no improvements.

The Karatsuba method

$$\boxed{x_1 \mid x_0} \times \boxed{y_1 \mid y_0}$$

$$= \left(2^{n/2} x_1 + x_0\right)\left(2^{n/2} y_1 + y_0\right)$$

$$= 2^n x_1 y_1 + 2^{n/2}\left(x_1 y_0 + x_0 y_1\right) + x_0 y_0 .$$

$$= 2^n x_1 y_1 + 2^{n/2}\Big((x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0\Big) + x_0 y_0 .$$

Identify 3 multiplications of size $\frac{n}{2}$

$$T(n) = 3T\left(\frac{n}{2}\right) + O(n) \implies T(n) = O(n^{\log_2 3}) = O(n^{1.58})$$

Improving integer multiplication

- Fast integer multiplication is used in high-precision arithmetic
- Storing a number to n-bits of precision is equal to 2^{-n} precision
- Karatsuba's algorithm is not the fastest
  - Fastest is $O(n \log n)$ based on the fast Fourier transform (not covered)
  - These are galactic algorithms (not useful in practice)

Matrix multiplication

- **Input**: Two matrices $A, B \in \mathbb{R}^{n \times n}$
- **Output**: The matrix $AB \in \mathbb{R}^{n \times n}$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

where

$$c_{ij} = \sum_k a_{ik} b_{kj} .$$

Trivial algorithm for matrix multiplication

- **Algorithm**:
  - Initialize $n \times n$ array C as zeroes
  - For $i \in [n], j \in [n], k \in [n]$, $\qquad C_{ij} \leftarrow C_{ij} + A_{ik} \cdot B_{kj}$
  - Return C.

- **Runtime**: n^3 multiplications + n^3 additions
- Can we improve this with divide and conquer?

Matrix multiplication naturally decomposes

- **Matrix multiplication of matrices**

terms do not commute

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

- **Divide and conquer:**
  - Decompose into 8 matrix multiplications of $n/2 \times n/2$ matrices and 4 matrix additions of $n/2 \times n/2$ matrices
  - $T(n) = 8T\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2 \implies T(n) = O(n^{\log_2 8}) = O(n^3)$

$a = 8$
$b = 2$
$k = 2$

$a > b^k$

leaf-heavy computation

Strassen's divide and conquer (1968)

- Can we decrease the number of mini-multiplications at the cost of increasing the number of mini-additions?
- If we were to somehow decrease to 7 multiplications but 18 additions …
- $T(n) = 7T\left(\frac{n}{2}\right) + \frac{18}{4}n^2 \implies T(n) = \frac{18}{4} \cdot O(n^{\log_2 7}) = O(n^{2.8074})$

$a = 7$ $\quad a > b^k$ but

$b = 2$ $\quad \log_b a$ is smaller…

$k = 2$

- But how do we achieve this decrease?
- **Find repeated terms**.

A clever decomposition

We know that if

$$\boxed{A} \cdot \boxed{B} = \boxed{C}$$

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \cdot \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

then $C_{11} = A_{11} B_{11} + A_{12} B_{21}$.

Pictorally, let's represent this fact by

$$C_{11} = \begin{array}{c|cccc} & A_{11} & A_{12} & A_{21} & A_{22} \\ \hline B_{11} & 1 & & & \\ B_{21} & & 1 & & \\ B_{12} & & & & \\ B_{22} & & & & \end{array}$$

A clever decomposition

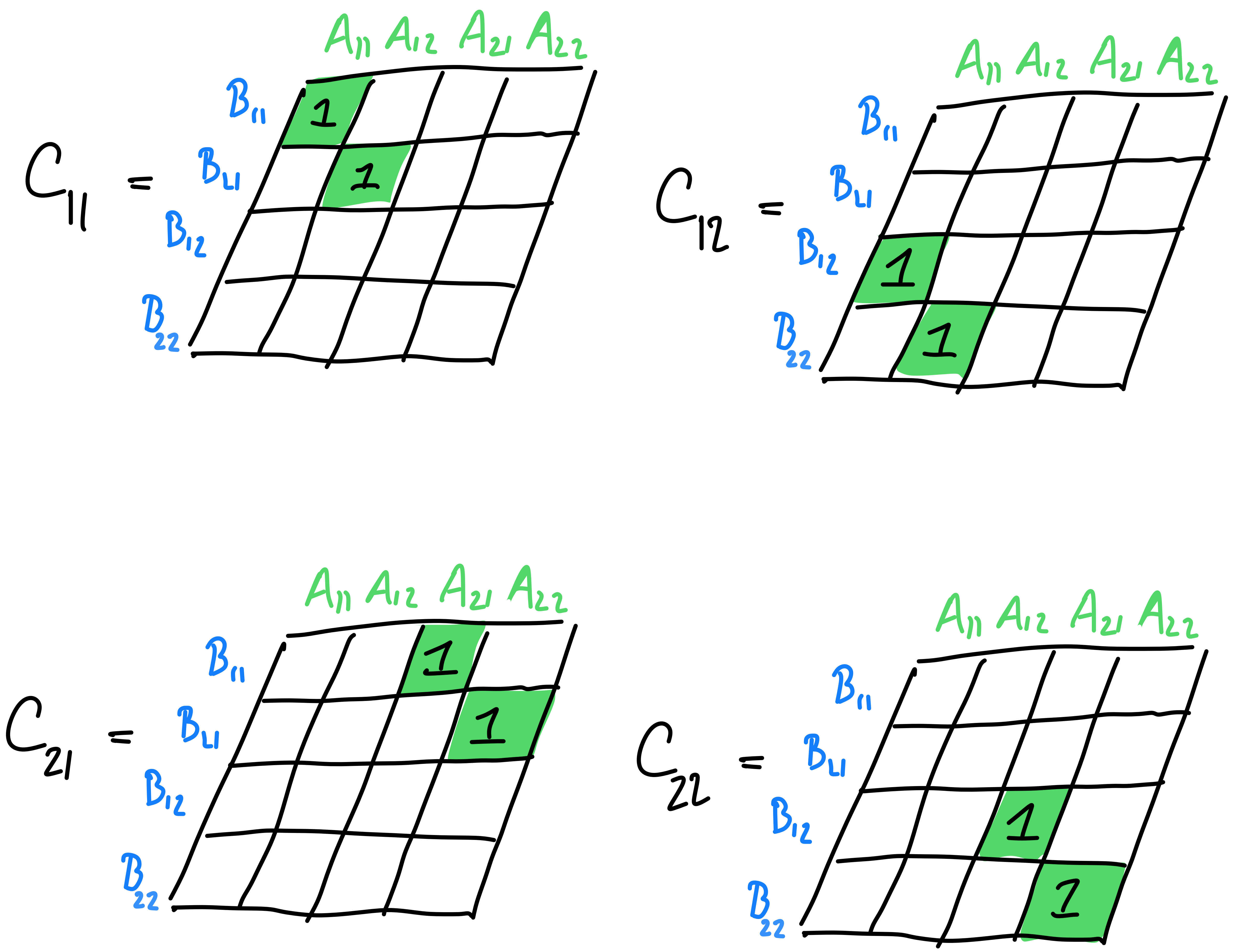
Similarly,

$$C_{11} = A_{11}B_{11} + A_{12}B_{21}$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22}$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21}$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22}$$



A clever decomposition

Now, what happens if we want to calculate

$$M = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$= A_{11}B_{11} + A_{11}B_{22} + A_{22}B_{11} + A_{22}B_{22}$$

$$M = \begin{array}{c} \\ B_{11} \\ B_{21} \\ B_{12} \\ B_{22} \end{array} \begin{array}{c} \begin{array}{cccc} A_{11} & A_{12} & A_{21} & A_{22} \end{array} \\ \left[\begin{array}{cccc} 1 & & & 1 \\ & & & \\ & & & \\ 1 & & & 1 \end{array}\right] \end{array}$$

A clever decomposition

Another example...

$$M' = A_{11}(B_{12} - B_{22})$$

$$= A_{11}B_{12} - A_{11}B_{12}$$

$$M' = \begin{array}{c|cccc} & A_{11} & A_{12} & A_{21} & A_{22} \\ \hline B_{11} & & & & \\ B_{21} & & & & \\ B_{12} & 1 & & & \\ B_{22} & -1 & & & \end{array}$$

A clever decomposition

We can add these diagrams...

$$M = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M' = A_{11}(B_{12} - B_{22})$$

$$M = \begin{array}{c|cccc} & A_{11} & A_{12} & A_{21} & A_{22} \\ \hline B_{11} & 1 & & & 1 \\ B_{21} & & & & \\ B_{12} & & & & \\ B_{22} & 1 & & & 1 \end{array}$$

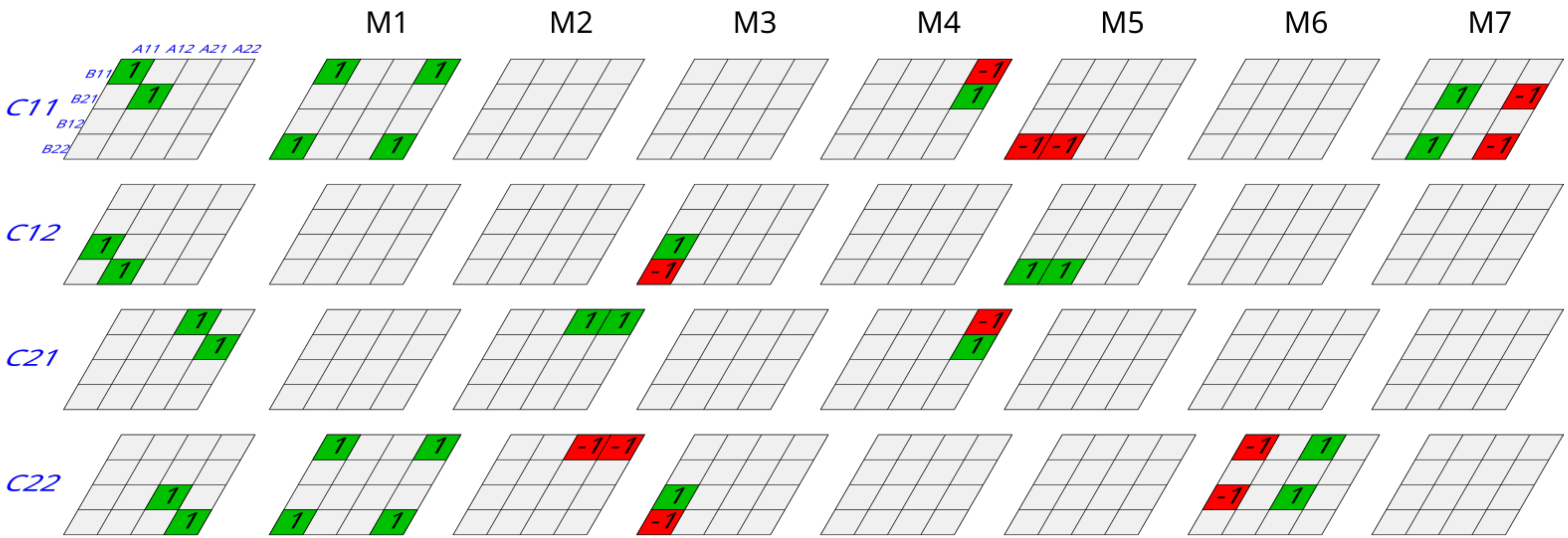
$$M' = \begin{array}{c|cccc} & A_{11} & A_{12} & A_{21} & A_{22} \\ \hline B_{11} & & & & \\ B_{21} & & & & \\ B_{12} & 1 & & & \\ B_{22} & -1 & & & \end{array}$$

$$M + M' = \begin{array}{c|cccc} & A_{11} & A_{12} & A_{21} & A_{22} \\ \hline B_{11} & 1 & & & 1 \\ B_{21} & & & & \\ B_{12} & 1 & & & \\ B_{22} & & & & 1 \end{array}$$

A clever decomposition

1 mult + 2 additions | 1 mult + 1 addition | 1 mult + 1 addition | 1 mult + 1 addition | 1 mult + 1 addition | 1 mult + 2 addition | 1 mult + 2 additions

7 multiplications + 10 additions



$C_{11} = M_1 + M_4 - M_5 + M_7$

$C_{12} = M_3 + M_5$

$C_{21} = M_2 - M_4$

$C_{22} = M_1 - M_2 + M_3 + M_6$

8 additions

Wikipedia article for Strassen's algorithm

$M_1 = (A_{11} + A_{22})(B_{11} + B_{12})$

$M_5 = (A_{11} + A_{12})B_{22}$

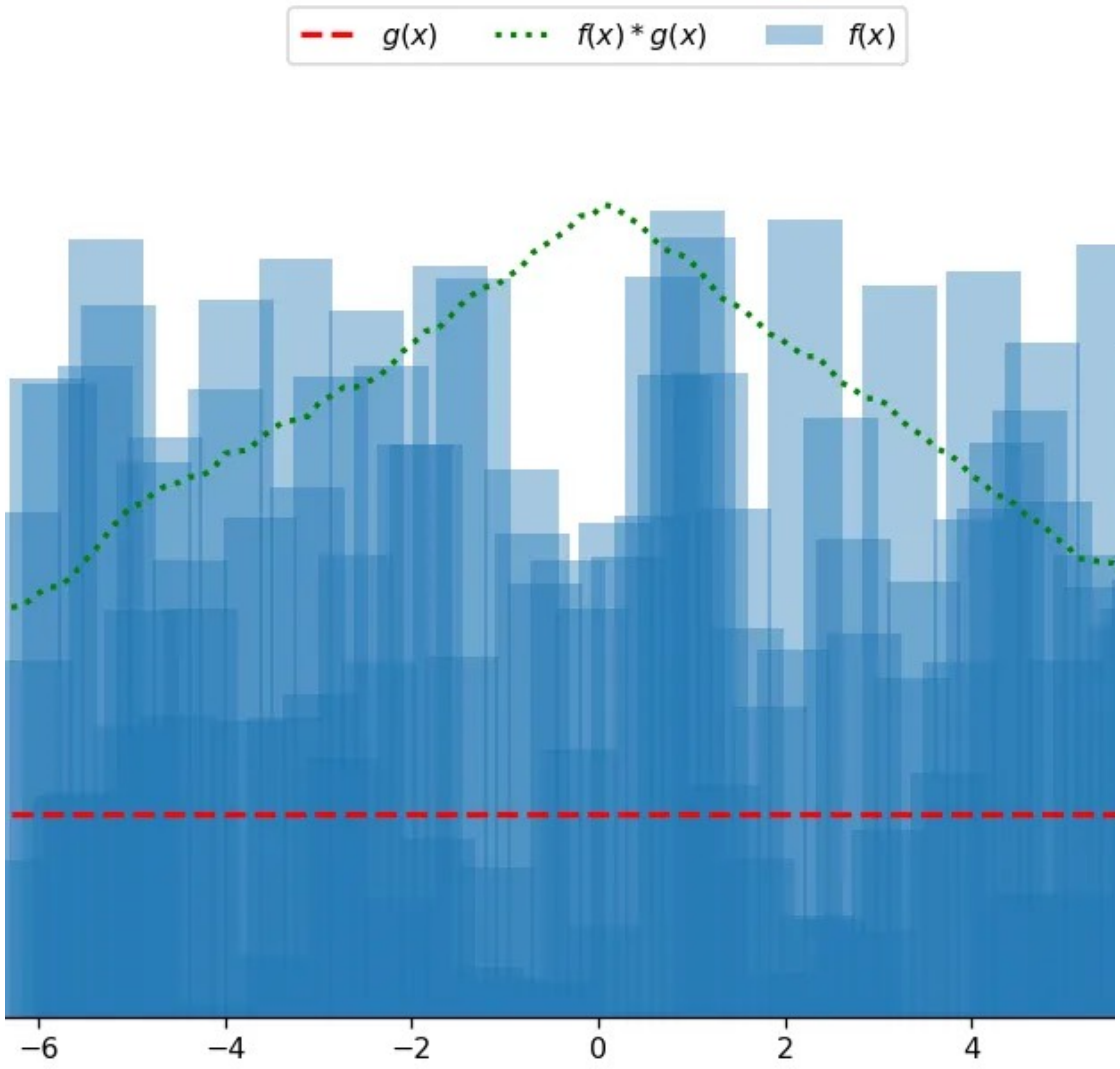
$M_4 = A_{22}(B_{21} - B_{11})$

Strassen's algorithm details

- Best for matrices of size $2^m \times 2^m$. Pad the matrix with zeroes until it is.
- Strassen's has 18 mini-additions. Only beneficial if $n \geq 32$.
  - For smaller matrices, use $O(n^3)$ algorithm.
  - Still a base case for the recursive definition. Only adjust $O(\cdot)$ constants.
- Is there an even cleverer decomposition into fewer mini-multiplications?
  - Not for dividing into $n/2 \times n/2$ mini-matrices
  - Other divisions plus clever tricks have gotten algorithms down to $O(n^{2.371339})$ [May 2024]
  - **Major open question**: $O(n^{2+\epsilon})$ time algorithm possible for all $\epsilon > 0$.

Convolution

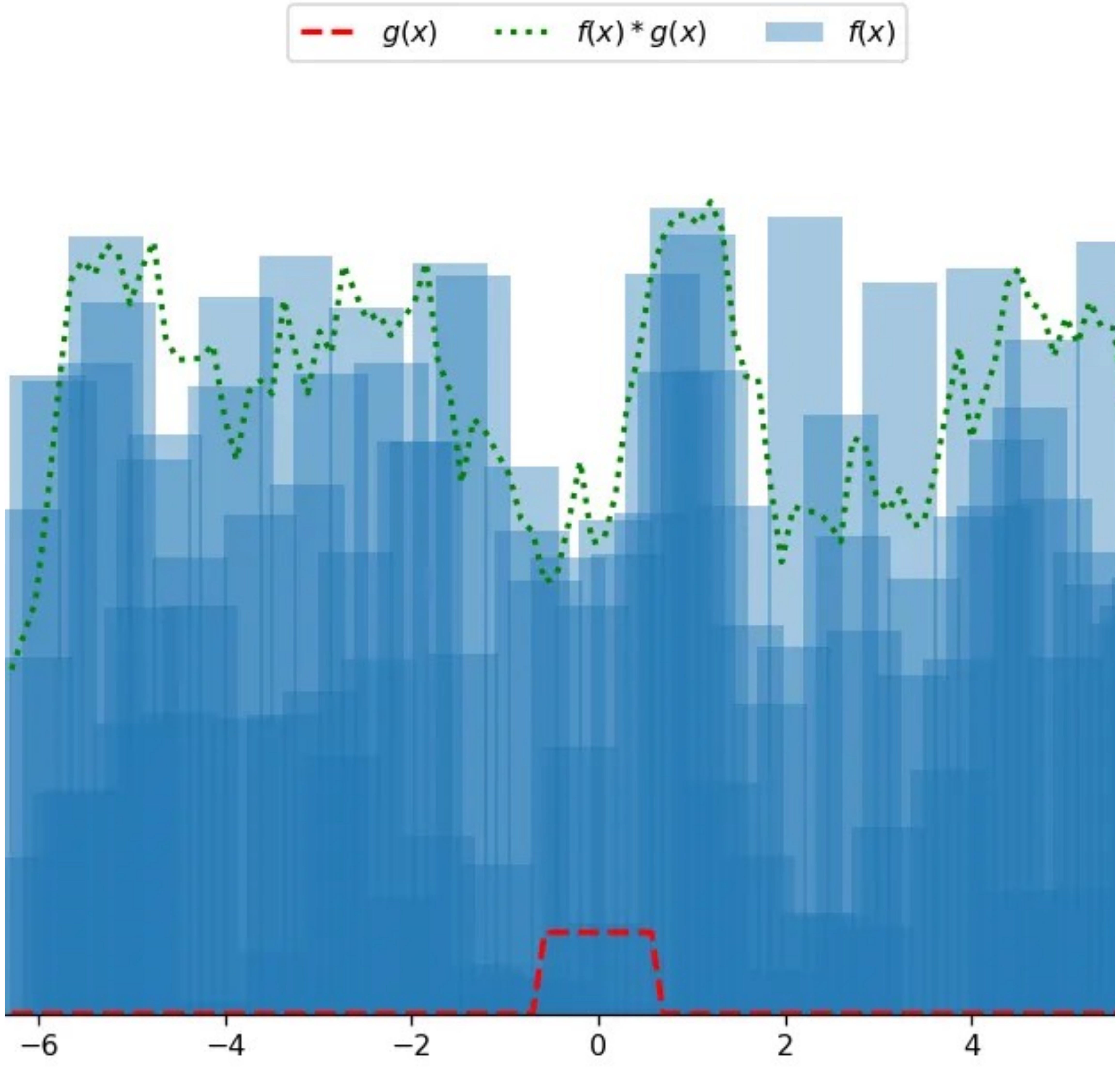
- An algorithm for combining two signals to form a third signal
- Shows up most commonly now in *convolution neural networks*
- $(f * g)_k := \sum_{j=0}^{n} f_j \cdot g_{k-j}$ vs
- $(f * g)(x) = \int_{-\infty}^{\infty} f(\tau) g(x - \tau) d\tau$
    - This is the area under the curve f with weights defined by g
    - Let's you smooth out the curve f by picking g



Source: Medium post by TDS archive.

Convolution

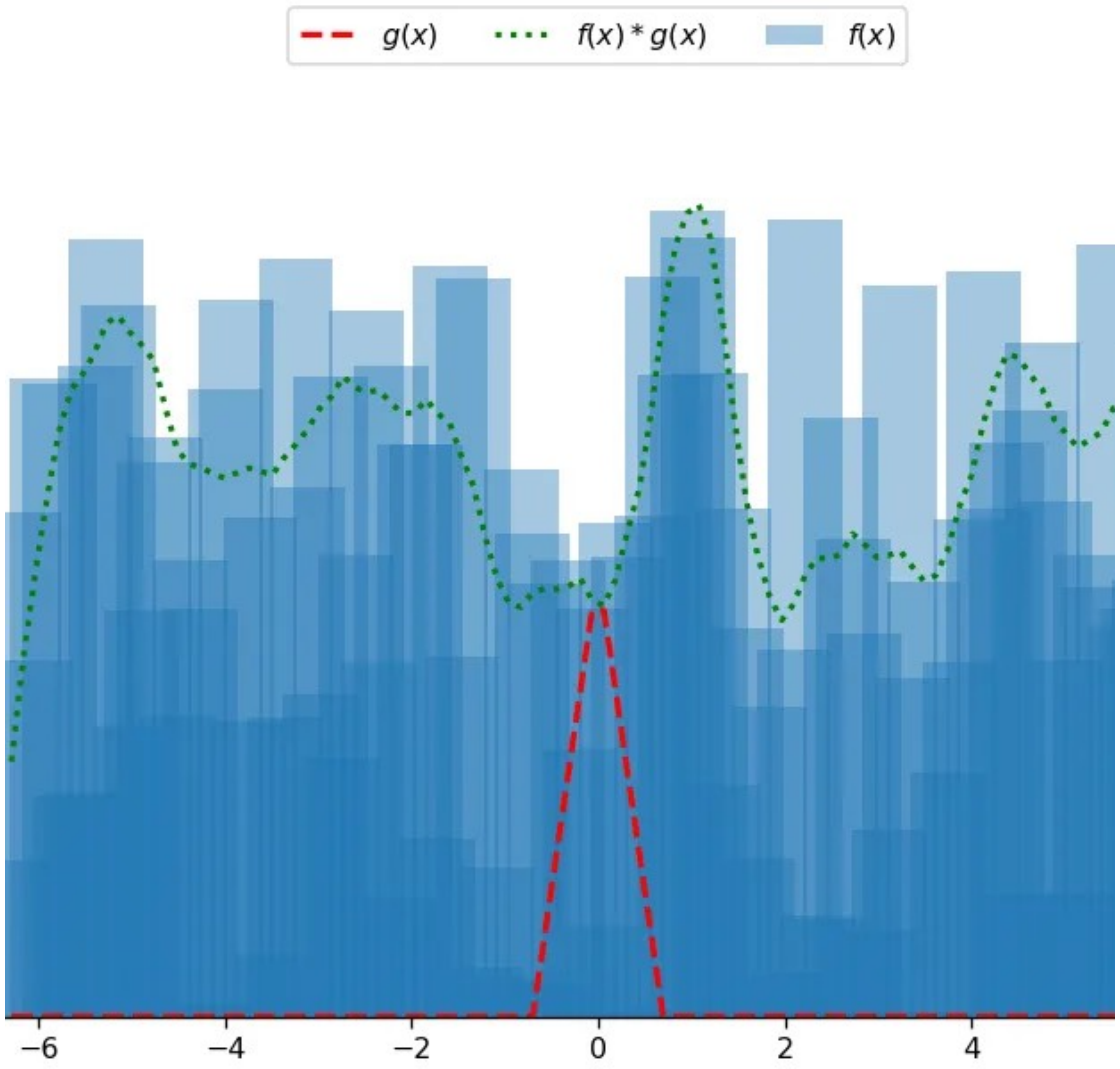
- An algorithm for combining two signals to form a third signal
- Shows up most commonly now in *convolution neural networks*
- $(f * g)_k := \sum_{j=0}^{n} f_j \cdot g_{k-j}$ vs
- $(f * g)(x) = \int_{-\infty}^{\infty} f(\tau) g(x - \tau) d\tau$
    - This is the area under the curve f with weights defined by g
    - Let's you smooth out the curve f by picking g



Source: Medium post by TDS archive.

Convolution

- An algorithm for combining two signals to form a third signal
- Shows up most commonly now in *convolution neural networks*
- $(f * g)_k := \sum_{j=0}^{n} f_j \cdot g_{k-j}$ vs
- $(f * g)(x) = \int_{-\infty}^{\infty} f(\tau)g(x-\tau)d\tau$
  - This is the area under the curve f with weights defined by g
  - Let's you smooth out the curve f by picking g



Source: Medium post by TDS archive.

Convolution

Gaussian blurring and edge detection

- Ex. We can also apply a 2D version of convolution for image processing

Source: Stanford 315b lectures

Convolution

- Filtering signals (low-pass, high-pass)
  - Convolve with a signal to filter out certain frequencies
- Audio effects (reverb, echo, suppression)
- Image processing
- And more!

Median

- **Input**: Input list $\vec{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ for n odd.
- **Output**: The median element i.e. $y_{(n+1)/2}$ when $\vec{y} = \mathrm{sort}(\vec{x})$.

- An upper bound for the runtime is $O(n \log n)$ from sorting + selecting.
- Can we do better? Could we achieve $O(n)$?

Median

- Consider a divide and conquer algorithm for median
- What would the recurrence relation have to be for $T(n) = O(n)$?
- **Case 1**: $T(n) = 2T(n/2) + O(1)$
  - Challenge is to split the problem X into two halves with $O(1)$ compute
  - And to “stitch” the solutions to the two subproblems together in $O(1)$ compute
- **Case 2**: $T(n) = T(n/2) + O(n)$
  - With $O(n)$ time, we can make a constant number of passes through the list X
  - After constant number of passes, we need to find a sublist X' of size $n/2$ which must contain the median
  - Then we recurse on the sublist X'

Selection

- Let's define a more general problem called "Selection"
  - **Input:** pair $(\vec{x}, k) \in \mathbb{R}^n \times [n]$.
  - **Output:** The k-th element y_k when $\vec{y} = \mathrm{sort}(\vec{x})$.
- Generalizes the median problem

Selection

Find the 6th element

0	1	7	2	1	3	4	4	8	2	1	0	3	3	6	5	5	2	2	3

Selection

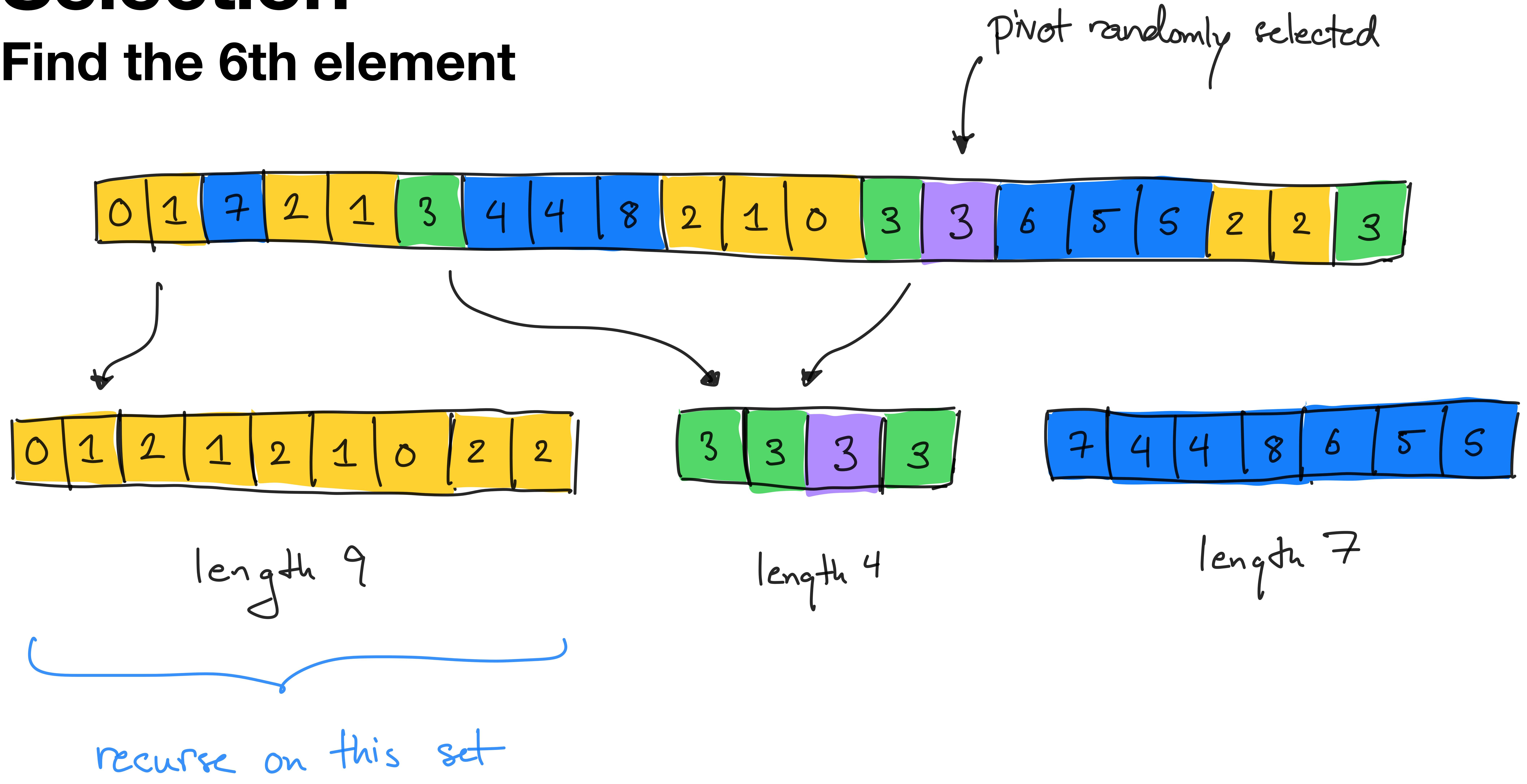
Find the 6th element

pivot randomly selected

0	1	7	2	1	3	4	4	8	2	1	0	3	3	6	5	5	2	2	3

Selection

Find the 6th element



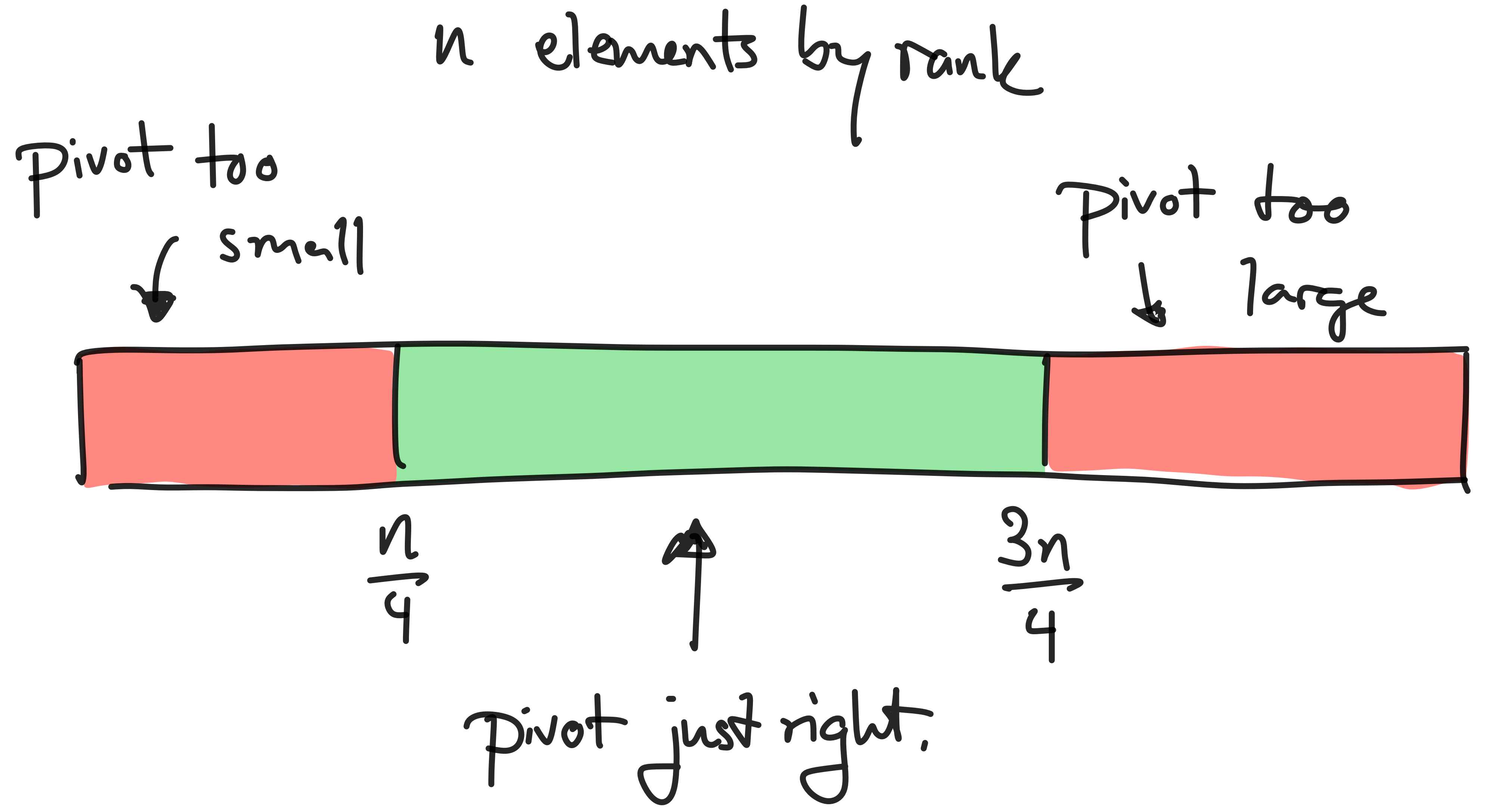
Selection

- **Recursive algorithm** $\text{Selection}(X, k)$**:**
  - Randomly sample j from $[n]$. Call x_j the "**pivot**".
  - Filter X into X_L, X_E, and X_R based on if $x_i < x_j$, $x_i = x_j$, or $x_i > x_j$.
  - If $|X_L| \geq k$, recursively output $\text{Selection}(X_L, k)$.
  - Else if, $|X_L| + |X_E| \geq k$, output x_j.
  - Else, recursively output $\text{Selection}(X_R, k - |X_L| - |X_E|)$.

Runtime analysis

- In order to apply the master theorem, we would need to argue that each recursive call was reducing the input size from n to n/b for $b > 1$
  - $T(n) = T(n/b) + cn \implies T(n) = \dfrac{c}{1 - 1/b}n$
- However, each call may not reduce the size from n to n/b
- Depends on how close the randomly chosen x_j is to the middle
  - If pivot x_j was the largest element, then $|X_L| = n - 1, |X_E| = 1$, and $|X_R| = 0$.
  - Decreases instance size from n to $n - 1$.
  - Fortunately, the probability this occurs is $1/n$.

Runtime analysis



- **Amortized analysis**:
  - If pivot x_j is the ℓ-th element, then the next problem is of size $\leq \max\{\ell, n-\ell\}$.
  - With probability $\geq 1/2$, pivot x_j is the ℓ-th element for $\ell \in \{n/4,\ldots,3n/4\}$.
  - The expected compute in reducing from n-sized instance to a $3n/4$-sized instance is $O(n)$.
- Total **expected** runtime: $T(n) = T(3n/4) + O(n) \implies T(n) = O(n)$.

Runtime analysis

- **Amortized analysis**:
  - If pivot x_j is the ℓ-th element, then the next problem is of size $\leq \max\{\ell, n-\ell\}$.
  - With probability $\geq 1/2$, pivot x_j is the ℓ-th element for $\ell \in \{n/4, \ldots, 3n/4\}$.
  - The expected compute in reducing from n-sized instance to a $3n/4$-sized instance is $O(n)$.
    - $\geq 1/2$ probability, shrinks in 1 reduction.
    - $\geq 1/4$ probability, shrinks in 2 reductions.
    - … $\geq 1/2^j$ probability, shrinks in j reductions …
    - Expected compute is $\leq O(n) \cdot (\frac{1}{2} + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \ldots) = O(n) \cdot 2$
- Total **expected** runtime: $T(n) = T(3n/4) + O(n) \implies T(n) = O(n)$.