Lecture 18: Algorithm Complexity and Matrix Operations

Announcements and reminders

- Homework 7 (Moodle) is posted and is due **Friday at 12 PM Noon**
- The CU final exam schedule is up. You must take your final exam during your scheduled final exam time.
  Tony's section: 7:30 - 10 PM, Sunday 16 Dec
  Rachel's section: 1:30 - 4 PM, Wednesday 19 Dec

What did we do last time?

- We learned about estimating the **complexity of algorithms**
- We learned about estimating the **growth rates of functions**

    … because that's useful to see **how (in)efficient an algorithm is**, depending on the size of the input

Today:

- A bit more about ***algorithm complexity***, particularly in the context of ***matrix operations***

Quick recap

Growth of functions:

Definition: Let f and g be functions from the set of integers. We say that $f(n)$ is $\mathcal{O}(g(n))$ if there are constants C and k such that

$$| f(n) | \leq C \, | g(n) |$$

whenever $n > k$. [[*"f(n) is big-oh of g(n)"*]]

Definition: S'pose that $f(n)$ and $h(n)$ are functions from the set of integers. We say that $f(n)$ is $\Omega(h(n))$ if there are constants C and k such that

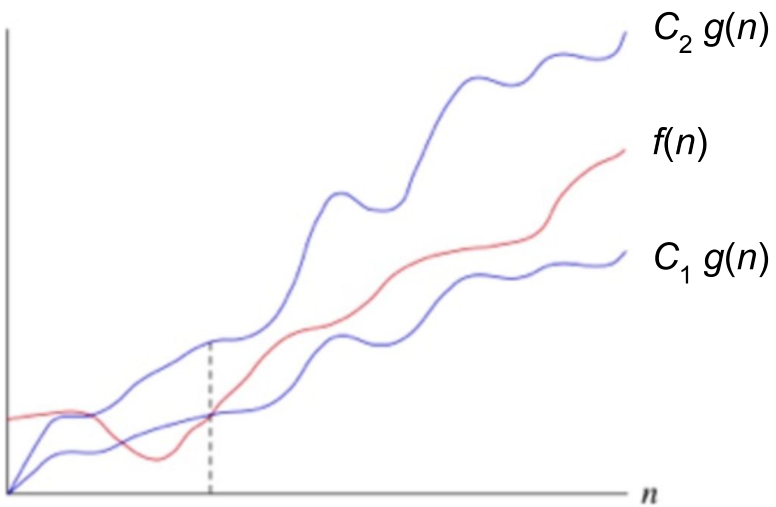
$$| f(n) | \geq C \, | h(n) |$$

whenever $n > k$. [[*"f(n) is big-Omega of h(n)"*]]

Quick recap

Growth of functions:

Definition: If $f(n)$ is both $\mathcal{O}(g(n))$ and $\Omega(g(n))$ then we say $f(n)$ is $\Theta(g(n))$, and also say that $f(n)$ is of **<u>order</u>** $g(n)$. [[*"f(n) is big-Theta of g(n)"*]]



Algorithm complexity

Example: Show that $h(n) = 2n^2 + 5n \log n$ is $\Theta(n^2)$

Solution: Two steps: first, show $h(n)$ is $\mathcal{O}(n^2)$; then, show $h(n)$ is $\Omega(n^2)$

Showing $h(n)$ is $\mathcal{O}(n^2)$ …

Algorithm complexity

Example: Show that $h(n) = 2n^2 + 5n \log n$ is $\Theta(n^2)$

Solution: Two steps: first, show $h(n)$ is $\mathcal{O}(n^2)$; then, show $h(n)$ is $\Omega(n^2)$

Showing $h(n)$ is $\mathcal{O}(n^2)$ …

Note that for $n > 1$, $\log n \leq n$

$\Rightarrow n \log n \leq n^2$

$\Rightarrow h(n) = 2n^2 + 5n \log n \leq 2n^2 + 5n^2 = 7n^2$

$\Rightarrow$ So we have $h(n)$ is $\mathcal{O}(n^2)$ (with $C = 7$ and $k = 1$) ✔

Algorithm complexity

Example: Show that $h(n) = 2n^2 + 5n \log n$ is $\Theta(n^2)$

Solution: Two steps: first, show $h(n)$ is $\mathcal{O}(n^2)$; then, show $h(n)$ is $\Omega(n^2)$

Showing $h(n)$ is $\mathcal{O}(n^2)$ …

Note that for $n > 1$, $\log n \leq n$

$\Rightarrow n \log n \leq n^2$

$\Rightarrow h(n) = 2n^2 + 5n \log n \leq 2n^2 + 5n^2 = 7n^2$

$\Rightarrow$ So we have $h(n)$ is $\mathcal{O}(n^2)$ (with $C = 7$ and $k = 1$) ✔

Showing $h(n)$ is $\Omega(n^2)$ …

Easier: $h(n) = 2n^2 + 5n \log n \geq 2n^2$ for $n > 1$

$\Rightarrow$ So we have $h(n)$ is $\Omega(n^2)$ (with $C = 2$ and $k = 1$) ✔

Since h(n) is both O(n2) and Ω(n2), it is Θ(n2) □

Matrices and matrix operations

- **Linear algebra** is the workhorse of computational science.
- In scientific/engineering computing, a huge amount of computing time is spent on matrix operations.
- Applications of matrices are broad, but they were invented for **a simple purpose**: *to make solving systems of equations cleaner.*

$$
\begin{aligned}
3x + 4y + 5z &= 1 \\
2x + 8y + 3z &= 2 \\
4x + 2y + 2z &= 3
\end{aligned}
\quad \Leftrightarrow \quad
\underbrace{\begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix}}_{\textbf{Matrix}}
\begin{bmatrix} x \\ y \\ z \end{bmatrix}
=
\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}
$$

Vectors

Matrices and matrix operations

$$\begin{aligned} 3x + 4y + 5z &= 1 \\ 2x + 8y + 3z &= 2 \\ 4x + 2y + 2z &= 3 \end{aligned} \Leftrightarrow \begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

- Rectangular thing is a **matrix**, tall skinny things are **vectors**
- **Definition:** A matrix with *m* rows and *n* columns has **dimensions** $m \times n$
- **Definition:** A vector with *n* entries has **length** *n*
- **Notation:** Matrices are represented by capital letters, like *A* and *M*.
  Vectors are represented by lowercase letters like **x** and **b**
  (often bold-faced)
- **Example:** The above **matrix equation** could be written as *A***x** = **b**

Matrices and matrix operations

- Matrices and vectors can be **added** and **multiplied** (but not divided)
- **Definition:** The **sum** of matrices A and B is the matrix obtained by adding the corresponding entries of each matrix together
- **Example:**

$$\begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 4 & 6 & 8 \\ 6 & 13 & 9 \\ 11 & 10 & 11 \end{bmatrix}$$

- **Note:** Only makes sense if A and B have the **same** dimensions!
- **Notation:** We refer to the entry in the i^{th} row and j^{th} column of the matrix A as a_{ij}, or $A[i, j]$.

Matrices and matrix operations

Complexity of matrix addition:

- Straightforward: We add each pair of entries.

  ⇒ for two $m \times n$ matrices, there are mn entries, so mn additions

  ⇒ for square matrices of size $n \times n$, that's n^2 additions, so this is $\mathcal{O}(n^2)$

Matrices and matrix operations

Complexity of matrix addition:

- Using the slick psuedocode method (for adding two square matrices):

```
def matrixAdd (A, B):
    S = "0"                          # initialize as n × n zero matrix
    for i in 1,num_rows:
        for j in 1,num_cols:
            S[i,j] = A[i,j] + B[i,j]
    return (S)
```

Turn loops into sums and count up the basic operations:

Complexity: $$\sum_{i=1}^{n}\sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n^2$$

Matrices and matrix operations

- Matrices can also **multiply** vectors, resulting in a new vector.

$$\begin{aligned} 3x + 4y + 5z &= 1 \\ 2x + 8y + 3z &= 2 \\ 4x + 2y + 2z &= 3 \end{aligned} \Leftrightarrow \begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

- This operation is defined directly out of the analogy between matrix equations and linear systems of equations.

$$\begin{aligned} &3x + 4y + 5z \\ &2x + 8y + 3z \\ &4x + 2y + 2z \end{aligned} \Leftrightarrow \begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Matrices and matrix operations

- Think of it as taking the vector and setting it down on top of the matrix.

$$\begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \cdot 1 + 4 \cdot 2 + 5 \cdot 3 \\ 2 \cdot 1 + 8 \cdot 2 + 3 \cdot 3 \\ 4 \cdot 1 + 2 \cdot 2 + 2 \cdot 3 \end{bmatrix} = \begin{bmatrix} 26 \\ 27 \\ 14 \end{bmatrix}$$

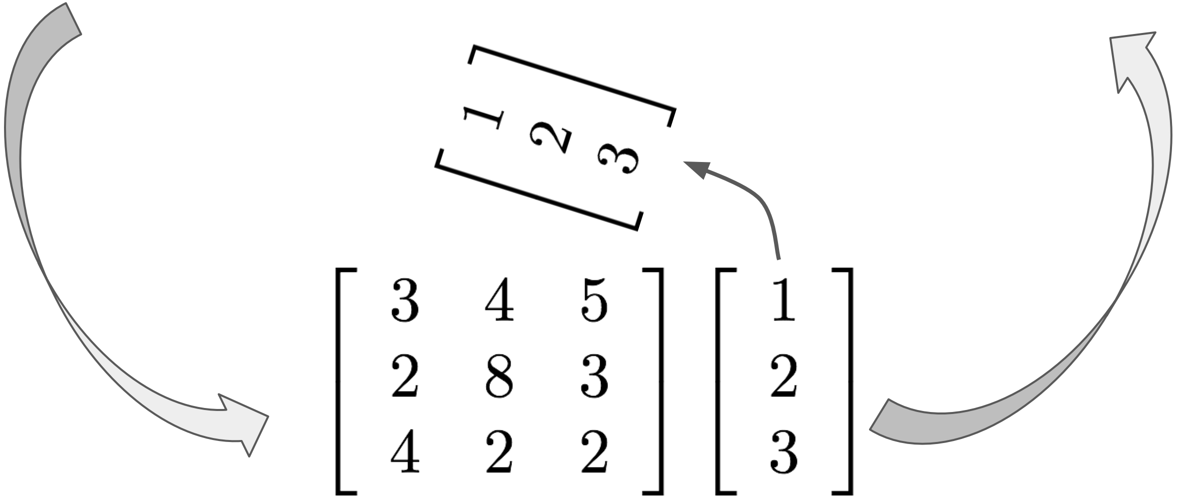
$$\begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Important rule: This means that the length of the vector **must equal** the number of columns of the matrix.

Matrices and matrix operations

- Think of it as taking the vector and setting it down on top of the matrix.

$$\begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \cdot 1 + 4 \cdot 2 + 5 \cdot 3 \\ 2 \cdot 1 + 8 \cdot 2 + 3 \cdot 3 \\ 4 \cdot 1 + 2 \cdot 2 + 2 \cdot 3 \end{bmatrix} = \begin{bmatrix} 26 \\ 27 \\ 14 \end{bmatrix}$$

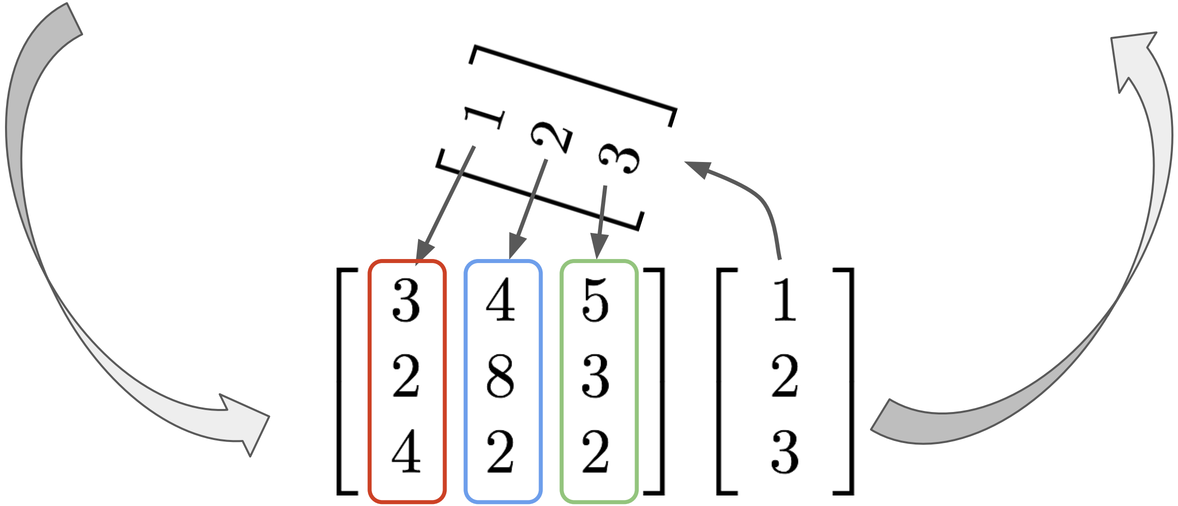


Important rule: This means that the length of the vector **must equal** the number of columns of the matrix.

Matrices and matrix operations

- Think of it as taking the vector and setting it down on top of the matrix.

$$\begin{bmatrix} 3 & 4 & 5 \\ 2 & 8 & 3 \\ 4 & 2 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 3 \cdot 1 + 4 \cdot 2 + 5 \cdot 3 \\ 2 \cdot 1 + 8 \cdot 2 + 3 \cdot 3 \\ 4 \cdot 1 + 2 \cdot 2 + 2 \cdot 3 \end{bmatrix} = \begin{bmatrix} 26 \\ 27 \\ 14 \end{bmatrix}$$



Important rule: This means that the length of the vector **must equal** the number of columns of the matrix.

Matrices and matrix operations

Example: Compute $A\mathbf{x}$, where

$$A = \begin{bmatrix} 1 & 3 \\ -2 & 4 \\ 1 & 5 \end{bmatrix}, \text{and } \mathbf{x} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Matrices and matrix operations

Example: Compute $A\mathbf{x}$, where

$$A = \begin{bmatrix} 1 & 3 \\ -2 & 4 \\ 1 & 5 \end{bmatrix}, \text{and } \mathbf{x} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ -2 & 4 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 + 3 \cdot 1 \\ -2 \cdot 2 + 4 \cdot 1 \\ 1 \cdot 2 + 5 \cdot 1 \end{bmatrix} = \begin{bmatrix} 2 + 3 \\ -4 + 4 \\ 2 + 5 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 7 \end{bmatrix}$$

Matrices and matrix operations

Pseudocode and complexity: intuition and estimation - counting multiplications/additions, what is a rough estimate of the complexity?

$$\begin{bmatrix} 1 & 3 \\ -2 & 4 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 + 3 \cdot 1 \\ -2 \cdot 2 + 4 \cdot 1 \\ 1 \cdot 2 + 5 \cdot 1 \end{bmatrix} = \begin{bmatrix} 2 + 3 \\ -4 + 4 \\ 2 + 5 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 7 \end{bmatrix}$$

Matrices and matrix operations

Pseudocode and complexity: Let *A* be *n* x *n* and let **x** be length *n*

```
In [244]: y = []              # initialize output vector
     ...: n = len(A)
     ...: for i in range(0,n):
     ...:   y.append(A[i][0]*x[0])
     ...:   for j in range(1,n):
     ...:       y[i] = y[i] + A[i][j]*x[j]
```

Matrices and matrix operations

Pseudocode and complexity: Let *A* be *n* x *n* and let **x** be length *n*

```
In [244]: y = []              # initialize output vector
     ...: n = len(A)
     ...: for i in range(0,n):
     ...:   y.append(A[i][0]*x[0])
     ...:   for j in range(1,n):
     ...:       y[i] = y[i] + A[i][j]*x[j]
```

- Count additions and multiplications. (Usually, we count FLOPs (floating point operations) instead.)
- **Complexity:** $= \sum_{i=1}^{n} 1 \sum_{j=1}^{n} 2 = \sum_{i=1}^{n} 2n = 2n^2$

Matrices and matrix operations

FYOG: Modify the psuedocode to multiply a generic m x n rectangular matrix A by a length n vector $\mathbf{x}$.

FYOG: Compute the associated operation count/complexity.

Multiplying matrices: think of it as doing a few matrix-vector multiplications

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} 2 \\ 3 \\ 1 \end{matrix}} & \boxed{\begin{matrix} 1 \\ -1 \\ 0 \end{matrix}} & \boxed{\begin{matrix} 0 \\ 2 \\ -2 \end{matrix}} \end{bmatrix} = ?$$

$$AB = A[\mathbf{b}_1 \ \mathbf{b}_2 \ \mathbf{b}_3] = [A\mathbf{b}_1 \ A\mathbf{b}_2 \ A\mathbf{b}_3]$$

Multiplying matrices: think of it as doing a few matrix-vector multiplications

$$\begin{bmatrix} \boxed{1 \quad 3 \quad -1} \\ -2 \quad 4 \quad 0 \\ 1 \quad 5 \quad 2 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} 2 \\ 3 \\ 1 \end{matrix}} & \begin{matrix} 1 \\ -1 \\ 0 \end{matrix} & \begin{matrix} 0 \\ 2 \\ -2 \end{matrix} \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 + 3 \cdot 3 + -1 \cdot 1 & \_ & \_ \\ \_ & \_ & \_ \\ \_ & \_ & \_ \end{bmatrix}$$

Multiplying matrices: think of it as doing a few matrix-vector multiplications

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & -1 & 2 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 + 3 \cdot 3 + -1 \cdot 1 & \_ & \_ \\ \_ & \_ & \_ \\ \_ & \_ & \_ \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & -1 & 2 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 10 & \_ & \_ \\ -2 \cdot 2 + 4 \cdot 3 + 0 \cdot 1 & \_ & \_ \\ \_ & \_ & \_ \end{bmatrix}$$

Multiplying matrices: think of it as doing a few matrix-vector multiplications

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & -1 & 2 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 1 \cdot 2 + 3 \cdot 3 + -1 \cdot 1 & \_ & \_ \\ \_ & \_ & \_ \\ \_ & \_ & \_ \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & -1 & 2 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 10 & \_ & \_ \\ -2 \cdot 2 + 4 \cdot 3 + 0 \cdot 1 & \_ & \_ \\ \_ & \_ & \_ \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & -1 & 2 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 10 & \_ & \_ \\ 8 & \_ & \_ \\ 1 \cdot 2 + 5 \cdot 3 + 2 \cdot 1 & \_ & \_ \end{bmatrix}$$

Multiplying matrices: think of it as doing a few matrix-vector multiplications

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & -1 & 2 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 10 & -2 & 8 \\ 8 & -6 & 8 \\ 19 & -4 & 6 \end{bmatrix}$$

- **Question:** What must be the dimensions of *A* and *B* for this to work?

Multiplying matrices: think of it as doing a few matrix-vector multiplications

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & -1 & 2 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 10 & -2 & 8 \\ 8 & -6 & 8 \\ 19 & -4 & 6 \end{bmatrix}$$

- **Question:** What must be the dimensions of *A* and *B* for this to work?
- **Answer:** Need the # columns of *A* to match the # rows of *B*.

- **Question:** What are the dimensions of $C = AB$?

Multiplying matrices: think of it as doing a few matrix-vector multiplications

$$\begin{bmatrix} 1 & 3 & -1 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix} \begin{bmatrix} 2 & 1 & 0 \\ 3 & -1 & 2 \\ 1 & 0 & -2 \end{bmatrix} = \begin{bmatrix} 10 & -2 & 8 \\ 8 & -6 & 8 \\ 19 & -4 & 6 \end{bmatrix}$$

- **Question:** What must be the dimensions of *A* and *B* for this to work?
- **Answer:** Need the # columns of *A* to match the # rows of *B*.

- **Question:** What are the dimensions of *C* = *AB* ?
- **Answer:** *C* = *AB* will have the same # rows as *A* and # columns as *B*.

- **Summary:** If *A* is *n x k* and *B* is *k x m* then *C* = *AB* is *n x m*.

Complexity: Multiply *A* and *B*, both *n x n* matrices.

$$AB = A[\mathbf{b}_1 \;\; \mathbf{b}_2 \;\; \mathbf{b}_3] = [A\mathbf{b}_1 \;\; A\mathbf{b}_2 \;\; A\mathbf{b}_3]$$

Complexity: Multiply *A* and *B*, both *n x n* matrices.

$$AB = A[\mathbf{b}_1 \ \mathbf{b}_2 \ \mathbf{b}_3] = [A\mathbf{b}_1 \ A\mathbf{b}_2 \ A\mathbf{b}_3]$$

- Each column of $C = AB$ is a “mat-vec”
- We saw these each require ~$2n^2$ FLOPs
- And we have n columns to do

  ⇒ Total mat-mat is ~$n \times 2n^3 = 2n^3$ FLOPs

Complexity: Multiply *A* and *B*, both *n x n* matrices.

$$AB = A[\mathbf{b}_1 \ \mathbf{b}_2 \ \mathbf{b}_3] = [A\mathbf{b}_1 \ A\mathbf{b}_2 \ A\mathbf{b}_3]$$

- Each column of $C = AB$ is a "mat-vec"
- We saw these each require ~$2n^2$ FLOPs
- And we have n columns to do

  ⇒ Total mat-mat is ~$n \times 2n^3 = 2n^3$ FLOPs

Summary:

- Matrix addition is $\mathcal{O}(n^2)$
- Matrix-vector multiplication (mat-vec) is $\mathcal{O}(n^2)$
- Matrix-matrix multiplication (mat-mat) is $\mathcal{O}(n^3)$

Algorithm complexity and matrix operations

FYOG: Determine the complexity of matrix-matrix multiplication for rectangular matrices A of size m x n and B of size n x k.

FYOG: Write pseudocode (or better yet, actual Python code!) for multiplying two such matrices.

FYOG: Redo the operation count based on your (pseudo)code.

Algorithm complexity and matrix operations

Extra special FYOG: There is a special kind of a square matrix called **lower-triangular**, where all of the elements above the **main diagonal** are 0.

- **Example:** $L = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 4 & 0 \\ 1 & 5 & 2 \end{bmatrix}$

S'pose we want to compute $L\mathbf{x}$, where $\mathbf{x}$ is an appropriately-sized vector. Any time an entry of $\mathbf{x}$ hits one of the upper 0s, we already know the result will be 0. So we don't want to waste time computing those multiplications.

- **TODO:** Modify your mat-vec code to skip the unnecessary multiplications, assuming a lower-triangular matrix is used.
- **TODO:** Determine the complexity of the new algorithm when L is n x n.

Algorithm complexity and matrix operations

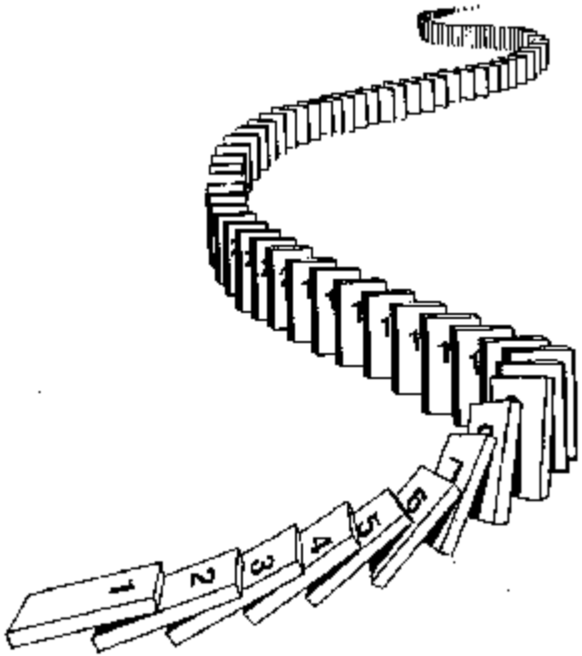
Recap:

- We can multiply/add matrices and matrices,
- and multiply matrices and vectors,
- and estimate the complexity of these operations.

Next time:

- Back to proofs, but with flavors of recursion and sequences:

  A powerful and mystical proof strategy: **by induction**



Bonus material!