# CS 2500: Algorithms

## Lecture 2: Basics of Algorithm Analysis

Shubham Chatterjee

Missouri University of Science and Technology, Department of Computer Science

August 22, 2024

# Logistics

## Recitations

- Recitations are **NOT** mandatory.
- BUT you are **strongly encouraged** to attend the recitations.
- We have reserved a room 2 days a week for recitations.
  - Where: Toomey Hall 295
  - When: Every Wednesday and Thursday, 4 PM - 5 PM.
- GTAs will hold the recitation sessions.
- You will be given two types of homework assignments:
  1. Weekly assignments (due every Tuesday).
  2. Quick assignments (due the next day).
- GTA will go over the **quick assignments** and help you work through them.
- **No recitation today!**

# Grading Information [Updated]

- Longer (Weekly) Assignments. [40%]
  - Problems + coding (2-3 total).
  - Released every Tuesday.
  - Due at midnight next Tuesday.
  - Turnaround time: 7 days
  - **No homework this week!**
- Quick Assignments. [10%]
  - 2-3 (simple) problems based on material discussed in class that day.
  - Released on the day of the class.
  - Due next day.
  - Turnaround time: 24 hours
  - **No homework this week!**
- 1 midterm examination. [25%]
- 1 final comprehensive examination. [25%]

# Logistics

**Office Hours**

- **Shubham Chatterjee (Instructor)**
  - When: Tuesday and Thursday, 12.30 PM - 1.30 PM.
  - Where: Room 320, Computer Science Building

**Other meetings by appointment.**

# Logistics

## My Contact

- **Email:** shubham.chatterjee@mst.edu
- **Office:** Room 320, Computer Science Building

## Recommended Books

- **Algorithms Illuminated: Omnibus Edition**. *Tim Roughgarden*. Asked the library to reserve the book.
- **Fundamentals of Algorithmics**. *Gilles Brassard and Paul Bratley*.

**Important**

**You are STRONGLY ENCOURAGED to read at least one textbook.**

# Counting in Algorithm Analysis

Many times, algorithm analysis is reduced to counting of steps or operations. The following are some of the common summations that are encountered in algorithm analysis:

$$\sum_{k=0}^{n} c = cn$$

$$\sum_{k=0}^{n} 1 = n + 1$$

Some of the additional useful rules are as follows:

$$\sum_{k=1}^{n} k = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$$

# Counting in Algorithm Analysis

$$\sum_{k=1}^{n} k^2 = 1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{n} k^3 = 1^3 + 2^3 + 3^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2$$

# Counting: Example

## Question

Let us assume that the analysis of an algorithm yields $f(n)$ as follows:

$$f(n) = \sum_{i=1}^{n} 6i(i+1)$$

What is the estimate of the total number of operations?

# Counting: Example

## Solution

The given equation can be written as:

$$
\begin{aligned}
f(n) &= \sum_{i=1}^{n} 6i^2 + \sum_{i=1}^{n} 6i \\
&= 6 \times \frac{n(n+1)(2n+1)}{6} + 6 \times \frac{n(n+1)}{2} \\
&= n(n+1)(2n+1) + 3(n+1) \\
&= 2n^3 + 6n^2 + 4n
\end{aligned}
$$

Therefore the time complexity function of the given algorithm is: $f(n) = 2n^3 + 6n^2 + 4n$

# Counting Example: Linear Search

**Remember the example about trying to find a book in a library?**

## Question

**Problem:** Find the best, worst, and average case complexities for a linear search algorithm.

# Counting Example: Linear Search

## Solution

- **Best Case:** The element is the first item in the list. So $\mathbf{O(1)}$.
- **Worst Case:** The element is the last item or is not present in the list. So $\mathbf{O(n)}$.

# Counting Example: Linear Search

## Solution

**Average Case.** The average-case complexity for a linear search problem can be derived as follows:

- The item can be present anywhere in the list.
- Probability of the item being in any position $k$ is $\frac{1}{n}$.
- If the item is at position $k$, the number of comparisons required is $k$.

The expected number of comparisons $f(n)$ is given by:

$$f(n) = \frac{1}{n} \times (1 + 2 + 3 + \cdots + n) = \frac{1}{n} \times \frac{n(n+1)}{2} = \frac{n+1}{2}$$

**Conclusion:** The average-case complexity is $\mathbf{O(n)}$.

# Rate of Growth

## Question

Let $A$ and $B$ be two algorithms with the following complexity functions:

$$f_A = n^2, \quad f_B = 40n + 1200$$

Which algorithm is better?

# Rate of Growth

## Solution

To find the instance where $f_A = f_B$, solve the equation:

$$n^2 = 40n + 1200$$

Rearranging gives us the quadratic equation:

$$n^2 - 40n - 1200 = 0$$

Solving this using the quadratic formula:

$$n = \frac{40 \pm \sqrt{1600 + 4800}}{2} = \frac{40 \pm \sqrt{6400}}{2}$$

$$n = 60 \quad (\text{since } n > 0)$$

# Rate of Growth

## Solution

**Conclusion:**

- For $n < 60$, Algorithm $A$ is better.
- For $n > 60$, Algorithm $B$ is better.

**See the Visualization in Action:**

Click Here to View the Animation

# Tractable vs. Intractable Problems

- Tractable problems are those that generally have polynomial-time complexity.
- Intractable problems typically have exponential or worse time complexities, making them impractical to solve within a reasonable time for large input sizes.

# Tractable Problems: Polynomial-Time Algorithms

- These algorithms have complexities like $O(n)$, $O(n^2)$, $O(n^3)$, etc.
- Polynomial-time algorithms are generally considered efficient.
- Problems that can be solved by such algorithms are known as tractable problems.
- Example: Sorting algorithms like Merge Sort, with complexity $O(n \log n)$, are tractable.

# Intractable Problems: Exponential-Time Algorithms

- These algorithms have complexities like $O(2^n)$, $O(n!)$, etc.
- Exponential-time algorithms are considered inefficient for large input sizes.
- Example: The Traveling Salesman Problem (TSP) is a classic example of an intractable problem.

# Example: Exponential Growth

**Consider the story of the chessboard and the grains of rice**.

- On the first square, place 1 grain.
- On the second square, place 2 grains.
- On the third square, place 4 grains.
- Continue this pattern, doubling the grains on each subsequent square.

**Question:** How many grains will be on the 64th square?

# Example: Exponential Growth

**Grains on the 64th square:**

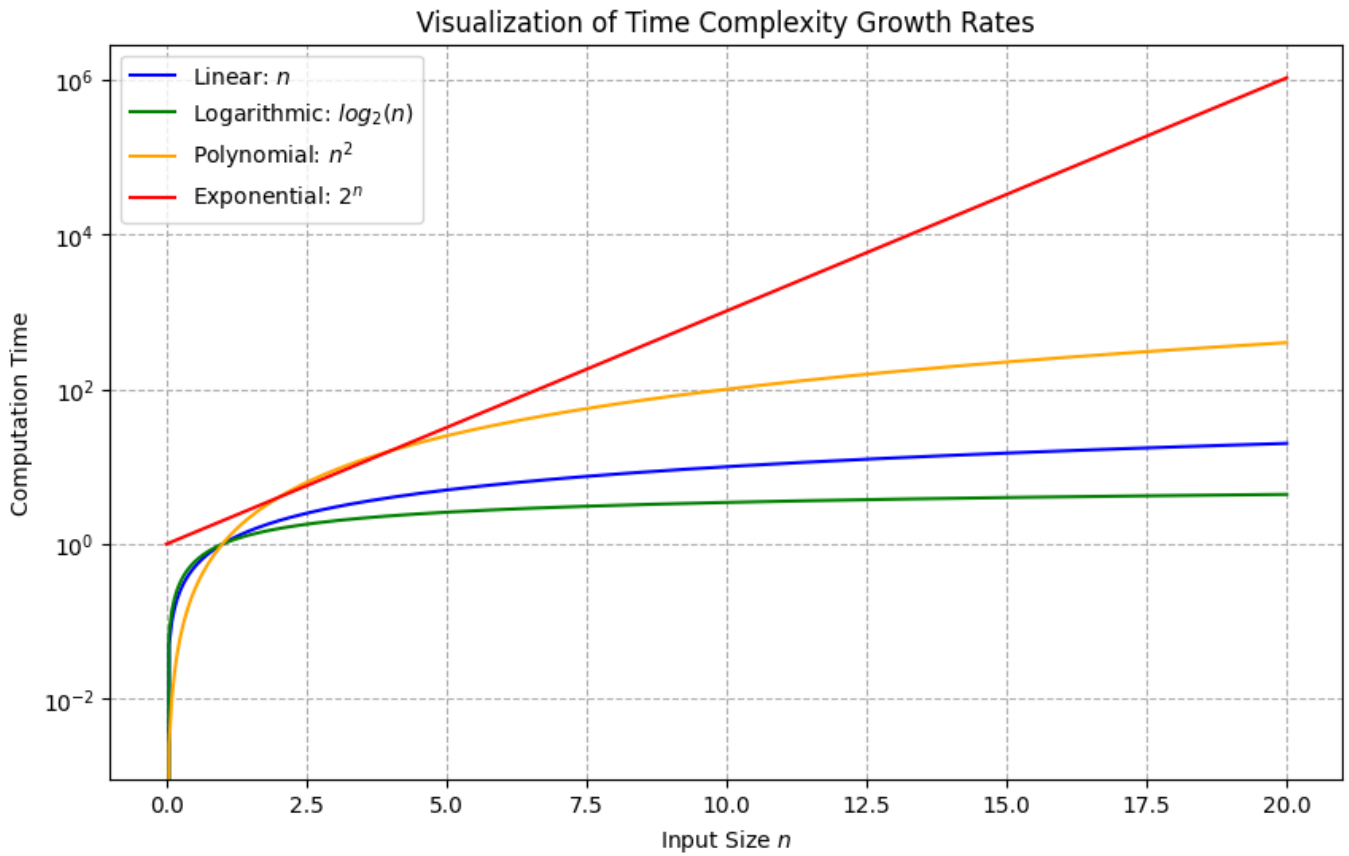
- The number of grains on the $n$-th square is given by $2^{n-1}$.
- For the 64th square, the number of grains is $2^{63}$.

**Implication:** The total number of grains on the chessboard becomes astronomically large, illustrating how quickly exponential growth can escalate.

# Consequences of Exponential Growth

**Impact on Algorithms:**

- Exponential growth in time complexity means that even a small increase in input size can result in a dramatic increase in the computation time.



# Consequences of Exponential Growth

**Impact on Algorithms:**

- Exponential growth in time complexity means that even a small increase in input size can result in a dramatic increase in the computation time.
- Such algorithms may take years or even centuries to complete for large input sizes.

**Conclusion:** Intractable problems are generally unsolvable in practice due to their extreme computational requirements.

# Asymptotic Analysis

**Asymptotic analysis** is the theory of approximation.

Consider the Gauss summation:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

This is an exact formula, with a "closed-form" solution. In real-world situations, finding an exact formula is often not possible.

Asymptotic analysis is very effective in algorithm analysis, especially when it's difficult to determine the exact time complexity curve for large input sizes.

# Why Asymptotic Analysis?

- Asymptotic analysis helps approximate the time complexity of algorithms, especially for large input sizes.
- It focuses on how input-dependent parameters affect the algorithm's performance.
- This approach is essential when exact formulas are difficult or impossible to obtain.

# Asymptotic Notation: Big-O

- Upper bound or worst case time complexity of an algorithm.
- Let $f$ and $g$ be functions from the integers or the real numbers to the real numbers. The function $f(n)$ is $O(g(n))$ if there exist positive constants $C$ and $k$ such that:

$$|f(n)| \leq C \cdot |g(n)| \text{ for all } n > k$$

# Big-O: Example 1

## Question 1

Let $f(n) = 3n^3$ for an algorithm. Prove that $f(n)$ is in $O(n^3)$

# Big-O: Example 1

## Solution 1

- Let $g(n) = n^3$.
- Choose $C = 4$ and $k = 1$.
- For $n > 1$, $3n^3 \leq 4n^3$, so $f(n)$ is $O(n^3)$.

# Example 1: Why Choose $g(n) = n^3$?

**Objective:** Prove that $f(n) = 3n^3$ is in $O(n^3)$.

**Why $g(n) = n^3$?**

- **Form of the Function:** $f(n) = 3n^3$ is a cubic function, so the most natural upper bound is also a cubic function.
- **Tight Bound:**
  - Big-O notation aims to provide the tightest possible upper bound on the growth of a function.
  - Choosing $g(n) = n^4$ or $n^5$ would still be correct but less informative because it would not be the tightest bound.

# Example 1: Why Choose $g(n) = n^3$?

- **Simplifying Comparison:**
  - To show $f(n)$ is in $O(n^3)$, we need to find a constant $c$ such that $f(n) \leq c \cdot g(n)$ for sufficiently large $n$.
  - With $g(n) = n^3$, the comparison becomes:

$$3n^3 \leq 3 \cdot n^3$$

  - This satisfies the Big-O condition easily with $c = 3$.

**Conclusion:** $g(n) = n^3$ is the best choice because it provides the most accurate and tightest upper bound for $f(n) = 3n^3$.

# Big-O: Example 2

## Question 2

Let $f(n) = 3n^3 + 2n^2 + 3$ for an algorithm. Prove that $f(n)$ is in $O(n^3)$

# Example 2: Proving $f(n) = 3n^3 + 2n^2 + 3$ is $O(n^3)$

**Problem Statement:**
Prove that $f(n) = 3n^3 + 2n^2 + 3$ belongs to the Big-O class $O(n^3)$.

**Definition of Big-O:**
$f(n)$ is $O(g(n))$ if there exist positive constants $c$ and $k$ such that for all $n \geq k$:

$$f(n) \leq c \cdot g(n)$$

**Target Function:**

$$f(n) = 3n^3 + 2n^2 + 3$$

We need to show that $f(n) \leq c \cdot n^3$ for some constant $c$.

# Example 2: The Key Step: Grouping Terms

Notice that the term with the highest degree of $n$ dominates the growth of the function as $n$ becomes large.

In this case, $n^3$ is the highest degree term. The other terms $2n^2$ and 3 grow much slower than $n^3$.

To express $f(n)$ in a form that makes it easier to compare to $n^3$, consider the following:

$$f(n) = 3n^3 + 2n^2 + 3$$

# Example 2: Comparing Each Term to $n^3$

Each of these terms can be compared to $n^3$:

- $3n^3$ is already a term of the same form as $n^3$.
- $2n^2$ can be approximated by a term proportional to $n^3$ because, for large $n$, $n^3$ grows much faster than $n^2$.
- 3 can be approximated by a term proportional to $n^3$ because, for large $n$, $n^3$ dwarfs the constant term 3.

# Example 2: Expressing $f(n)$ in Terms of $n^3$

So, we can write:

$$f(n) = 3n^3 + 2n^2 + 3 \leq 3n^3 + 2n^3 + 3n^3$$

This simplifies to:

$$f(n) \leq (3 + 2 + 3)n^3 = 8n^3$$

# Example 2: Conclusion

Since $f(n) \leq 8n^3$ for sufficiently large $n$, we can choose $c = 8$.

Therefore, $f(n) = 3n^3 + 2n^2 + 3$ is $O(n^3)$.

This demonstrates that the function's growth is bounded above by a constant multiple of $n^3$, confirming that it belongs to the Big-O class $O(n^3)$.

# Big-O: Example 3

## Question 3

Let $f(n) = \frac{2n^3+13log_2 n}{7n^2}$ for an algorithm. Prove that $f(n)$ is in $O(n)$

# Big-O: Example 3

## Solution 3

It can be observed that $log_2 n < n$ is always true. Therefore, one can argue that that $13 log_2 n \leq 13n$.

Since $13n \leq 13n^3$ is always true, we can rewrite $f(n)$ as:

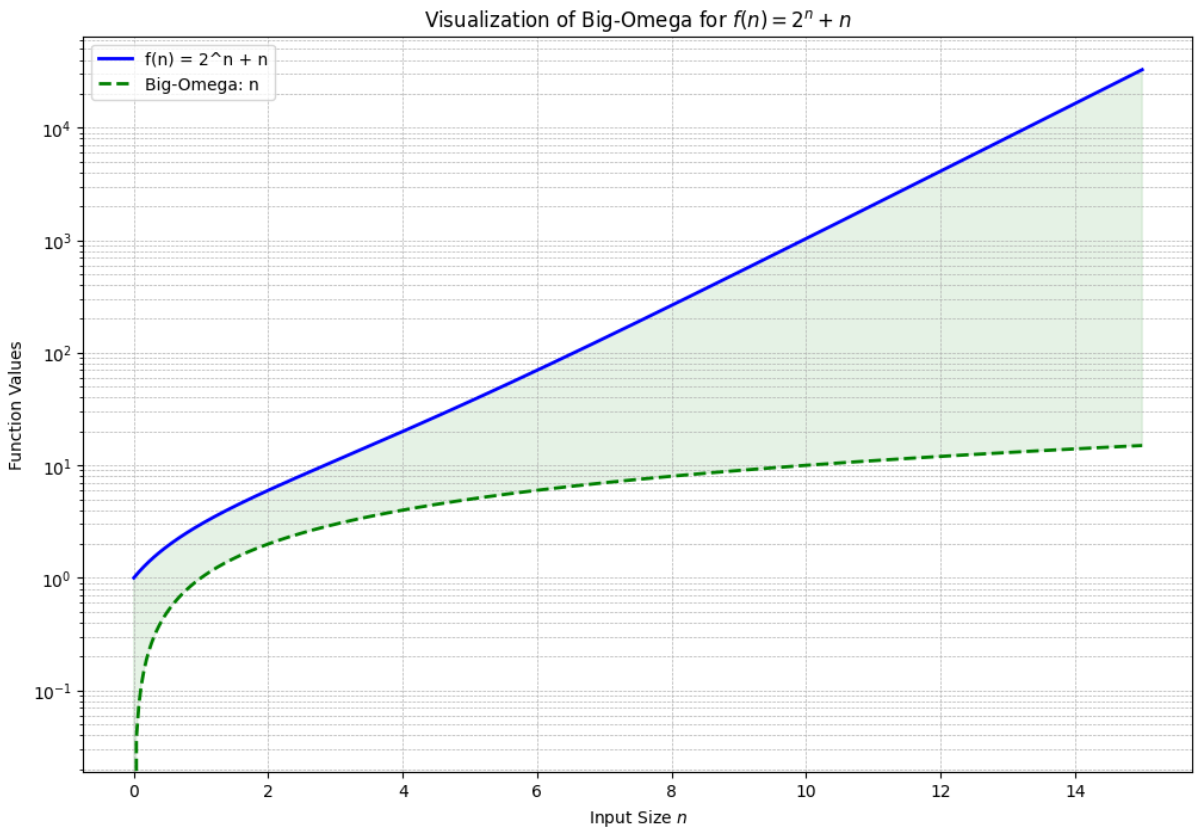
$$f(n) \leq \frac{15n^3}{7n^2} = 2n \quad \forall n > 1$$

So $f(n) = O(n)$ for $c = 2$. Hence proved.

# Asymptotic Notation: Big-Omega ($\Omega$)

**Big-Omega:** Let $f$ and $g$ be functions from the integers or the real numbers. The function $f(n)$ is $\Omega(g(n))$ if there exist positive constants $C$ and $k$ such that:

$$f(n) \geq C \cdot g(n) \text{ for all } n > k$$



# $\Omega$: Example

**Example**

Let $f(n) = 4n^4 + 3n^3 + 2n + 1$ for an algorithm. Prove that $f(n)$ is in $\Omega(n)$

# $\Omega$ Example: Understanding $\Omega$ Notation

**Big-Omega Notation $\Omega(g(n))$:**

- Provides a **lower bound** on the growth rate of a function.
- Our goal: Find the Big-Omega notation for $f(n) = 4n^4 + 3n^3 + 2n + 1$.

# $\Omega$ Example: Analyzing the Function $f(n) = 4n^4 + 3n^3 + 2n + 1$

**Breakdown of $f(n)$:**

- $4n^4$ is the term with the highest degree; it dominates as $n$ increases.
- $3n^3$ is the second-highest degree term.
- $2n$ grows linearly.
- 1 is a constant term.

**See the Visualization in Action:**

Click Here to View the Animation

# $\Omega$ Example: Why Not Choose $g(n) = n$?

**Growth Comparison:**

- $3n^3$ grows much faster than $n$.
- If we choose $g(n) = n$, we would be stating that $f(n)$ grows at least as fast as $n$, which is true but not informative. This is because $n$ is much smaller compared to $3n^3$ and $4n^4$, so it doesn't provide a meaningful or tight lower bound for $f(n)$.

# $\Omega$ Example: Why Not Choose $g(n) = n$?

**Tightness of Bound:**

- A lower bound is more informative if it is as close as possible to the function's actual growth rate.
- Choosing $g(n) = n$ would be an overly loose bound, meaning it doesn't accurately reflect the growth behavior of $f(n)$.

# $\Omega$ Example: Why Choose $g(n) = n^3$?

**Big-Omega Bound:**

- $f(n)$ includes a $3n^3$ term.
- $f(n)$ must grow at least as fast as $3n^3$, meaning $f(n) = \Omega(n^3)$.

**Reasonable Tightness:**

- By choosing $g(n) = n^3$, we establish a lower bound that is close to the actual growth rate of the function, providing a more precise and informative description of $f(n)$'s behaviour.

# $\Omega$ Example: Proving $f(n) = \Omega(n^3)$

**Formal Proof:**

$$f(n) = 4n^4 + 3n^3 + 2n + 1 \geq 3n^3$$

**Conclusion:**

- For sufficiently large $n$, $f(n) \geq c \cdot n^3$ where $c = 3$.
- Therefore, $f(n) = \Omega(n^3)$.