ALGORITHMS

INTRODUCTION

DR. MUDASSIR SHABBIR

LUMS UNIVERSITY

JANUARY 19, 2026

Outline

- Introduction
- Course Logistics
- Runtime Analysis

About Your Instructor

- Hi there! I'm **Dr. Mudassir Shabbir**.
- I am an Associate Professor at CS.
- Before that ...

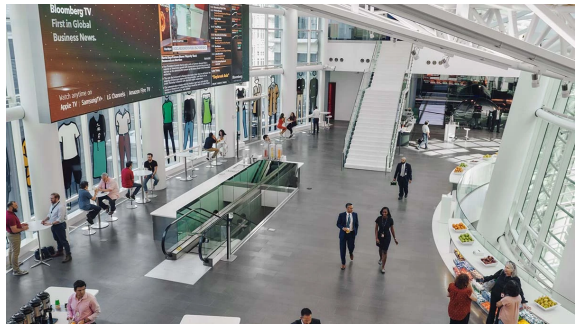
About Your Instructor

Rutgers

Vanderbilt

Los Alamos Nat. Lab.

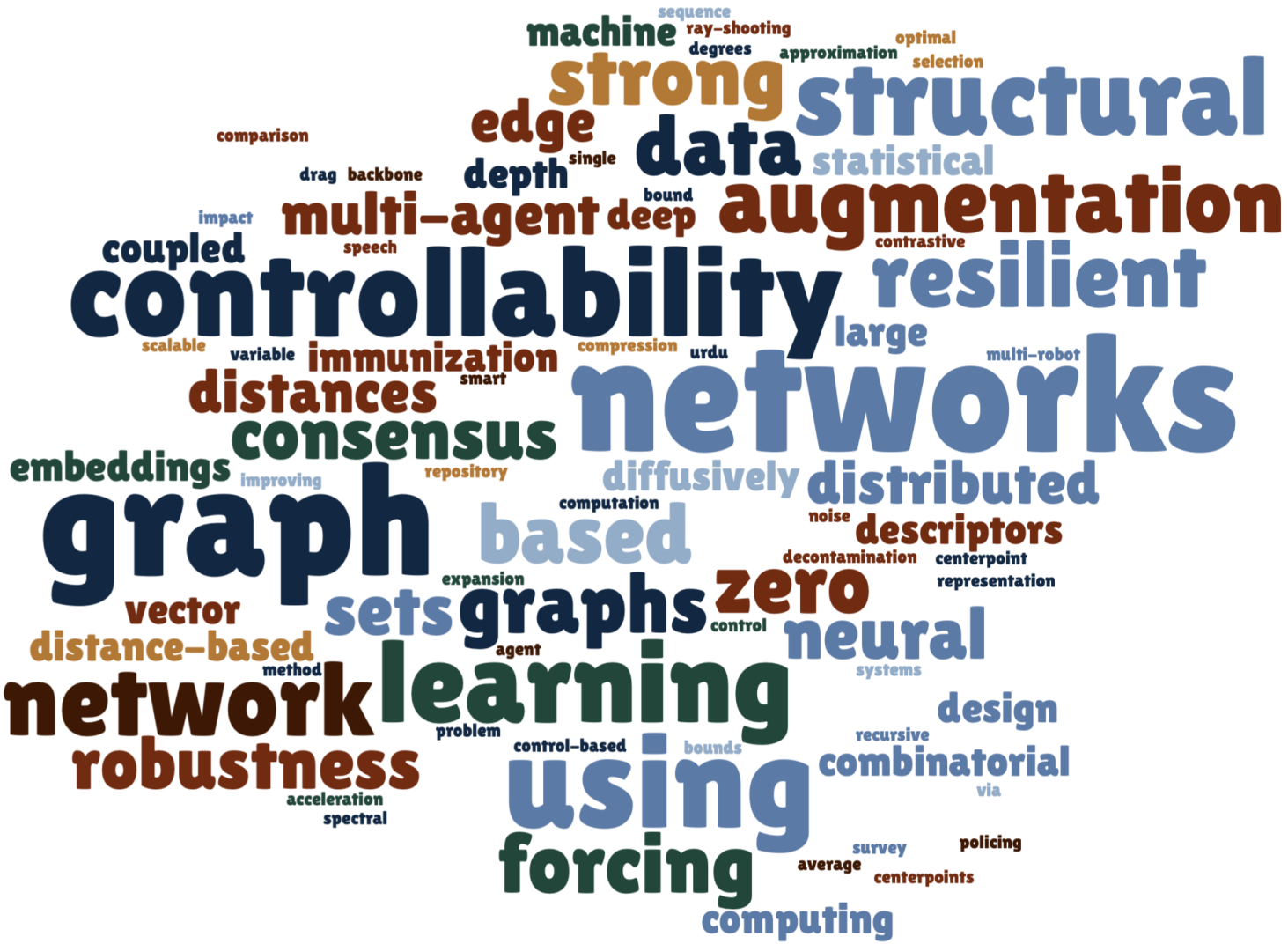
ITU



Bloomberg

Punjab Univ.

About Your Instructor



About Teaching Staff

I will be co-teaching this course with **Aamina Jamal Khan**, with the help of Teaching Assistants:

- Abdullah Mushtaq
- Monish Kumar
- Mir Huzaifa
- Zain Ahmed
- Hafiz Muhammad Ahmad
- Meesum Ali
- Manaal Malik
- Saif Shakir
- Muhammad Ali Shah
- Zainab Usman
- Muhammad Reyyan Asad

CS310/5102 Teaching Club

Aamina Jamal Khan

Abdullah Mushtaq

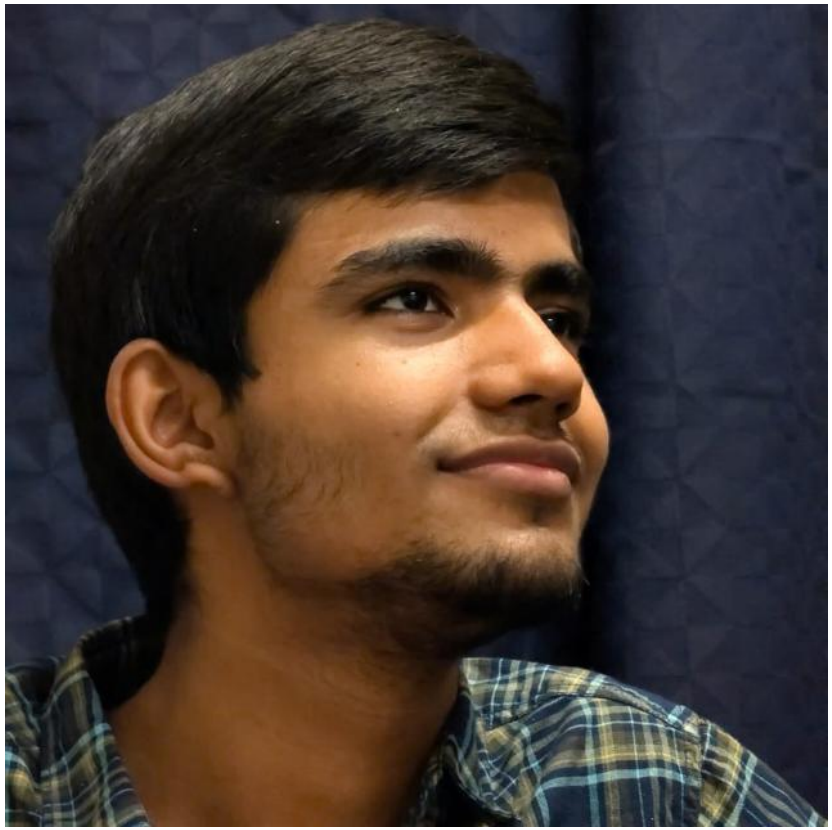
CS310/5102 Teaching Club

Monish Kumar

CS310/5102 Teaching Club

Mir Huzaifa

CS310/5102 Teaching Club



Zain Ahmed

CS310/5102 Teaching Club

Hafiz Muhammad Ahmad

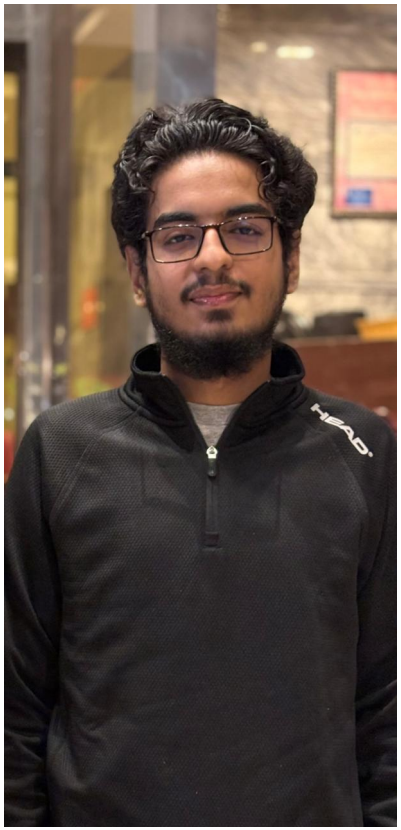
Meesum Ali

Manaal Malik

CS310/5102 Teaching Club

Saif Shakir

CS310/5102 Teaching Club



Muhammad Ali Shah

CS310/5102 Teaching Club

Zainab Usman

CS310/5102 Teaching Club

Muhammad Reyyan Asad

Reaching Out

- Keep up with course announcements on LMS and **Slack**. Join here: http://tiny.cc/slack310
- Office Hours: **MW 12pm to 1:00pm**.
- Email: **mudassir.shabbir@lums.edu.pk**. ALWAYS write CS310 in the subject line alongwith the main subject matter.
- Always CC: **"Aamina Jamal Khan" <aamina.khan@lums.edu.pk>** and **TA team** in your emails.
- Email: ALWAYS follow below format for emails.

Subject: Question Regarding HW 3 Problem 1 | Discrete Structures
Dear Dr. Mudassir,

My name is Sophie (ID: BSCS23439), and I am student in your Discrete class section A. I am writing to ask for your guidance on a problem I encountered while working on the homework 3 Problem 1.

Specifically, I am having difficulty with [briefly describe the problem or the specific part of the problem you're struggling with]. I have reviewed the course materials and attempted to approach the problem in various ways, but I am still uncertain about [what exactly you are unsure about—e.g., the method, the concept, or the solution].

Could you please provide some clarification or suggest any resources that might help me better understand this issue? I would greatly appreciate any advice or direction you could offer.

Thank you for your time and assistance.

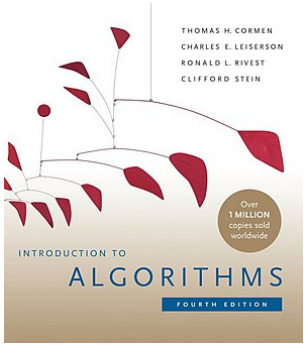
- Office Location: **SSE 9-G47A**

Rules of the Game

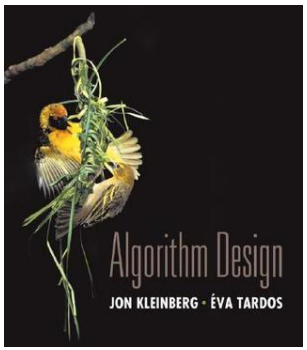
- You can either call me Dr. Mudassir or you can call me Professor or just Mudassir.
- Turn off your phones and other electronic stuff.
- Be on Time.
- No whispering; talk to me instead.
- Be honest.
- Register yourself for PollEv at PollEv.com/cs310

Recommended Algorithm Books

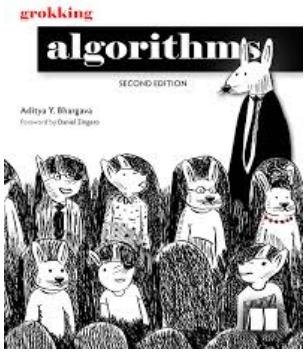
- **Introduction to Algorithms**
  Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein
  4th Edition, MIT Press
  "The Bible" of algorithms - rigorous and comprehensive.



- **Algorithm Design**
  Jon Kleinberg, Eva Tardos
  1st Edition, Pearson
  Focuses on design paradigms and problem-solving intuition.



- **Grokking Algorithms**
  Aditya Bhargava
  1st Edition, Manning
  Visual, beginner-friendly, great for building intuition.



Grading

Component	Weight
Exams	20+25+30%
Homework Assignments	10%
Classroom Worksheets	10%
Lecture Notes	0%
Slack/Github Participation	5%
Total	100%

Grading Scale

The following table should only give you an idea of a typical grade distribution. Note that final cutoff points will be determined via relative grading at the end of the semester.

Average	Assigned Grade
≥ 90	A category
80 - 89.9	B category
70 - 79.9	C category
60 - 69.9	D category
< 60	F

Exams

Three Exams

Midterm I	20%
Midterm II	25%
Final Exam	30%
Total	75%

- Midterm exams will be held in-person during regular class hours.
- No make-up exams.

Homework Assignments

Weekly / Biweekly Problem Sets

Homework	10%

- Use LaTeX.
- Free to discuss but you must write your own solutions.
- No late submissions.
- LUMS academic honesty policy will be strictly enforced.
- Please cite all resources used.
- Be able to defend your solutions via oral or written explanation.
- Questions from homework may appear in exams.

In-Class Activities

Worksheets	10%

- Short exercises done during lectures—participation matters.
- Almost weekly.
- No make-up for missed worksheets.
- Can be done in pairs.

Student-Created Notes

Lecture Notes	0%

Quality, clarity, and collaboration in shared notes.

- Clone github repo: `https://github.com/amessbee/310S26`
- Create a separate branch(use your ID) from main for your lecture notes.
- Email (and tag) to TF/TAs.
- Separate announcement to Slack channel.

Online Engagement

Slack/Github Participation	2+2+1%

Asking questions, helping peers, creating live commentaries of your work, and fix/improve lecture notes.

Class Outcomes

By the end of this course, you will be able to:

- Explore a computational problem, assess its requirements, and apply an appropriate algorithmic approach.
- Analyze your algorithm by calculating its running time and expressing it using asymptotic (order) notation.

Secondary Outcomes

By the end of this course, you will also:

- Gain a perspective to appreciate the theoretical foundations of computer science , i.e., be more inclined to opt for electives I am going to offer in Fall
- Be better prepared for technical interviews in software and data-oriented roles.
- Become more skilled at giving and receiving professional, constructive critiques on technical work.
- Become comfortable using modern, industry-standard tools for online collaborative teamwork.

Lecture Breakdown (Tentative)

Weeks 1–7	Weeks 8–14
Week 1: Introduction + Asymptotic Notation	Week 8: Dynamic Programming
Week 2: Order Statistics + Randomized Analysis + Prune-and-Search	Week 9: Dynamic Programming
Week 3: Sorting + Divide-and-Conquer	Week 10: Flows + Exam 2
Week 4: Sorting Lower Bounds + Basic Graphs & Greedy Algorithms	Week 11: Flows + Cuts (cont.)
Week 5: MST + Exam 1	Week 12: NP-completeness
Week 6: MST + Shortest Paths	Week 13: NP-completeness (cont.)
Week 7: Shortest Paths	Week 14: Approximation Algorithms

Tentative Topics for Review Sessions

- Solving Recurrences
- Sorting Algorithms
- Graph Basics
- Minimum Spanning Trees

Some Tips

The course is mile wide and foot deep. There will be a lot of new concepts/topic almost every week. So, Do not fall behind.

Participate, do not be shy to ask questions, So, Be active.

Often students say "I understood everything in class, but am unable to solve problems". The secret is Practice, practice, and more practice.

Runtime Analysis

A Simple Example

Problem: Given a list of n (distinct) numbers, find the largest one.

Example: Consider the list

$$[3, 7, 2, 9, 5]$$

The largest number is

$$9.$$

Different Implementations

- One could write this in Python:

```
largest = lst[0]
for x in lst:
    if x > largest:
        largest = x
```

- Or in C++:

```
int largest = lst[0];
for(int i=0;i<n;i++){
    if(lst[i] > largest)
        largest = x[i];
}
```

- Or using a built-in function:

Why We Need Algorithm Analysis

- We want a measure of efficiency independent of:
  - Programming language
  - Hardware
  - Current system load
- We focus on the algorithm itself, not a particular implementation.
- We count **primitive operations** instead.

Definition: An **algorithm** is a precise set of instructions in plain English (or pseudocode) that solves a problem.

Primitive Operations

- What counts as a **primitive operation**? It depends on the problem:
  - Finding the maximum: comparison might be primitive.
  - Sorting: comparison is usually primitive.
  - Moving data in an array: sometimes counted too.
- We only care about the "basic step" that dominates the running time for large inputs.

"We abstract away details that are machine-dependent."

Counting Operations: Our Example

- List has n numbers: $x_1, x_2, \ldots, x_n$
- Algorithm (find largest):
  1. Set `largest = x1`
  2. For each element x_i, compare x_i with `largest` and update if larger
- Count comparisons:

$$\text{Number of comparisons } = n - 1$$

- Independent of programming language or hardware!

Key Takeaways

- Algorithms are abstract; implementation details come later.
- Efficiency is measured by counting primitive steps, not by clock time.
- This allows us to compare algorithms fairly:
  - Example: finding max vs sorting then picking max
- We are laying the groundwork for **time complexity analysis**.

Preview: Complexity Notation

- We will introduce **Big-O notation** to describe how the runtime grows with input size n.
- For our example:

$$\text{finding max} \sim O(n)$$

- Compare with sorting first, then picking max:

$$O(n \log n)$$

Counting primitive operations lets us abstract away implementation details and focus on the algorithm.

Watch video at `http://tiny.cc/yt1`, and at `http://tiny.cc/yt2` at 1.5x speed!

Don't care about absolute exact time.

```
bubbleSort(A)
0.  n = len(A))
1.  for i in range(n):
2.      for j in range(n-1):
3.          if A[j] > A[j + 1]:
4.              temp = A[j]
5.              A[j] = A[j+1]
6.              A[j+1] = temp
7.  return A
```

Upper Bound: $O(.)$ read Big Oh

$$T(n) \leq f(n)$$

Lower Bound: $\Omega(.)$, read Big Omega

$$T(n) \geq g(n)$$

Input Size : len(A)

Tight Bound: $\Theta(.)$, read Big Theta

$$T(n) \geq f(n), \text{ and } T(n) \leq f(n).$$