CSCE 3110
Data Structures and Algorithms

Instructor Information

Instructor Contact Information

Instructor:	Dr. Wenting Wang
Email Address:	wenting.wang@unt.edu
Office Hours Location/Times:	NTDP E260G Monday/Wednesday 1:00 pm– 3:00 pm (or by appointment)

TA/IA	TBD

Course Webpage and Schedule

Course Information

Course Number/Section:	CSCE 3110.001
Course Title:	Data Structures and Algorithms
Credit Hours:	3
Term:	Fall 2024
Meeting Times & Location:	Monday/Wednesday 9:00 am – 10:20 am, NTDP K110

Course webpage can be found at: CANVAS

Textbooks

- Mark Weiss. 2013. Data Structures & Algorithm Analysis in C++. 4 edition. ISBN 9780132847377

- Cormen, Thomas, Charles Leiserson, Ronald Rivest, and Clifford Stein. Introduction to Algorithms. 3rd ed. MIT Press, 2009. ISBN: 9780262033848.

- *Algorithms by Jeff Erickson,1st edition, June 2019*

Objectives

- Understand time and space analysis for both iterative and recursive algorithms and be able to prove the correctness of a non-trivial algorithm.
- Be able to translate high-level, abstract data structure descriptions into concrete code.
- Understand dictionary/search data structures (lists, trees, hash tables).
- Understand graph representations and algorithms.
- Understand how real-world problems map to abstract graph problems.
- Be able to communicate clearly and precisely about the correctness and analysis of basic algorithms (both oral and written communication).

Course Content

Topics include:

- Time and Space analysis (Asymptotic notation)
- Recursion and Recurrence relations
- Review of Basic Data Structures (Lists, stacks, queues, etc.)
- Tree-based data structures, including heaps, BSTs, union/find data structures and AVL trees
- Hashing
- Data structures for storing graphs, elementary graph algorithms (breadth-first search, depth-first search) and their applications
- Algorithms for solving minimum spanning tree problem (Prim's) and their implementations

ABET outcomes

- Understand time complexity of algorithms
- Be able to solve recurrence relations
- Understand and be able to analyze the performance of data structures for searching, including balanced trees, hash tables, and priority queues
- Apply graphs in the context of data structures, including different representations, and analyze the usage of different data structures in the implementation of elementary graph algorithms including
  - depth-first search
  - breadth-first search
  - topological ordering
  - Prim's algorithm
  - Kruskal's algorithm
- Be able to code the above-listed algorithms

Prerequisites

CSCE 2100 Computing Foundations I and CSCE 2110 Computing Foundations II.

- You should be familiar with algorithms
  - loops, recursivity, etc.
  - pseudo-code
- You should know how to program without help in C++
  - design a program, compile, run experiments, etc.

Grading

- Assignments 30%
- In-class Quizzes 10%
- Exam 1 20%
- Exam 2 30%
- Attendance 10%

Note: There is no make-up quiz, assignment, project, or exam. There might be some changes to homeworks, Exam 1 and 2 schedule. I will update this schedule if needed as we proceed.

Grading scale

Tentative grading scale (based on 100 points)

A 90-100 points

B 80-89 points

C 70-79 points

D 60-69 points

F below points

- No absolute grading scale; appropriate letter grade cutoffs set by instructor at the end of semester.

Grading – Assignment (30%)

- There will be six homework assignments
- Assignment grade = average of the five highest-graded homework assignments
  - everybody may drop one
- Assignment is due at the end of the day
- Written and programming exercises (C++)
- All programming will be in C/C++ and must compile on a University Unix/Linux machine. No credit will be given for programs that do not compile.

Important Notes

- All assignments, shall be turned in electronically using Canvas.
- A late penalty of 20% /day will be applied to all late assignments for up to 5 calendar days. Assignments that are not turned in 5 days after the due date will not be accepted.
- All holidays and weekends will be counted as calendar days

Grading: In-class Quizzes

- In-class Quizzes (10%)
  - during class on TBD
  - every two to three weeks
  - 15 – 20 mins
  - may discuss solutions right after the quiz or in the next class
- There will be at least five quizzes
- Quiz grade = average of the $N - 1$ lowest-graded quizzes
  (everyone may skip one with 0)

Grading: Exams

- Exam 1 (20%)
  - during class on TBD
- Exam 2 (30%)
  - Comprehensive
  - Date and time: TBD
- Missed exams will be granted zero points. Illness and severe circumstances may be accommodated only with solid evidence. Email instructor before exam and provide solid evidence.

Grading: Attendance

Attendance (10%)

- Each student is allowed two absences from class for the entire semester without direct penalty to his or her grade (this does not include penalties that may result from missing in-class exams)
- For each absence over two, a student's final grade will be reduced by 1 percentage point, but no more than 5% accumulated

*Challenging Projects

- Individual work or working as a group with up to 2 members
- Coding problems
- Release time TBD

Academic Integrity

- Academic Integrity is defined in the UNT Policy on Student Standards for Academic Integrity. Any suspected case of Academic Dishonesty will be handled in accordance with the University Policy and procedures. Possible academic penalties range from a verbal or written admonition to a grade of F in the course. Further sanctions may apply to incidents involving major violations. You will find the policy and procedures at: http://vpaa.unt.edu/academic-integrity.htm.

Academic Integrity

- Each topic discussed in class will have associated assignment.
- Students may discuss assignment problems and approaches with each other but must write their solutions individually.
- Students may not copy assignment from any source (e.g., other students, the Internet).
- No collaboration is allowed in quizzes and exams.

Questions?

Course Overview

- This course provides a fun, fast-paced, modern introduction to algorithms and data structures.

- We will…
  - Practice using different algorithm design techniques.
  - Improve our analytical skills while learning mathematical tools for algorithm analysis.
  - Learn fundamental algorithms and data structures.
  - Learn the inner workings of some of the algorithms that currently have the most impact in practice.
  - Improve our coding skills.

Why Learn Algorithms?

- Algorithms are the heart and sole of computing!
- Algorithmic computing now plays a key role in nearly everything – proficiency opens many doors.

- Algorithmic prowess differentiates a true "computer scientist" from a run-of-the-mill "programmer" and provides the foundation for a long-term career in computing that can thrive as technology changes.
- Programming ≠ Algorithmic Problem Solving

A Good Algorithm (or Data Structure)…

- Always terminates and produces correct output.
  - A "close enough" answer is sometimes fine.
  - Some types of randomized algorithms can fail, but only with miniscule probability.
- Makes efficient use of computational resources.
  - Minimizes running time, memory usage, processors, bandwidth, power consumed, heat produced.
- Is simple to describe, understand, analyze, implement, and debug.

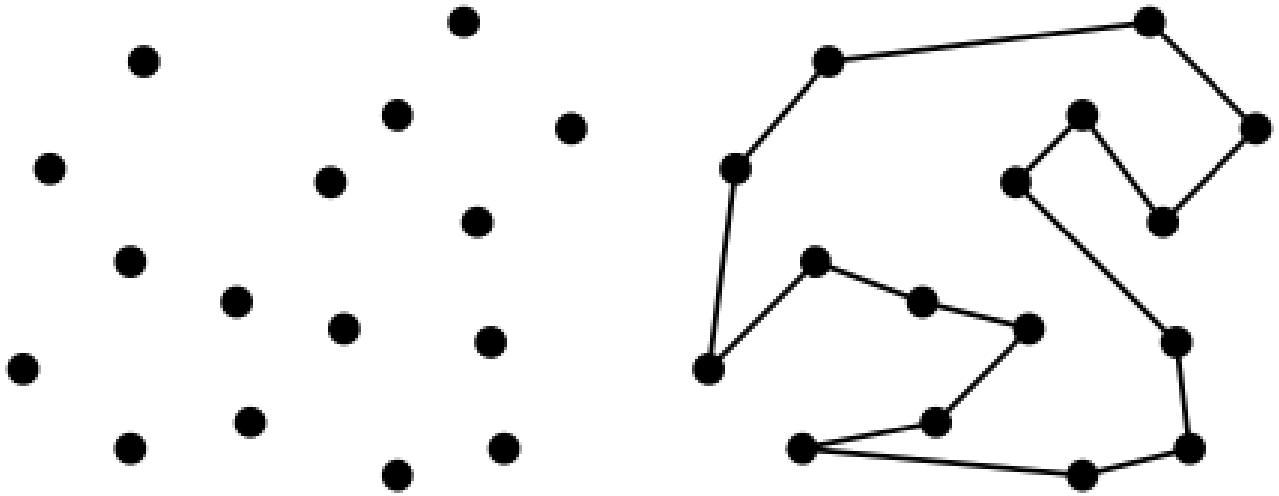
Robot Tour Optimization

Suppose you have a robot arm equipped with a tool, say a soldering iron. To enable the robot arm to do a soldering job, we must construct an ordering of the contact points, so the robot visits (and solders) the points in order.

We seek the order which minimizes the testing time (i.e., travel distance) it takes to assemble the circuit board.

Find the Shortest Robot Tour

- You are given the job to program the robot arm. Give me an algorithm to find the best tour!

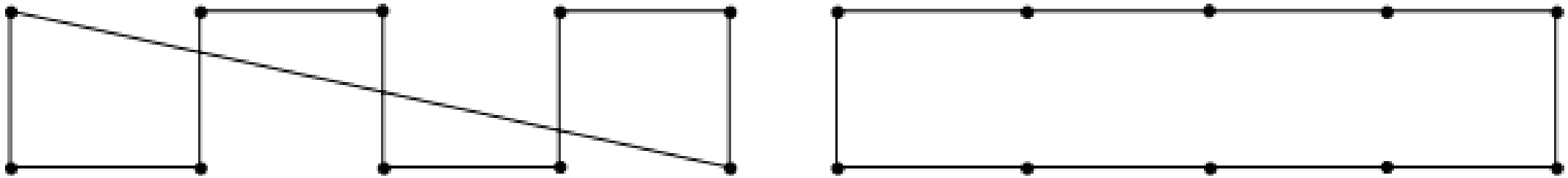


Exhaustive Search

- We could try all possible orderings of the points, then select the one which minimizes the total length.
- Since all possible orderings are considered, we are guaranteed to end up with the shortest possible tour.
- Because it tries all $n!$ permutations, it is much too slow to use when there are more than 10-20 points.

Closest Pair Tour

• One idea is to repeatedly connect the closest pair of points whose connection will not cause a cycle or a three-way branch, until all points are in one tour.



No efficient, correct algorithm exists for the traveling salesman problem, as we will see later.

Expressing Algorithms

We need some way to express the sequence of steps comprising an algorithm.

In order of increasing precision, we have English, pseudocode, and real programming languages. Unfortunately, ease of expression moves in the reverse order.

I prefer to describe the ideas of an algorithm in English, moving to pseudocode to clarify sufficiently tricky details of the algorithm.

Example: Searching a Sorted Array

- **Linear search: runs in N "steps" in the worst case.**

```
for (i=0; i < N; i++)
      if (target == A[i]) {found it!}
```

- **Binary search: ≤ $\log_2 N$ "steps" in worst case.**

```
low = 0; high = N-1;
while (low <= high) {
      mid = (low + high) / 2;
      if (target == A[mid]) { found it! }
      if (target > A[mid]) low = mid+1;
      else high = mid-1;
}
```