Data Structures
Computer Science 271

Instructor:	Erik Krohn
E-mail:	krohne@uwosh.edu
Text Message:	920-644-3745
Class Time:	Tuesday, Wednesday & Thursday: 10:20am - 11:20am
Classroom:	Halsey 237
Lab Time:	Monday: 10:20am - 11:20am
Lab Location:	Halsey 101C
Office Location:	Halsey 216
Office Hours:	Monday: 12:30pm - 1:30pm
	Tuesday: 12:30pm - 1:30pm
	Wednesday: 9:20am - 10:20am
	Thursday: 9:20am - 10:20am
Prerequisites:	CS 262 with a grade of C or better.
Course Website:	`http://www.uwosh.edu/d2l`
Recommended Textbook:	Data Structures and Algorithm Analysis in Java
	Mark Allen Weiss, ISBN 0132576279

Course Information
A course surveying the fundamental methods of representing data and the algorithms that implement and use those data representation techniques. Data structures and algorithms include; linked lists, stacks, queues, trees, heaps, priority queues, hashing, searching, sorting, data compression, graphs, recursion. Analysis topics include: elementary big-O analysis, empirical measurements of performance, time/space tradeoffs, and identifying differences among best, average, and worst case behaviors.

Course Website
You should check d2l on a regular basis - it will contain lecture notes, handouts, assignments, announcements, and grades. Ill do my best to let you know when something new and important comes up, but it is your responsibility to check the web site frequently for information that you might not get otherwise.

Mini Assignments
You will have daily mini assignments. Mini assignments are generally short and should take less than fifteen minutes to complete. You will be assigned a mini assignment every lecture to ensure you are staying current with the material. I will drop your 2 lowest mini assignments. Not all mini assignments will be graded. **No late mini assignments will be accepted.**

Projects & Labs

Most projects and labs will consist of short programming projects. One of your goals (during this class and beyond, in Java or any programming language) should be to write understandable, readable code. You should be making every effort to comment anything that might be confusing to a reader unfamiliar with your program, to name variables intelligently, to use indentation that reflects the code's organization, and so on. All of this will be taken into account during grading: poorly organized or written code may have a negative impact on your grade, even if the resulting program works fine.

One of the goals of this class is to teach you to write functioning programs in Java - thus, your code must compile in order for you to receive any credit. Code that does not compile will not be tested and your score will be a 0. Keep this in mind when writing programs: write your code in small pieces, making sure each piece works before moving on to the next one. It is much better to turn in a project that is not finished but has many working pieces than to turn in one that doesn't work at all, even though most of the code is written.

All assignments must be submitted electronically via d2l. It is your responsibility to ensure that your submission was submitted correctly. You must double check to ensure your program was uploaded correctly. Each assignment must be submitted by 11:00pm on the night of the due date. **There are no late submissions**.

Project Resubmissions

Each project must pass **all** of my test cases. A program that fails just one test case will receive at most 70%. That said, you will be allowed to resubmit assignments *once* for regrading within 10 days of the initial program being graded. Complete test cases will be posted after the initial grading so you can go back and test your code to see what went wrong. Regraded programs can receive a maximum score of 85% if the modifications made were very minor. Major modifications or additions will receive very little, if any, additional points.

Exams

Exam material will come from the lecture notes, labs, mini assignments, book and assignments. There will be more information about each exam as it approaches. The *tentative* exam dates are listed below. All exams will be taken during the regular class period. These may change, so as the date approaches make sure you've got the most recent information.

- **Exam One** - Wednesday, March 2^{nd}, 2016
- **Exam Two** - Wednesday, April 6^{th}, 2016
- **Exam Three** - Wednesday, May 11^{th}, 2016

If you are unable to take a scheduled exam, it may be possible to take a make-up exam provided that you do both of the following, which are then subject to my approval:

1. Make arrangements prior to the scheduled exam. For last minute emergencies, telephone me at 424-7080 or leave a message at the computer science office, 424-2068 or send me a text message No after-the-fact notifications will be accepted.

2. Have a written medical excuse signed by the attending physician OR have a note of justification from the Dean of Students Office.

If allowed, only one make-up exam will be given. It will be a comprehensive exam given at an arranged time during the last week of the semester.

Grading

Course grades will be based on assignments, mini assignments and exams. Your final grade will be computed with the following percentages:

- 40% - projects
- 15% - mini assignments/labs
- 45% - exams

If you believe anything was graded incorrectly or unfairly and would like to have it regraded, you must let me know about it within *one week* of having the item graded. I will regrade the entire assignment or exam and you may gain or lose points.
Grading will be on a plus/minus system. Grading may be done on a curve depending on the overall performance of the class. If no curve is used, your grade will be computed based on the following:

Percentage	Grade	Percentage	Grade
≥ 92	A	72 - 78	C
90 - 92	A-	70 - 72	C-
88 - 90	B+	68 - 70	D+
82 - 88	B	62 - 68	D
80 - 82	B-	60 - 62	D-
78 - 80	C+	< 60	F

Learning Outcomes

- Given a non-recursive algorithm, the student will be able to examine its loop structures and infer its asymptotic runtime using big-O notation.
- Given a recursive algorithm, the student will be able to examine its recursive structure, determine the corresponding recurrence relation (from a small collection of commonly occurring recurrence relations), and use the recurrence relation in determining the asymptotic runtime of the algorithm using big-O notation.
- Given the description of a computational problem requiring a mixture of search, insertion, and/or deletion operations on collections of data, the student will be able to compare the relative advantages of using arrays and linked lists in solving the problem efficiently.

- Given a classical computational problem (e.g., infix-to-postfix conversion, postfix-expression evaluation, path planning, minimum-spanning tree computation), the student will be able to trace a solution to the problem using appropriate data structures (e.g., stacks, queues, binary trees, binary search trees, red-black trees, graphs) and to predict the asymptotic runtime of the solution based on the selected data structures.

- Given a collection of unordered data, the student will be able to trace the execution of an advanced sorting algorithm (such as quick sort and heap sort) on this data set.

- Given a set of data keys, the student will be able to trace through a sequence of key insertions, searches and deletions on a balanced tree structure. The student will also be able to discuss the relationship between the number of keys and the execution time of these operations.

- Given a set of data keys, a hash function, a table size, and a collision handling strategy, the student will be able to trace through a sequence of key insertions and searches, and to discuss how varying the table size, hash function or collision-handling would affect the execution time of these operations.

- Given a graph data structure, the student will be able to implement it using either adjacency lists or an adjacency matrix, to traverse it using either a depth-first or breadth-first strategy, to identify its structural properties (whether it is directed, cyclic, connected, complete), and to trace the execution of one or more classical graph algorithms (e.g., Dijkstras shortest path, topological sort or minimum-spanning tree computation).

- Given a problem requiring the efficient use of a variety of data structures, the student will be able to apply object-oriented design principles in implementing and testing a solution to that problem in an appropriate object-oriented language.

Academic Dishonesty

Academic dishonesty of any kind will not be tolerated. All assignments, mini-assignments and exams are to be completed individually unless otherwise specified. While high-level discussion of ideas and problems with fellow students is encouraged, all work must be done individually. In certain circumstances, code fragments from the instructor may be provided to eliminate tedious coding or to provide a common framework for all students. **All other code must be original**. Online resources may be used to help you understand the material, but you may not copy online code nor can you "borrow" code from other students, past or present. If you use a book, website or any reference to help you solve a problem, you must cite the reference in your assignment.

Any suspected academic dishonesty will be dealt with on a case-by-case basis. Any clarification of what does or does not constitute academic dishonesty must take place **before** you turn in questionable work. For clarification on what constitutes academic dishonesty, contact me or consult the printed policy in the UWO Student Discipline Code, Chapter UWS 14.