Data Structures
(Comp Sci 271, Section 001)

Instructor: Erik Krohn
E-mail: krohne@uwosh.edu
Text Message Only: 920-644-3745
Class Information: Monday, Wednesday, Friday: Halsey 212, 3:00pm – 4:00pm
Monday: Halsey 101C, 4:00pm – 5:00pm
Office Location: Halsey 216
Office Hours: Monday, Wednesday, Friday: 2:00pm – 3:00pm
Thursday: 1:00pm – 3:00pm
Tutor Hours: Sunday - Thursday: Halsey 101C, 6:00pm – 7:00pm
Prerequisites: CompSci-262 with a grade of C or better.
Course Website: http://www.uwosh.edu/d2l/
Recommended Textbook: *Data Structures and Algorithm Analysis in Java* by 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 trade-offs, 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. I'll 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.

Grading
Course grades will be based on 5-6 programming projects, ~10 labs, quizzes and three exams. Your final grade will be computed as follows:

40% - Programming Projects
15% - Quizzes/Labs
45% - Exams

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
>91	A	71 - 77	C
89 - 91	A-	69 - 71	C-
87 - 89	B+	67 - 69	D+
81 - 87	B	61 - 67	D
79 - 81	B-	55 - 61	D-
77 - 79	C+	<55	F

Exams
Exam material will come from the lecture notes, quizzes, book, programming assignments and labs. 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.

Midterm: Wednesday and Friday, October 23rd and 25th, 2013
Final: Wednesday and Friday, December 11th and 13th, 2013

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. **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.

Quizzes
You will have quizzes throughout the semester. Quizzes are generally short and should only take a few minutes to complete. You will be given a quiz every 3-5 class periods to ensure you are staying current with the material. Your lowest quiz will be dropped. There are no make-up quizzes.

Assignments & Labs
Most assignments 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 and run correctly in order for you to receive full credit. **Code that does not compile will receive <u>at most</u> 50% credit, and often substantially less.** 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.

Each project *must* include a README text file with any information I need to know regarding this project. The information in this file will be taken into account during grading, so it will be beneficial for you to make sure that everything I need to know about your work is written in this file.

All assignments must be submitted electronically via d2l. It is your responsibility to ensure that your submission was submitted correctly. Each assignment must be submitted by 11:00 PM on the night of the due date. You are allowed 1 late submission up to 3 days late. **Subsequent late submissions are penalized at 15% per day.**

If you believe an assignment or exam was graded incorrectly or unfairly and would like to have it re-graded, please let me know about it *within one week* of having the assignment or exam graded. I will re-grade the entire assignment and you may gain or lose points.

Academic Dishonesty
Academic dishonesty of any kind will not be tolerated. All assignments, labs, quizzes and exams are to be completed individually. While discussion of ideas and problems with fellow students is encouraged, all projects 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, as is "borrowing" code from other students, past or present.

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.

Learning Outcomes

- Given a non-recursive algorithm, the student will be able to examine its loop structures, infer its asymptotic runtime, and express its efficiency using big-O notation.
- Given a recursive algorithm, the student will be able to examine its recursive structure, determine and solve the corresponding recurrence relation, and infer 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, vectors, and linked lists in solving the problem efficiently.
- Given a classical computational problem (e.g., infix-to-postfix conversion, postfix-expression evaluation, Huffman data compression, 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., Dijkstra's, 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.
- Given the contents of an uncompressed data stream, the student will be able to trace the Huffman, LZ77, LZ78, and LZW algorithms to produce the corresponding compressed contents.
- Given the contents of a data stream compressed with the Huffman, LZ77, LZ78, and LZW algorithms, the student will be able to decompress the stream manually and recover its original contents.
- Given the contents of a data stream to be compressed, the student will be able to compare and contrast the relative merits of the Huffman, LZ77, LZ78, and LZW algorithms from both a time and space perspective.