CS 171 Fall 2020 Syllabus

Section 1 (Oser) - M/W 2:40pm - 3:55pm

Instructor: Paul Oser  Email: poser3@emory.edu
Zoom Classroom: 667-472-091
Zoom Office: 652-382-252 (hours posted in Canvas)

Overview

This course is a continuation of CS170. Emphasis is on Java, the use and implementation of abstract data types for collections, the algorithms related to these, introductory algorithm analysis, and object oriented design.

Students will be given many opportunities to write programs to demonstrate their mastery of the algorithms and data structures covered in this course. Problem solving and real-world applications will play an important role, providing a driving motivation for developing and/or selecting appropriate algorithms or data structures to accomplish the associated goals as efficiently as possible.

Goals for Student Learning: Students at the conclusion of this course should be able to...

- Effectively use the following features of the Java language: inner classes, interfaces, exceptions, and generic types
- Implement and use various abstract data structures associated with collections, including (but not limited to) stacks, queues, linked lists, binary search trees, 2-3 tree, red-black trees, heaps, hash tables
- Analyze the efficiency of algorithms with regard to how their running times increase as the related problem size increases, in the best, average, and worst-case scenarios
- Implement a variety of sorting algorithms, including selection sort, insertion sort, merge sort, quicksort, and heap sort, understanding the advantages and disadvantages of each
- Implement a backtracking search algorithm
- Implement and use graphs, along with their related algorithms, to accomplish various tasks

Materials for the Course

All necessary material for the course can be found on the Canvas site for CS 171.

An excellent book that mirrors much (but not all) of the content we cover is *Algorithms, 4th Edition* by Robert Sedgewick and Kevin Wayne. Students intending continued study of computer science may wish to add this wonderful book to their library. For students not wanting to purchase this book, but who would still like more material to reference than just that provided on Canvas should know that large chunks of Sedgewick’s book (but not everything) are also freely available at https://algs4.cs.princeton.edu/home/

Students will need access to a computer with a Java-based Integrated Development Environment (IDE). Eclipse is recommended towards this end, and is free.

Pre-requisites

Successful completion of CS 170 or equivalent preparation.
Expected Workload

Students’ weekly commitment will be split across three components: asynchronous material online, synchronous class meetings, and individual homework and practice. All three components are vital to success with the course.

- **Asynchronous Material Online**: The bulk of the material for this course will be delivered via recorded lectures, online notes, and examples linked to from Canvas. These materials will be organized by day in the Canvas learning management system. Canvas will be used to organize most aspects of this course and to communicate any office hours, announcements, assignments, and updates. Consequently, it is important that students check Canvas daily.

- **Synchronous Class Meetings**: The instructor will be available via Zoom during the regularly scheduled class time. Most of this time will be spent providing additional examples beyond those already in the asynchronous video lectures for those that wish it, helping students with questions about the programming assignments, "live coding" some of the data structures we cover to reinforce what is covered in the notes online, and addressing questions from students or other important details about the course material that need further explanation. Importantly – roughly once every two weeks – this time will also be used for students to take a quiz on the material encountered in class so far. For all synchronous sessions attended, students are expected to keep their cameras on throughout the session. Also, during these sessions students should use their electronic devices in a way that is conducive to the task at hand: learning. To that end, students should keep cell phones silenced and refrain from responding to texts or other electronic messages and similar distracting uses of their computers during class meetings.

- **Individual Homework and Practice**: Programming is a skill, and its improvement requires regular meaningful practice. The same can be said when it comes to understanding the algorithms that drive the data structures explored in this course. Practice with both will be provided to you in the form of collections of problems linked to from Canvas. That said, how much you use these problems to practice the material and study for the course will largely be left to your own discretion. Students that meet with success in the course, however, often spend a considerable amount of time each day working through these problems.

**Quizzes**

Instead of assessing students with a small number of high-stakes tests, frequent quizzes will be used instead. Given roughly once every two weeks during our synchronous sessions, these quizzes will be the means used in this class to test student’s conceptual understanding of the basic algorithms of the course. Importantly, for these quizzes, students will need a means to scan their written work and upload it to Canvas. Doing well on these quizzes will strongly correlate to having read and understood the notes and examples online and having well-digested the content of the video lectures posted in Canvas.

**Assignments**

Assignments will often (but not exclusively) involve designing, coding, testing, and debugging non-trivial programs based on some written specifications. Some of these will involve completing some given partially-fleshed-out code, others will be left to the student to craft completely on their own. The primary intent of these assignments is to expose students to how the conceptual ideas encountered in class play out in practice when connected to specific applications – although some will also involve exposing students to variations on approaches to classic problems.

The time given to complete programming assignments will minimally be one week, although it may often be longer. Completed programming assignments should be submitted to Canvas, as appropriate. Students are required to regularly backup their Java workspace and programs written to protect against the loss and/or catastrophic failure of their computer.

Should group programming assignments be assigned (most often to pairs of students), students should avoid the temptation to "divide and conquer” the assignment in question. Students giving into this temptation will only
end up “doing” half of the work, and consequently, only really learning half of the material (which invariably gets revealed on the quizzes or final exam).

Allowing students to work with each other in pairs or small teams is intended to create conversations between students as they both develop each program in their own way – conversations that will reveal when one approach is better than another, and should be adopted by both.

Typically, all of the individual programming assignments will be equally weighted. However, the instructor reserves the right to weigh some of these grades more heavily, so that if a particularly challenging program is assigned, students can be rewarded for the extra effort involved.

**How to Approach Programming Assignments in This Class**

Programming is a skill best learned by “doing” – so students should plan on spending a considerable amount of time in front of a computer outside of class this semester.

Be aware that a certain amount of struggle in trying to figure out what’s going on in a program, or why a particular error results is healthy – this is largely how programmers learn their craft. The programming assignments are designed to do just this – to put students into situations where they may need to experiment with several ways to proceed before settling on the one that seems to work best. In this light, students should avoid getting discouraged when things don’t go smoothly the first time – or the first several times – they attempt a programming assignment. That said, when students are completely flummoxed – they are both encouraged and expected to talk to and get help from their instructor.

Java code can be compiled and run regardless of whether or not it follows the good naming conventions, or is properly indented, or has appropriate explanatory comments laced throughout the code, etc. Such programs may even produce correct output. That said – to be successful, students should care a great deal about these things. Towards this end, all students are expected to follow the standard and/or consistent conventions of “good style” when programming, and points can and will be deducted when deviations from good style interfere with the grading of a student’s program.

**Final Exam**

A cumulative, synchronous final exam will be given after Thanksgiving, no earlier than Dec 3rd. The precise date and time will be announced later in the course. The exam will be in a format similar to that of the quizzes in the course – just longer and more comprehensive.

**Grading:** Final course grades will be determined as follows:

<table>
<thead>
<tr>
<th>Component</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quizzes</td>
<td>40%</td>
</tr>
<tr>
<td>Assignments</td>
<td>40%</td>
</tr>
<tr>
<td>Final Exam</td>
<td>20%</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>100%</strong></td>
</tr>
</tbody>
</table>

In general, letter grades will be determined as follows, based on points each student earns:

- A: at least 90%; B: 80-89%; C: 70-79%; D: 60-69% points; F: less than 60%.

Grades of A-, B+, B-, C+, C-, D+ may be assigned for sums of points near these cut-off totals.

**Late Policy**

Students are expected to be present for all scheduled quizzes. Any conflicts should be brought to the instructor’s attention as soon as possible. If a legitimate reason exists for missing a quiz – as determined by the instructor –
then the quiz must be taken prior to the regularly scheduled date.

In the unusual circumstance where taking the quiz on time or early is not possible, the instructor may (at his discretion) offer a makeup quiz. Students should be aware that any make-up quizzes given will be designed to be more difficult to offset the additional time given for study. Alternatively, in some cases, the instructor may offer to let the final exam score (or some subset of this score) replace a missing quiz score. Students must provide written documentation in advance of any special accommodations required for testing. This includes additional time or other needs. The final exam cannot be rescheduled.

In general, late programming assignments will not be accepted; this policy will be waived only in an "emergency" situation at the instructor’s discretion, and only after any documentation the instructor requires has been provided.

**Honor Code (see http://oxford.emory.edu/catalog/regulations/honor-code.html)**

The Oxford College Honor Code applies to all work done in this class.

Students may not give, access, or receive any information not expressly permitted by the instructor on quizzes or the final exam. Collaboration between students on quizzes and the final exam is strictly prohibited.

Unless otherwise indicated by the instructor, individual programming assignments must be completed without consulting code written by other students or resources outside of those provided.* That said, students are welcome to discuss general principles and concepts about the assignments with each other and with the instructor.

For any group programming assignments where two students work together, a comment similar to the following (with the names changed appropriately, but everything else absolutely identical) must be included as the first line of every java file submitted for the project:

//SUBMITTED BY: Mike Beck and Jon Fowler

Similarly, if three students work together, the comment should take the following form (with the three names changed appropriately, but everything else absolutely identical):

//SUBMITTED BY: Mike Beck and Jon Fowler and Velina Veleva

Collaboration on group projects between students that are not in the same group is not allowed.*

* Importantly – for both individual and group programming assignments – if the instructor starts to get the impression that inappropriate collaboration is occurring (which nullifies the capability of these assignments to provide a genuine measure of students’ ability to write code), the instructor will be forced to ask more questions on the quizzes that will require students to write code “on the spot”. As such code-writing is done with pencil or pen, and without the benefits of code-completion, alerts to syntax errors, or a debugger - like those the Eclipse IDE provides - students will undoubtedly find such quizzes significantly more difficult.

**Special Accommodations**

The Office of Accessibility Services (OAS) works with students who have disabilities to provide reasonable accommodations. In order to receive consideration for reasonable accommodations, please contact the OAS and complete the registration process. Faculty may not legally provide you with accommodations until an accommodation letter has been processed and discussed with them; accommodations do not start until this point and are not retroactive. Students registered with OAS who receive a letter outlined specific academic accommodations are thus strongly encouraged to immediately coordinate a meeting with their professors to discuss a protocol to implement accommodations that will (or may) be needed over the course of the semester. This meeting should occur as early in the term as possible. Contact Megan Bohinc in OAS for more information at (770) 784-4690 or oas_oxford@emory.edu
Tentative Order of Topics:

1. Analysis of Algorithms, Doubling Time, Useful Approximations
2. Abstract Data Types, Stacks, Postfix Expressions, Evaluating Infix Expressions, Delimeter Matching
3. The Shunting Yard Algorithm
4. The N-Queens Problem and Backtracking
5. Generics, Type Parameters, Autoboxing, and Auto-unboxing
6. Iterating over Collections, Resizing Arrays, Garbage Collection
7. Implementing Stacks, Queues, and Bags
8. Linked Lists, Common Linked List Operations
9. Searching and Sorting - First Thoughts, Linear Search, Binary Search, Bubble Sort, Selection Sort, Insertion Sort
10. Merge Sort, Merge Sort Analysis
11. Quick Sort, Quicksort Analysis
12. An Alternate Partitioning Technique, Another Way to Analyze the Average Quicksort Case, Better Handling of Duplicates (3-way Quicksort)
14. Symbol Tables, Binary Search Trees, Searching and Inserting in a BST, A Connection Between Binary Search Trees and QuickSort
15. Operations on Binary Trees Involving Key Order
16. Tree Traversals, Supporting Rank Operations in BST, Hibbard Deletion
17. Priority Queues and Heaps, Sinking and Swimming in a Heap, Immutability, Heapsort, Heapsort Analysis
18. 2-3 Trees, Red-black trees, Representing Red Links, Searching and Inserting in a Red-black Tree
19. Hash Tables, Using the hashCode() method, Generating Hash Codes, Resolving Collisions
20. Graphs, Representing Graphs
21. Directed and Edge-Weighted Graphs
22. Traversing Graphs (Depth-First Searches, Breadth-First Searches)
23. Identifying Connected Components, Identifying Cycles, Identifying Bipartite Graphs
24. Dijkstra’s Shortest Path Algorithm
25. Prim’s Algorithm
26. Topological Sort