Sorting out Sorting

Colin Fitzpatrick

Instructor's experience

I have TA-ed a total of eight undergraduate courses across Brown University, UW-Madison, and Northwestern University on topics of music history, music theory, world cultures, and communication and technology. In some, I ran separate weekly discussion sessions for the students, creating and facilitating exercises based on both the current lectures and the ways in which students were responding to material. Specific to this proposal, in the 2014-15 academic year I was partnered with a middle school seventh and eighth grade class room and taught them intro to computer programming, which included lessons on algorithms, as well as last year when I taught this workshop to the first Computing Everywhere group.

Availability

Both Winter and Spring quarter, with preference for earlier in the quarter

Purpose

The concept of algorithms and their design can be difficult to grasp in the abstract. It is one thing to understand that algorithms affect the ways in which we interact with content online and that teams of engineers and designers make decisions about their design; it is quite another to "go behind the curtain" to see what a decision looks like, how it effects the algorithm, and how to decide among a set of algorithms for a particular task at hand. The purpose of this lesson is to first build out students' understanding of how algorithms are implemented through activities and pseudocode, then apply those ideas in an exploration of commonly known sorting algorithms.

Overview

This lesson unfolds in four parts:

(1) discussion around algorithms' role and function (15 minutes)
(2) sorting activities (45 minutes)
(3) pseudocode and implementation (40 mins)
(4) evaluation of algorithms (20 mins)

We begin by situating ourselves through a discussion of algorithms' role and function in our daily lives, from the complex (e.g. Newsfeed) to the seemingly mundane (sorting an Excel spreadsheet column). Arriving at a working definition of algorithms, we put it into practice through a set of activities around sorting (e.g. line up by height order, line up by age, line up alphabetically by major). Each time students complete a sort, they will formalize their process by marking down the steps they took; for each successive activity, they will be prompted to determine a new process for sorting. For part three, we will review the pseudocode recorded in part two and together implement them in Python, connecting each step in their process to lines of code (anticipating common sorting algorithms such as Bubble, Insertion, Selection, Merge, etc). Finally, in part 4, we will discuss edge cases and see if/when/where and how certain algorithms will break or outperform others.

Student Outcomes

Students will be able to:

1. Identify problem spaces for algorithmic intervention
2. Generate pseudocode for algorithm design
3. Make implementation decisions given the problem space
4. Evaluate algorithms

Time

one 2-hour session

Level

Undergraduate students with little to no experience in computer science

Materials and Tools

Text Editor
Python
Terminal
Projector
Datasets of sorted and unsorted data to run through implementations
Implementations of common sorting algorithms

Preparation

It is beneficial for the instructor to be proficient with common sorting algorithms and their comparative advantages and disadvantages. Preparing standard implementations of common sorting algorithms would also be helpful.

Prerequisites

There are no prerequisites for this lesson. Familiarity with algorithmic concepts will be helpful, but not necessary.

Background

No formal or informal knowledge about algorithms is necessary to be successful in this lesson. This lesson instead is designed to tease out the ways we already implement algorithmic thinking without necessarily being aware of it, and how to translate that thinking into how programmers would approach it. If you've ever put a set of items into order (a deck of cards, a list of song titles, the colors of the rainbow, etc), you will be able to participate fully.

Teaching Notes

The lesson begins with an open discussion of algorithms students encounter in their daily lives. There are several points of departure for prompting this discussion; perhaps through playing music from a playlist that's also projected as students are coming in (and then asking about how those songs are ordered and other ways of ordering them). This should pivot from a conversation about a real world thing, to its data structure, to ways of sorting that data structure. From there, the instructor builds and guides the conversation (when and where necessary) to generate a working definition of what an algorithm is and how simple or complex they may be. Emphasizing the utility of taking simple cases for exploration, the instructor pivots to Part 2 and leads the class through a set of exercises for sorting.

It is most critical in Part 2 that the students isolate and articulate all of the steps in their process, particularly their assumptions around the data they are working with. For example, if prompted to organize by height, it might be that the shortest person self-selects him/herself and initiates a new "line" by moving to the front of the classroom and others follow suit. It is the instructors' job to help facilitate breaking down this process into implementable chunks of "code". In this case, given an unsorted list of data, choose one piece of data randomly and initiate a new list; continue to grab pieces from the unsorted data and compare its value to that of sorted data in the new list, placing it in the appropriate place (effectively, an insertion sort with two lists).

In part three, the instructor codes versions of the students' algorithms in a text editor so that students may see what code looks like and test their implementation on various sets of data (troubleshooting any problems). It is essential in this step that the instructor include edge cases, such as lists that are already sorted, sorted in reverse order, or contain more than one of the same item in order to get students to confront common problems in algorithm design and the decision process for dealing with them.

Finally, in part four, the instructor will query the students as to what makes a good algorithm. Responses should include, but not be limited to, its implementation, performance, and the appropriateness of its application given the context. This discussion will be rooted in the examples previously generated where possible (for example, assuming the students developed an insertion sort algorithm, asking them how they would implement a version that would only use one list instead of two, and what that means in terms of data structures and storage). Issues around data structure and its limitations may also come up.

Assessment

Assessment for this activity is an emergent and responsive process. In addition to a rudimentary grasp of the student outcomes listed above, this lesson aims to impart an experience with formal properties of algorithmic design and implementation. The sorting activities in part two are designed in such a way to be simple enough to get through them, but complex enough to require precise steps for successful completion. Assessment is in part captured through the trial and error that happens during these activities, how the students iterate on their algorithm design, and how they formalize these steps into pseudocode on paper. For part three, we will run through implementations of the algorithms with sets of sorted and unsorted data. Again, how students respond to success or failure, and, if present, the effects of impromptu tinkering of their algorithm, will indicate understanding for assessment. Finally, for part four, the instructor can assess how robustly they capture determinants of algorithm performance (how well does this algorithm perform?) and application (is this algorithm right for this job?).

Additional Information

Samples of Python implementations of common sorting algorithms are included with the proposal. No additional data, technology, or resources are needed/anticipated.

A specific implementation of this lesson plan can be found at: http://fitzcn.net/sorting/