CSCE 120: Learning To Code

Organizing Data I
Hacktivity 12.1

Introduction

Prior to engaging in this hacktivity, you should have completed all of the pre-class activities as outlined in this module. At the start of class, you will be randomly assigned a partner to work with on the entirety of this hacktivity as a *peer programming* activity. Your instructor will inform you of your partner for this class.

One of you will be the driver and the other will take on the navigator role. Recall that a driver is in charge of the keyboard and computer while the navigator is in charge of the handout and directing the activity. However, you are *both* responsible for contributing and discussing solutions. If you were a driver/navigator in the prior activity, switch roles for this activity.

1 Knowledge Check

1. Explain the advantages and disadvantages of linear search vs binary search.
2. Suppose you have an array of size $n = 10^{12}$ (1 trillion elements). About how many operations would linear search take? About how many operations would binary search take?
3. Suppose we want to sort the array with $n = 10^{12}$ elements. Using selection sort, about how many operations would be required?
4. Consider the following code.

   ```
   1 var arr = [10, 5, 200, 23, 8];
   2 arr.sort();
   ```

   What will the configuration of `arr` be after this code executes? Discuss why with your partner.

5. Write a piece of code that correctly sorts the array in the previous problem so that the elements are sorted in *descending order* (that is, the result should be `[ 200, 23, 10, 8, 5 ]`).

2 Benchmarking Sorting Algorithms

Download the code we've provided from GitHub using the URL, https://github.com/cbourke/SortingProject. Open the project in Light Table.

2.1 Understanding Sorting Algorithms

To begin, let's get a better understanding of some sorting algorithms. Open your browser to http://sorting-algorithms.com. We will examine several sorting algorithms and how they behave on *random* data.

1. With your partner, read the page on Random Initial Order: http://sorting-algorithms.com/random-initial-order
2. Run the simulation (click the green "Restart All" button) and observe the results. In particular, pay attention to Bubble Sort, Insertion Sort, Selection Sort, Quick Sort. Order these four algorithms from fastest to slowest based on your observations.

2.2 Benchmarking

We will now benchmark these four algorithms as well as JavaScript's built-in `sort()` method. A *benchmark* is a comparison of software or algorithms in order to gauge their empirical performance against each other. Often a benchmark tests many different types of scenarios and data, but we'll only focus on *randomly* generated data as in the demonstration earlier.

Open and evaluate the HTML and JavaScript files in the `Benchmark` folder. This page provides several implementations of sorting algorithms as well as the framework to randomly generate arrays of size n filled with random data. It then runs the specified sorting algorithm on the data and reports the runtime.

1. Run the experiment on each algorithm for each of the array sizes in Table 1 and fill in the results (you may use a separate sheet of paper if you do not have a printout).
2. Without running each algorithm again, predict how much time it would take for each one to run on n =200,000 elements.
3. Once again, order these algorithms in order of performance. Does this match your earlier ordering? How does JavaScript's `sort()` method compare to these?

Algorithm	Size, n				
	1,000	5,000	10,000	50,000	100,000
Bubble Sort					
Insertion Sort					
Selection Sort					
Quick Sort					
`sort()`					

Table 1: Empirical Results measured in seconds.

3 Proper Sorting

Hopefully you concluded that using JavaScript's built-in `sort()` is the best choice as far as performance goes. In fact, it is generally best practice to use the searching and sorting algorithms provided by the programming language or standard library rather than writing your own. We'll now get some practice using JavaScript's `sort()` method.

We have provided several files involving student enrollment data.

1. Open the files in the `Roster` folder and evaluate the HTML page. It will load student data from the JSON file and display them in the order in which they are stored in the JSON file. The user is allowed to select a different ordering which will invoke the `sort()` function, passing one of 4 different callbacks.
2. Only the first sorting option has been implemented. Implement the three comparator callbacks, `byCourse()`, `byGPADesc()`, `byGPAAsc()`, and `byYear()`. Take particular care with the last one: what order would a user expect? (Hint: use the array we've defined, `years` along with the `indexOf()` method to convert the string representations into ordered numbers).
3. Try sorting by the same criteria multiple times. What behavior do you observe? Discuss possible reasons for this behavior and think of some possible solutions.