(http://csmatters.org) 5 - 4

0b101 - 0b100

Comparing Algorithms

Unit 5. Data Manipulation

Revision Date: Feb 06, 2020
Duration: 2 50-minute sessions

Lesson Summary

Pre-lesson Preparation: You should familiarize yourself with www.sorting-algorithms.com (http://www.sorting-algorithms.com) paying particular attention to the variety of algorithms and settings along the top of the page. For session 2, you should have the timedsorts.py code and data files (in the lesson folder) readily available for your students.

Summary

In this two-session lesson, students will explore algorithmic efficiency. They will understand the idea through discussion, manual analysis of simple algorithms, and data collection for implemented algorithms.

Outcomes

Students will be able to:

- identify algorithms that have different efficiencies in their problem solving approach.
- explain the metrics used to describe efficiency.
- perform an empirical analysis of sorting algorithms by running the algorithms on different inputs.

Overview

Session 1:

1. Getting Started (5 min)
2. Guided Activity (40 min)
    1. Good Algorithms and Better Algorithms (5 min)
    2. Algorithmic Efficiency (10 min)
    3. Computational Complexity (10 min)
    4. Comparing Sorting Algorithms (15 min)
3. Wrap Up (5 min)

Session 2:

1. Getting Started (5 min)
2. Empirical Investigation (40 min)
    1. Introduction (5 min)
    2. Experimental Design (10 min)
    3. Data Collection (25 min)
3. Wrap Up (5 min)

Learning Objectives

CSP Objectives

- *EU CRD-1 - Incorporating multiple perspectives through collaboration improves computing innovations as they are developed.*
    - LO CRD-1.A - Explain how computing innovations are improved through collaboration.

- *EU CRD-2 - Developers create and innovate using an iterative design process that is user-focused, that incorporates implementation/feedback cycles, and that leaves ample room for experimentation and risk-taking.*
    - LO CRD-2.C - Identify input(s) to a program.
- *EU AAP-2 - The way statements are sequenced and combined in a program determines the computed result. Programs incorporate iteration and selection constructs to represent repetition and make decisions to handle varied input values.*
    - LO AAP-2.A - Express an algorithm that uses sequencing without using a programming language.
    - LO AAP-2.B - Represent a step-by-step algorithmic process using sequential code statements.
    - LO AAP-2.L - Compare multiple algorithms to determine if they yield the same side effect or result.
- *EU AAP-3 - Programmers break down problems into smaller and more manageable pieces. By creating procedures and leveraging parameters, programmers generalize processes that can be reused. Procedures allow programmers to draw upon existing code that has already been tested, allowing them to write programs more quickly and with more confidence.*
    - LO AAP-3.D - Select appropriate libraries or existing code segments to use in creating new programs.
    - LO AAP-3.F - For simulations: a. Explain how computers can be used to represent real-world phenomena or outcomes. b. Compare simulations with real-world contexts.
- *EU AAP-4 - There exist problems that computers cannot solve, and even when a computer can solve a problem, it may not be able to do so in a reasonable amount of time.*
    - LO AAP-4.A - For determining the efficiency of an algorithm: a. Explain the difference between algorithms that run in reasonable time and those that do not. b. Identify situations where a heuristic solution may be more appropriate.

Math Common Core Practice:

- MP5: Use appropriate tools strategically.

Common Core Math:

- S-ID.1-4: Summarize, represent, and interpret data on a single count or measurement variable
- S-ID.5-6: Summarize, represent, and interpret data on two categorical and quantitative variables

Common Core ELA:

- WHST 12.1 - Write arguments on discipline specific content
- WHST 12.4 - Produce clear and coherent writing in which the development, organization, and style are appropriate to task, purpose, and audience
- WHST 12.6 - Use technology, including the Internet, to produce, publish, and update writing products
- WHST 12.7 - Conduct short as well as more sustained research projects to answer a question

NGSS Practices:

- 1. Asking questions (for science) and defining problems (for engineering)
- 2. Developing and using models
- 3. Planning and carrying out investigations
- 4. Analyzing and interpreting data
- 8. Obtaining, evaluation, and communicating information

Essential Questions

- How does abstraction help us in writing programs, creating computational artifacts and solving problems?
- How can computational models and simulations help generate new understanding and knowledge?
- What kinds of problems are easy, what kinds are difficult, and what kinds are impossible to solve algorithmically?
- How are algorithms evaluated?

Teacher Resources

Student computer usage for this lesson is: **required**

Sorting:

- Python code for bubble sort and other sorting methods (for use in Session 1 guided activity):
    - http://interactivepython.org/runestone/static/pythonds/SortSearch/sorting.html (http://interactivepython.org/runestone/static/pythonds/SortSearch/sorting.html)

- Sorting Algorithms Animation timing comparison tool (for use in Session 2 guided activity):
  - http://www.sorting-algorithms.com/ (http://www.sorting-algorithms.com/)
- Many online resources are available to help in understanding different sort algorithms.
  - 15 Sorting Algorithms in 6 minutes
    - https://www.youtube.com/watch?v=kPRA0W1kECg (https://www.youtube.com/watch?v=kPRA0W1kECg)
  - Bubble Sort versus Quick Sort
    - https://www.youtube.com/watch?annotation_id=annotation_3243502817&feature=iv&src_vid=92WHN-pAFCs&v=aXXWXz5rF64 (https://www.youtube.com/watch?annotation_id=annotation_3243502817&feature=iv&src_vid=92WHN-pAFCs&v=aXXWXz5rF64)
  - Merge Sort versus Quick Sort
    - https://www.youtube.com/watch?annotation_id=annotation_492880&feature=iv&src_vid=aXXWXz5rF64&v=es2T6KY45cA (https://www.youtube.com/watch?annotation_id=annotation_492880&feature=iv&src_vid=aXXWXz5rF64&v=es2T6KY45cA)
  - Merge Sort in more detail
    - http://www.zutopedia.com/ms_vs_qs.html (http://www.zutopedia.com/ms_vs_qs.html)
  - Classic “Sorting Out Sorting” four parts
    - https://www.youtube.com/watch?v=YvTW7341kpA (https://www.youtube.com/watch?v=YvTW7341kpA)
    - https://www.youtube.com/watch?v=plAi7kcqMNU (https://www.youtube.com/watch?v=plAi7kcqMNU)
    - https://www.youtube.com/watch?v=gtdfW3TbeYY (https://www.youtube.com/watch?v=gtdfW3TbeYY)
    - https://www.youtube.com/watch?v=wdcoRfS8edM (https://www.youtube.com/watch?v=wdcoRfS8edM)
  - Sorting Animations
    - https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html (https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html)
  - Comparing three sorts
    - http://vinayakgarg.wordpress.com/2011/10/25/time-comparison-of-quick-sort-insertion-sort-and-bubble-sort/ (http://vinayakgarg.wordpress.com/2011/10/25/time-comparison-of-quick-sort-insertion-sort-and-bubble-sort/)
  - Comparison of Sorting Algorithms Approach
    - http://warp.povusers.org/SortComparison/index.html (http://warp.povusers.org/SortComparison/index.html)
  - Integers
    - http://warp.povusers.org/SortComparison/integers.html (http://warp.povusers.org/SortComparison/integers.html)
  - Integers with High Repetition
    - http://warp.povusers.org/SortComparison/integers_rep.html (http://warp.povusers.org/SortComparison/integers_rep.html)
  - Strings (slow compare fast move)
    - http://warp.povusers.org/SortComparison/strings.html (http://warp.povusers.org/SortComparison/strings.html)
  - Arrays (fast compare slow copying)
    - http://warp.povusers.org/SortComparison/arrays.html (http://warp.povusers.org/SortComparison/arrays.html)

Lesson Plan

Session 1

Getting Started (5 min)

Think-Pair-Share: Alternate Routes

- If you assigned the homework from the previous lesson, ask your students to get out their journals to discuss their entries. If not, you could have them write a response in their journal to the following prompt:
  - Identify two places that you often travel between. Of the alternative routes available, what do you consider to be the best route? Why? Are there circumstances in which an alternate route is better? When is that the case?
- Have your students pair off to discuss their responses for a minute or two.
- Ask some of the pairs to share and summarize their journal entries.

Guided Activity (40 min) - Good Algorithms and Better Algorithms

Briefly discuss with your class the topic: what properties make for a good algorithm? What makes one algorithm better than another? Properties you may want to discuss if your students do not volunteer them:

- correctness
- ease of understanding
- elegance (clarity, simplicity, and inventiveness)
- efficiency

A good analogy is purchasing a car, where people are concerned about:

- safety
- ease of handling
- style
- fuel efficiency

Today's session will address the topic of efficiency.

Algorithmic Efficiency [10 min]

Introduce the concept of algorithmic efficiency to your students by asking them if any can describe what algorithmic efficiency is, or what it means for an algorithm to be efficient. Briefly describe efficiency as how well an algorithm uses two resources, time and space (stored memory), to solve a problem. Some topics you may wish to discuss include:

- Two algorithms may both solve the same problem correctly, but with different degrees of efficiency.
- An algorithm that is maximally efficient will minimize the resources it uses.
- Algorithms typically face a space-time tradeoff, where they either use more memory to run faster or take more time but use less memory.
    - When you use a map, you are using more storage resources to go along your route more quickly
- An example of an algorithm that trades space for time (stores more in memory to operate faster) is a lookup table.
    - A real-world example of a lookup table is numbered valet parking. The valet gives the customer a number and goes to park the customer's vehicle in the parking space with that number. When the customer or valet needs to find the vehicle again, instead of having to search through all the spaces, all they need is the remembered (stored) number to go directly to that parking space.
- Most of the time, we are more interested in *computational efficiency*, or time usage of an algorithm.
- A decision problem is a problem with a yes/no answer (e.g., is there a path from A to B?). An optimization problem is a problem with the goal of finding the "best" solution among many (e.g., what is the shortest path from A to B?).
- Efficiency is an estimation of the amount of computational resources used by an algorithm. Efficiency is typically expressed as a function of the size of the input.

Computational Complexity [10 min]

A central idea of analysis of algorithms is that many algorithms will take more and more resources (time or space) as the size of their input increases. Algorithms can be different yet produce the same result. Algorithm's that require polynomial times are considered to run in a reasonable amount of time. Algorithms with exponential or factorial efficiencies are examples of algorithms that run in an unreasonable (slow) amount of time.

When comparing algorithms, time is not measured in seconds but rather in the number of computational steps needed for the algorithm to finish operation on a given input. Great algorithms grow linearly, at the same rate as their input, meaning the time it takes to finish is directly proportional to the size of the problem they are solving (amount of input data). For instance, an algorithm that takes 10 steps for an input of size 10 and 1000 steps for an input of size 1000 is said to be linear in its input. However, most algorithms take longer as their input gets larger. For instance, an algorithm that takes only 25 steps for an input of size 5 may take 100 steps for an input of size 10, 10000 steps for an input of size 100, and one million steps for a size of only 1000 (it is taking quadratically more time as the input gets larger).

When we analyze algorithms, we often talk about the algorithm's *computational complexity*, which is the order of magnitude of the algorithm's running time. An algorithms efficiency and be estimated by counting the nunber of times the code is executed hoever algorithmic efficiency is calcualted through formal reasoning rather than empirical measures.

Efficiency is typically expressed as a function of the size of the input. For example, if an algorithm finishes with the same number of steps regardless of the size of its input, it is called *constant time*, which is O(1) in mathematical form (read aloud as "big-oh one"). Constant time algorithms are the fastest in terms of computational efficiency, and any algorithm that takes a constant number of steps is considered O(1). An algorithm that takes 10 steps for an input of size 10 and also takes 10 steps for an input of size 1000 is likely O(1). However, very, very few algorithms are constant time because most algorithms necessarily take longer as the size of their input increases.

An algorithm that can finish by looking at each piece of its input only once is called *linear time* or *linear order*, and is written mathematically as O(n), where n stands for "the size of the input." An algorithm that takes 10 steps for an input of size 10 and also takes 1000 steps for an input of size 1000 is likely O(n). Very few algorithms are linear order, especially if they must compare pieces in their input, such as sorting algorithms. The best sorting algorithms are somewhere between linear time and quadratic *polynomial time*, written as $O(n^2)$, where n^2 stands for "the size of the input, squared." Any algorithm that is $O(n^2)$ typically must compare each piece of its input with every other piece of input at least once. An algorithm that takes 100 steps for input of size 10 and a million steps for input of size 1000 is likely $O(n^2)$.

Algorithm's that require polynomial - not exponential or factorial times - are considered to run in a reasonable amount of time.

Most sorting algorithms are of an order between O(n) and $O(n^2)$ known as *linearithmic time*, written as O(n log n), where log is the logarithmic function. In fact, O(n log n) is the fastest possible order for a comparison-based sorting algorithm. It is impossible for such algorithms to be O(n) since they must make at least some comparisons of their input data.

Comparing Sorting Algorithms [15 min]

Using the simulation tools at http (http://www.sorting-algorithms.com/)://www.sorting-algorithms.com (http://www.sorting-algorithms.com/)/ (http://www.sorting-algorithms.com/), students will investigate, compare, and contrast sorting algorithms. Notice the grid in the center of the page. Each column is a particular sorting algorithm, and each row is an ordering of horizontal bars (either random, nearly sorted, reversed order, or few unique). Each algorithm will sort the bars in a given cell from top to bottom in increasing order by length.

Ask your students to interact with the website by clicking the green start icons and observing how long it takes each algorithm to sort its bars relative to the other algorithms.

Some questions to have them discuss or record in their journal could include:

- Find the row for "Random" and click the icon above it to see each algorithm sort a randomly ordered set of bars.
- Which algorithms are going slow on average? Which ones are fastest?
- Experiment with larger input sizes by clicking a number for Problem Size near the bottom of the page (30, 40, or 50). Click the icon above Random again. What changes do you notice in the speeds of algorithms? Why are the slow algorithms taking even longer than before? Would you ever want to use them?
- Set the problem size back to 20.
- Find the column for "Bubble" (bubble sort) and click the icon above it to see it run on each of the ordering types. Which one finishes first? Why do you think that is?
- Find the row for "Nearly Sorted" and click the icon above it to see all the algorithms run on nearly sorted input data. Which algorithms finish first? What algorithm is slow on random data but finishes quickly on nearly sorted data? Why do you think it does so?
- Which algorithms do you think are $O(n^2)$?

Make sure your students understand that the size and order of input data can affect how long an algorithm takes. You should direct or help your students discover that bubble sort is a slow sorting algorithm that can be fairly fast for nearly sorted data. You may wish to discuss that bubble sort is $O(n^2)$ in the worst case, explaining why it takes so long for large input, but is O(n) in the best case, which is when input is already (or nearly) sorted. In contrast, selection sort is $O(n^2)$ in both the worst and best cases, and merge sort is O(n log n) in both the worst and best cases. In general, most sorting algorithms that we would want to use are O(n log n), since $O(n^2)$ is usually too slow. You may also want to mention that bubble sort is considered one of the most inefficient sorting algorithms and that quick sort's worst performance is on already sorted data, so some quick sort implementations shuffle the inputs before sorting to avoid that situation.

Wrapup (5 min)

Watch one or more of the available movie clips that compare the performance of sorting algorithms:

Suggested list of videos (Many more are available):

- 15 Sorting Algorithms in 6 minutes
  - https://www.youtube.com/watch?v=kPRA0W1kECg (https://www.youtube.com/watch?v=kPRA0W1kECg)
- Bubble Sort versus Quick Sort
  - https://www.youtube.com/watch?annotation_id=annotation_3243502817&feature=iv&src_vid=92WHN-pAFCs&v=aXXWXz5rF64 (https://www.youtube.com/watch?annotation_id=annotation_3243502817&feature=iv&src_vid=92WHN-pAFCs&v=aXXWXz5rF64)
- Merge Sort versus Quick Sort
  - https://www.youtube.com/watch?annotation_id=annotation_492880&feature=iv&src_vid=aXXWXz5rF64&v=es2T6KY45cA (https://www.youtube.com/watch?annotation_id=annotation_492880&feature=iv&src_vid=aXXWXz5rF64&v=es2T6KY45cA)

- Classic “Sorting Out Sorting” in four parts
  - https://www.youtube.com/watch?v=YvTW7341kpA (https://www.youtube.com/watch?v=YvTW7341kpA)
  - https://www.youtube.com/watch?v=plAi7kcqMNU (https://www.youtube.com/watch?v=plAi7kcqMNU)
  - https://www.youtube.com/watch?v=gtdfW3TbeYY (https://www.youtube.com/watch?v=gtdfW3TbeYY)
  - https://www.youtube.com/watch?v=wdcoRfS8edM (https://www.youtube.com/watch?v=wdcoRfS8edM)

Session 2

Getting Started (5 min)

Introduction [5 min]

Journal: Remind your students about the sorting algorithms from the previous session and have them answer the following questions:

- What are some ways in which one algorithm can be better than another, besides efficiency?
- Explain what algorithmic efficiency is by discussing two different sorting algorithms.

Guided Activity: Empirical Analysis (40 min)

The students will measure and analyze the effect of sorting set size on execution time for a given sorting algorithm using Python code. Using the timedsorts.py file in the lesson resources folder as a basis, the students will perform an experimental analysis to compare sorting algorithms by timing them on input data of different sizees. They will hypothesize, design and code their experiment, collect results, and write a report for homework.

Sorting functions available in the Python code include: quick sort, merge sort, selection sort, insertion sort, and bubble sort. For advanced students or classes may, you may wish to have them implement additional sorting algorithms.

The sample code includes helper functions to generate random data, to load data from a file, and to time sorting functions on the data. Example code for invoking these functions is included at the end of the file. You can remove this example code before sharing it with your students if you wish to emphasize the programming and critical thinking required to do this project.

Each student (or pair or group) needs their own copy of the Python code to modify for their experiments.

Experimental Design [15 min]

Students will compare sorting algorithms by timing them with Python code on input data of various sizes. Have your students (individually or in pairs) make a hypothesis about what will happen as the size of data input increases, answering the following questions:

- How can you determine which sorting algorithm is most efficient and which is least effiicient?
- What sorting algorithm do you think is most efficient, and which is least efficient?
- What do you hypothesize will happen to the time as the size of the data input increases?
- What is the independent variable in this experiment?
- What is the dependent variable?

Have your students write out a description of the steps they will take to perform the experiment.

Data Collection [20 min]

Have students modify their Python sorting code to implement the experimental steps they outlined. Students must:

- Time their sorting routines with different size sets of items to sort (e.g., 5000, 10000, 25000, 50000). Sample data files are available in the lesson resources folder, but students should use the provided helper function to generate arrays of random data, too.
- Record (write down) the size of each input array, the name of the sorting function, and the resulting time it took to sort the data for each algorithm/data combination they test.

Wrapup (5 min)

Discuss the results with another student or group. What patterns can be seen in the relationship between the amount of data and the time to run the program?

Homework

Explain the difference between algorithms that run in reasonable time and those that run in an unreasonable amount of time. Address how algorithms that run in reasonable and unreasonable times are categorized and what impact the difference has on program design.

Optional Extension

Students complete a short research report on their sorting algorithm research procedure, results, and analysis of the results.

- What algorithm or algorithms are most efficient? Why?
- What algorithm is least efficient? Why?
- What values did you use for your independent variable?
- Present the data you collected in a table and in a graph.
- What conclusions can you draw about sorting algorithms?
- Explain why algorithmic efficiency is important by discussing another problem (not sorting) where a correct but inefficient algorithm is unusable at larger input sizes.
- Pick two sorting algorithms you tested. Write a paragraph for each describing how it works, and one paragraph comparing the two algorithms explaining which is more efficient and why (you can do research and look at the Python code to figure out the reasons).

Options for Differentiated Instruction

The teacher may decide to have the students choose how they want to organize the empirical analysis effort. Alternatively, scaffolding with a worksheet or checklist could be used to guide the students through the data collection and analysis tasks.

Evidence of Learning

Formative Assessment

The following "Checks for Understanding" could be used to guide the students towards the three learning objectives:

Objective: SWBAT identify families of correct algorithms that have different efficiencies in their problem solving approach.

1. Students will pair-share what makes a good choice for the route taken to get from point A to point B.
2. Students will compare algorithms and explain why and when some are better than others in terms of efficiency.
3. Students will be able to identify and rank order the least efficient sorting algorithms in the simulations.

Objective: SWBAT demonstrate logical reasoning and metrics is used to describe an algorithm's efficiency.

1. Predict: Students will have seen sorting algorithms implemented as folk dances. Students will predict -- for their algorithm -- how adding additional dancers would increase the dance completion time.

Objective: SWBAT to perform empirical analysis of sorting algorithms by running the algorithms on different inputs.

1. Students will work in pairs to collect data on sorting execution times. The pairs will share their results with other groups to check for patterns before they write up their results.

Summative Assessment

Students will complete a short research report on their sorting algorithm research procedure, results, and analysis of the results.


(http://www.umbc.edu/)

(http://www.umd.edu/)

(http://www.nsf.gov/)

Authored by: CS Matters in Maryland
Website: csmatters.org (http://csmatters.org)
Email: csmattersinmaryland@gmail.com (mailto:csmattersinmaryland@gmail.com)