Sorting, Reductions, And beyond 373

Due date: Monday June 1, 11:59pm

This is so we can publish solutions after the last day to turn it in (June 4th, 11:59pm) and you'll have time to ask questions about it.

Instructions:

Submit your responses to the "Exercise 6" assignment on Gradescope here:
https://www.gradescope.com/courses/97095. Make sure to log in to your Gradescope account using your UW email to access our course.

1. Sorting design decisions

For each of the following scenarios, choose the best sorting algorithm from the list below. Justify your choice in 2 - 3 sentences and be sure to explain how your sort addresses the problems of particular prompt.

insertion sort, selection sort, heap sort, merge sort, quick sort

(a) You are a game programmer in the 1980s who is working on displaying a sorted list of enemy names that a player has encountered during their gameplay. Since it is a game, you want to display the names of the enemies fast as possible, but because it is the 1980s, your customers are used to and will be okay with occasional slow loading times. Additionally, the game is intended to run normally on consoles that don't have much memory available.

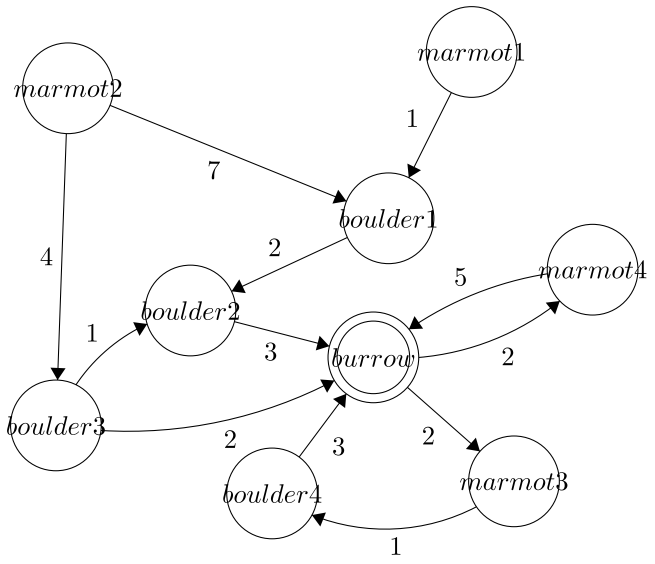
(b) Imagine that you are sorting a **small set** of computer files by their file name. You realize, however, that each computer file is huge and takes up a lot of disk space, so you do not want to copy excessively when sorting. In fact, even just moving and rearranging these large files is expensive, so you don't want to move them often. Hint: You may find it useful to refer to visuals of the sorting algorithms. Lectures slides and https://visualgo.net/en/sorting are good resources for remembering these general ideas.

(c) Imagine that you are a future NASA software engineer. You're assigned a task to sort data you receive from a probe on Mars, in which each piece of data includes time and temperature. The sensors on this probe capture very large amounts of data. The data is already given to you in sorted order of earliest to latest time, but you want to sort them by temperature, where ties in temperature are then sorted by time.

2. Multiple shortest paths problem (using reductions)

Imagine you are a marmot living in Olympic National Park, Washington and are in charge of guard duty; i.e. alerting your colony to eagles (your natural predator) and telling your colony the shortest paths back from their current locations to the burrow. If we phrase this as a graph problem, you can imagine that the vertices for this graph are locations: for example, boulders where the marmots can hide temporarily behind on the way back to the burrow (marmots' starting locations are also at boulders) and the burrow itself. There can be directed edges to represent possible paths between these locations and we can associate each edge with a weight representing the time it takes a marmot to travel on that path.

Below is a potential graph, with the marmots' location vertices named as marmot1, marmot2, etc. and the target location (the burrow) is also labeled. Every other vertex in the graph is just a boulder.



We would want to compute the shortest path from each marmot to the burrow, so for this particular graph the shortest paths we would want to compute would be the following:

- marmot1's shortest path would be marmot1, boulder1, boulder2, burrow
- marmot2's shortest path would be marmot2, boulder3, burrow
- marmot3's shortest path would be marmot3, boulder4, burrow
- marmot4's shortest path would be marmot4, burrow

One way we could compute the shortest routes from every marmot to the burrow is by running Dijkstra's algorithm (for this problem we define Dijkstra's as: given a start node, returns the edges that comprise the entire Shortest Paths Tree in an `edgeToV` map) starting from each marmot's current location and targeting the burrow. If you have k marmots who need to return to the burrow, this process would involve running Dijkstra's algorithm k separate times from each of the k marmots as the source vertex and would incur a worst-case runtime of $\Theta(k * (m * log(n) + n * log(n)))$ where m is the number of edges and n is the number of vertices.

(continued on the next page)

It turns out that by modifiying our graph, we can actually compute the paths for all the marmots to return to the burrow more efficiently. **Perform a reduction by modifying our graph so that afterwards you can compute all the shortest paths by utilizing Dijkstra's in $\Theta(m * log(n) + n * log(n))$ worst case runtime.** Because this a reduction, you don't need to make any changes to Dijkstra's algorithm and only need to modify the graph to achieve this runtime. **In 2-3 sentences, specify the changes to the graph.** (Are there new edges or vertices?. Are existing edges or vertices modified? Do the edges and / or vertices have different meaning?) Also, **be sure to specify the starting node for when you run Dijkstra's on your new graph**.

Hint 1: You may need to change the entire graph slightly instead of focusing on a change to a single vertex or edge.

Hint 2: You can try to work backwards because the expected runtime bound is given – if we are allowed $\Theta(m * log(n) + n * log(n))$ runtime, how does that affect how many times we can run Dijkstra's?

Warning: The reduction we're expecting here is different than the one necessary for the Seam Carving project. (The Seam Carving problem involves finding a single path whereas here we actually want multiple paths.)

For some completely optional context on marmots, feel free to use this National Geographic video as extra information.

3. Data structures and algorithms beyond CSE 373

This next problem is about learning something not covered in CSE 373 but will be graded only on completion (if you turn this in and show 30 min of effort). Try to spend at least 30 minutes on this, but feel free to continue exploring for as much time as you want.

In this course we've learned a lot of various data structures and algorithms concepts (link to a full list), but there's even more that we didn't cover. It's impossible for a course to covery every single data structure and algorithm out there, but the fundamental ideas we've learned will frequently appear in future data structures and algorithms you encounter.

As a small exercise to practice your data structures and algorithms literacy and learning skills, **take some time (at least 30 min) to try to learn about one new algorithm or data structure idea that we haven't discussed yet and write a brief response afterwards**. The course staff has listed a couple of our favorites below, with some reasonable videos and articles to get you started. However, feel free to choose something outside of this list and / or do your own research. If you have a particular field of interest, say biology for example, you could try to search for 'biology algorithms' or 'computational biology algorithms' to find something suited to your interests.

- PageRank *(ranking search results, Google, graphs, machine learning)*
  - https://www.youtube.com/watch?v=edoMiAYLNe8
  - https://www.youtube.com/watch?v=u8HtO7Gd5q0
- Term Frequency and Inverse Document Frequency(TF-IDF) *(ranking search results, Google, machine learning, natural language processing)*
  - https://www.youtube.com/watch?v=RPMYV-eb6lI
- Minimax and Alpha-beta pruning *(artifical intelligence, graphs, games)*
  - https://www.youtube.com/watch?v=l-hh51ncgDI
  - https://www.youtube.com/watch?v=STjW3eH0Cik
- A-star search *(artifical intelligence, graphs, path-finding, Dijkstra's)*
  - https://www.youtube.com/watch?v=ySN5Wnu88nE
- Count-min sketch *(hashing, probabilistic data structure, modern, large data)*
  - https://www.youtube.com/watch?v=ibxXO-b14j4
  - https://florian.github.io/count-min-sketch/
- Consistent hashing *(hashing, distributed systems, modern)*
  - https://www.toptal.com/big-data/consistent-hashing
  - https://www.youtube.com/watch?v=tHEyzVbl4bg
- Reservoir sampling *(large data, interview question)*
  - https://florian.github.io/reservoir-sampling/
- Max flow problem *(graphs, image segmentation, sports league elimination)*
  - https://www.youtube.com/watch?v=oHy3ddI9X3o
  - https://www.youtube.com/watch?v=VYZGlgzr_As
  - http://www.robots.ox.ac.uk/~az/lectures/opt/lect4.pdf
  - https://digitalcommons.georgiasouthern.edu/cgi/viewcontent.cgi?article=2720&context=etd
- Articulation points *(graphs, networks, biology)*
  - https://www.nature.com/articles/ncomms14223
  - https://www.youtube.com/watch?v=jFZsDDB0-vo

  - https://europepmc.org/article/pmc/pmc4766943

- Sequence alignment *(biology, dynamic programming)*
  - https://hoberg.github.io/Smith-Waterman/ (opening in Firefox may appear buggy)
  - https://www.youtube.com/watch?v=9bCkAsaP_z4

For reference, A-star search is probably the most similar to the ideas we've talked about in class (it's an algorithm that involves some modifications to Dijkstra's algorithm), while Sequence alignment and Term Frequency and Inverse Document Frequency probably have the least overlap with this course on this list. Everything else is likely somewhere in-between.

For the 1 algorithm or data structure that you decide to learn about, write at least 1 paragraph (4-5 sentences) about what you learned and include any reflections you made while learning. Feel free to use the following questions to structure your response:

- What sort of use cases is your algorithm or data structure involved in?
- What's something tricky / interesting about your data structure or algorithm?
- What sort of runtime or memory analysis did you learn about your data structure or algorithm? Are there any other tradeoffs to consider when choosing this?
- Is your data structure or algorithm similar to anything from this course? In what ways to do they differ?

As a reminder, this problem will be graded on completion. Please feel free to stop yourself after 30 minutes of research if you're busy or have other things to take care of. If you are concerned that this will be 30 minutes of busywork, then consider trying to find an algorithm or data structure that excites you before you start.