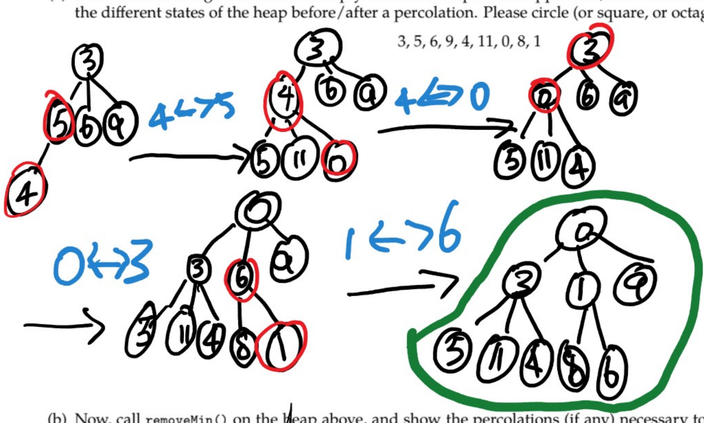
Week 7 Review Session

CSE 373 19su

1. A Heap of Mechanical Problems[1]

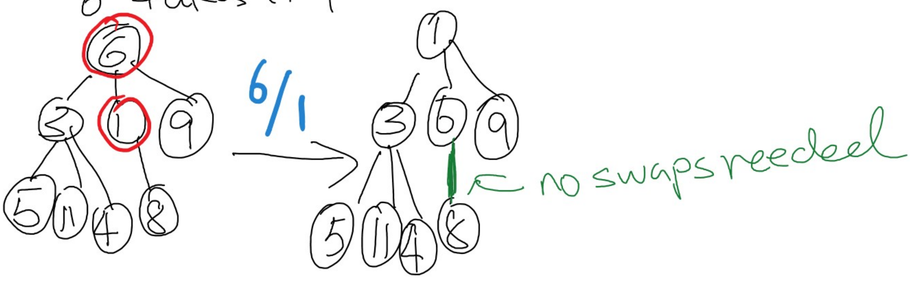
   (a) Insert the following values into an empty **three**-minheap. When applicable, draw out intermediate steps, such as the different states of the heap before/after a percolation. Please circle (or square, or octagon) your final result.

   3, 5, 6, 9, 4, 11, 0, 8, 1



   (b) Now, call `removeMin()` on the heap above, and show the percolations (if any) necessary to maintain the heap invariant.

0 is removed
6 takes its place



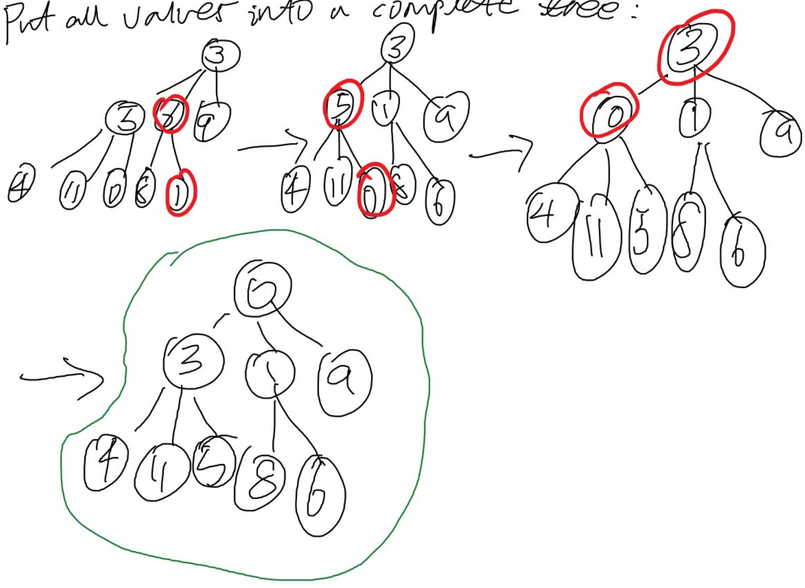
[1] In case you weren't at section, https://xkcd.com/835/

   (c) You are again working with a three-minheap and the same set of values:

   3, 5, 6, 9, 4, 11, 0, 8, 1

   Instead of inserting the values one-by-one, use the **Floyd's build-Heap** [2] algorithm to create a heap using the above values. Show the initial "heap" without percolations, then show all intermediate steps before you arrived at your final answer. Were you able to arrive at the same heap?

Put all values into a complete tree:



It's actually slightly different than our heap in part a!

[2] CSE 373/332 seem to be the only courses in the US that consistently uses the phrase "Floyd's buildHeap". USC and Columbia use "buildHeap"; University of Toronto, University of Virginia, Oregon State and New Mexico State use "BuildHeap"; Duke uses "Build-heap"; Florida State and Texas A&M use "Build-Heap". This question prompt therefore takes the average of each of these variations in consideration of scientific consensus (or the lack of it).

2. Design Decisions -AFTER STORY-

(a) Here are a couple of computational tasks for your consideration. For each one, choose the data structure that would best suit its needs. Briefly convince yourself of your response. Remember that there might be multiple valid answers.

Data structures: hash table, AVL tree, heap

i. A popular way to figure out what personal debt to pay towards next is described by the Debt Snowball[3] method, where you always pay off the smallest debts in amount due before moving on to the larger ones, regardless which debt you took out first. You are writing a personal finance app and your app needs to recommend which debt a user should pay towards next.

Heap → removeMin.

You can even run topKSort (k=1).

ii. You are given a long string of text, and you want to know if it contains only unique characters. The text may contain non-Latin alphabet, like Chinese characters and emojis, so you should not make an assumption about the number of possible unique characters.

Hash table (implement set)

return set.size() == string.length()

iii. You are writing a web server logger that records all requests to the CSE 373 website. Once in a while, you want to examine the logs and order them by response time, to see which parts of the website are have bottlenecks that cause inefficiency.

AVL Tree + inorder traversal.

Or see i).

(b) You are writing your own implementation of Quicksort[4]. Recall that when you are given an input list, you will have to decide on a pivot selection strategy before you can begin sorting using Quicksort.

i. For each of the following pivot choices, describe a situation (a specific example or in general) where it would be a good idea, and another situation where it would be a bad idea. Remember that the pivots determine how evenly the partitions are split for future recursive calls to QuickSort.

A. The pivot is always the first index.

Good: Random data.

Bad: Sorted, reverse sorted data

[][1][234567]
↓
[][2][34567]
↓
[][3][4567]

$O(n^2)$

B. The pivot is always the middle index.

Good: Sorted data. [12][3][45]

Bad: Element chosen is min / max of all.

C. The pivot is a randomly chosen index.

Good: For most cases when you stumble upon a number close to the median.

Bad: When you hit cases in A and B.

[3] https://en.wikipedia.org/wiki/Debt-snowball_method

[4] Lecture slides for review: https://tiny.cc/lecture15

3. Sort Out This Algorithm

(a) Consider the following code that sorts an list of numbers in ascending order:

```
public void sort(ArrayList<Integer> list) {
    for (int i = 0; i < list.size(); i++) {
        int minNum = list.get(i);
        int minAt = i;
        for (int j = i + 1; j < list.size(); j++) {
            if (list.get(j) < minNum) {
                minNum = list.get(j);
                minAt = j;
            }
        }
        if (i != minAt) {
            int temp = arr.get(i);
            list.set(i, minNum);
            list.set(minAt, temp);
        }
    }
}
```

The pivot moves L->R

Find the minimum number

Swap with pivot if needed

What sorting algorithm does the code above seem to implement?

Selection Sort

(b) What is the best, in practice and worst case of the algorithm? Give sample input(s) to explain your answer.

$\Theta(n^2)$ everything

1 2 3 4 5

Regardless if the list is sorted, you always do $\sum_{i=0}^{n-1} \sum_{j=0}^{i-1}$ work.

(c) Is this algorithm in place? How could you tell?

Yes. We don't make new data structures / no recursion = no new call stacks.

(d) Is this algorithm stable? How could you tell?

No. $[2_1\ 2_2\ 1] \rightarrow [1\ 2_2\ 2_1]$

(e) Sort the numbers below using the algorithm (you can either rely on the code above or just utilize your understanding of the sorting algorithm). Please show work, and note down the swaps that the algorithm performs.

3, 1, 4, 1, 5, 9

Sorted | Unsorted

3 | 1 4 1 5 9

1 3 | 4 1 5 9

1 1 4 | 3 5 9

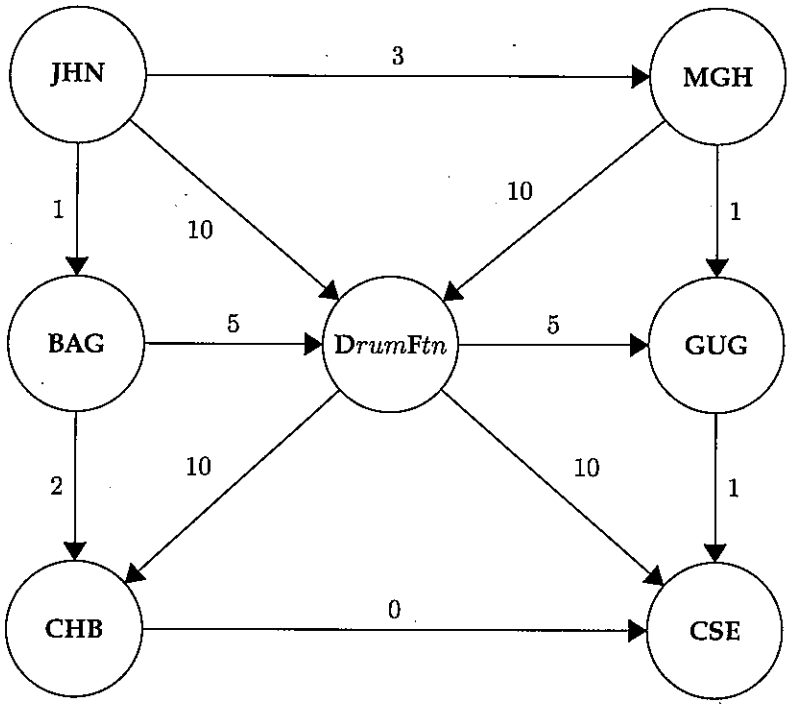
1 1 3 4 | 5 9 No swap

1 1 3 4 5 | 9 No swap

$\boxed{113459}$

4. Negotiating The Avian Séance [5]

You are travelling through the Drumheller Fountain area, and your main goal is to avoid the water splashes that are unfortunately contanimated by goose droppings. Fortunately, your research mentor in Atmospheric Sciences modeled Seattle wind velocities and was able to calculate the amount of water that a pedestrian is estimated to receive while traversing between two buildings near the fountain:



Run Dijkstra's Algorithm to find the best available path and its cost from your mentor's lab in **JHN** to Robbie's Office Hours in **CSE**. You should show intermediate steps (by crossing out intermediate values instead of erasing them). If you need to break ties between two vertices of equal cost, order again by lexical order.

Vertex	Cost (mL)	Predecessor	Processed
JHN	0	N/A	0
MGH	~~4~~ 3	JHN	4
BAG	~~4~~ 1	JHN	1
DF	~~4~~ ~~16~~ 6	~~DF~~ BAG	6
GUG	~~4~~ 4	MGH	5
CHB	~~4~~ 3	BAG	2
CSE	~~4~~ 3	CHB	3

JHN → BAG → CHB → CSE

The order is given here, but you technically only need a checkmark.

[5] https://redd.it/ci93cn

5. I Hope I Get It

In an effort to help CSE 373 students concentrate better, Robbie has asked UW-IT to install wireless jammers in PAA A118 so that students will no longer have access to the Internet during lecture. Because you are very eager to continue chatting with your other friends in the class, you decide to create a decentralized chat app using Bluetooth [6].

This kind of chat apps utilize a mesh network so that messages can hop between multiple phones. For example, if Kevin is in range of Zach, and Zach is in range of Matt, then Kevin will be able to send a message to Matt (with one hop via Zach's phone). On the other hand, Howard, who is sitting on the other side of the classroom, isn't in range of anybody else, and therefore he won't be able to send messages to Kevin, Matt or Zach.

As you develop your chat app, you want to better facilitate mesh network messaging using a graph. Please note that these design decision questions are open-ended - explain your assumptions for full credit.

(a) For a graph used to model this situation, what would your vertices and edges represent?

Vertices represent users (or their phone).
Edges represent two phones that are in range of each other.

(b) What information would you store for each vertex and each edge?

Vertex: name of user
Edge: No info needed if unweighted. Otherwise it makes sense to store 1 or the signal strength, etc.

(c) Is your graph: weighted or unweighted? Directed or undirected? Are self-loops acceptable? Do parallel edges make sense? Very briefly explain.

It depends on you :) My answers:
Weighted, I probably want to store signal strength.
Undirected, if two phones are in range I think it should go both ways.
No self-loops. It doesn't "cost" to reach yourself.
No parallel edges. Two phones don't have more than one link.

(d) Before you can send a message to another person, your app needs to figure out the path that the message will travel. Howard knows Oscar is somewhere in the room, but doesn't know how to reach him exactly. Using the graph that you have designed, explain what algorithm you would use/adapt to explore the best path to reach Oscar. Your algorithm should be able to tell if it is actually impossible to reach Oscar. Explain how your algorithm will work and what information will it be storing.

Run Dijkstra from Howard to Oscar. Standard Dijkstra's table.
Note: You cannot use Dijkstra's if your graph is unweighted.
In this case adapt BFS.

[6] It should be noted here, in the interest of ECE majors taking this course, that this pretext is actually unlikely in real life. This is because both WiFi and Bluetooth operate on the 2.4GHz ISM band, and to interfere with WiFI operations it is also likely that Bluetooth is affected. Unfortunately, WiFi and Bluetooth seem to be the only mechanisms available for a mesh network on a mobile phone, so we have to make do. Please write a Piazza post if you have any suggestions on improving the realism of this problem for future quarters.

(e) Robbie devised a classroom learning activity and asked the TAs to discuss the answers amongst themselves. In this activity, the left hand side of the classroom will be discussing mergesort, while the right hand side of the classroom will be discussing quicksort. Because the TAs on one side of the room do not want to bother the TAs on the other side with unrelated chatter, they want to be able to limit the hops that their messages can reach (they hypothesize that 2 hops/10 meters on Bluetooth should be enough to only reach one side of TAs). What algorithm would you use to efficiently discover the names of all TAs that can be reached within a certain number of hops? Explain how your algorithm will work and what information will it be storing.

Run BFS. Maintain Node <> # of hops hash map, and do not traverse further if already reached 2 hops.