## Context

When UW switched to online-only instruction in Winter Quarter 2020, CSE 373 cancelled its final and implemented a “final assignment” or “graded final review packet”.

## Questions

### Question 1: Instructions

All point values are temporary, and will be adjusted later.

You can punch the Submit button below a couple of times to relax.  Actually, you can submit your answers as many times as you'd like (and we'd recommend doing it fairly frequently); we will only grade your final submission.

To add people into your group, click Submit below, then scroll to the very bottom of the page and click View Your Submission.

### Question 2: Graph Modelling: Twitter Followers

The handlers of Dubs, UW's mascot, noticed that he does not follow anyone on Twitter. Because a lot of people already follow Dubs, they decide to start by following back people who either:

- Already follow Dubs directly.
- Follow somebody who follows Dubs.

You are given a graphical representation of Twitter user relationships, accessible using the following class:

```
public class TwitterGraphAPI {
    private static final String DUBS_USERNAME = "DubsUW";

    public List<MysteryEdge<String>> neighbors(String username) {
        // Returns a list of users who follow `@username` on
Twitter.
    }
}
```

You are tasked with finding a list of Twitter accounts for Dubs to follow.

## Q2.1: Analyzing the Model

In the graph representation that you were given, what do the vertices represent?

What kind of relationship do the edges represent?

Does it make sense for this graph to be directed? Briefly justify in 1 sentence.

Does it make sense of this graph to be weighted? Briefly justify in 1 sentence.

## Q2.2: Adapting Algorithms

Let's formulate an algorithm that, when given the Twitter username of Dubs to start, it will return a list of Twitter usernames that Dubs will follow.

When you ran an existing graph algorithm on Dubs' username, you noticed that the algorithm doesn't work, because Dubs does not follow anyone and Dubs' vertex does not have any out-neighbors. Describe in 1–2 sentences how you would modify the graph so that it would be possible to run a graph algorithm you know starting on Dubs' vertex.

Now that you have a better graph, how would you modify one of the graph algorithms that we learned so that you could obtain the desired results? As a reminder, you want a list of Twitter usernames for people who either:

- Already follow Dubs directly.
- Follow somebody who follows Dubs.

## Question 3: Sorting with Character Comparisons

Inspired by RadixSort, you decide to analyze the runtimes of other sorting algorithms on strings, in terms of character comparisons.

Fill out the big-Theta bounds for the runtimes of the following algorithms that could be used to sort strings, in terms of:

- *n*: number of strings being sorted
- *l*: length of strings being sorted (assume all strings are exactly this long)
- *r*: number of distinct characters in the strings

Assume that the `compareTo` method of Java's String simply calls the following helper method:

```
static int compare(String s1, String s2) {
    int length = Math.min(s1.length(), s2.length());
    for (int k = 0; k < length; k++) {
        char c1 = s1.charAt(k);
        char c2 = s2.charAt(k);
        if (c1 != c2) {
            return c1 - c2;
        }
    }
    return s1.length() - s2.length();
}
```

For your case analysis, assume that $r = n < l$. (You should not use these to substitute into your $\Theta$ expressions—only use them to reason about the cases.)

You don't need to include the $\Theta$ symbol in your answer; i.e., just answer "n^r" instead of "Θ(n^r)".

## Q3.1 MSD RadixSort

Best case:

Worst case:

## Q3.2 In-place InsertionSort

Best case:

Worst case:

## Q3.3 SelectionSort

Best case:

Worst case:

## Q3.4 MergeSort

Best case:

Worst case:

## Q3.5 In-place HeapSort

Best case:

Worst case:

## Q3.6 Naive QuickSort

Best case:

Worst case:

## Q3.7 “TrieSort”

Suppose we added all the strings into a trie, then traversed the trie to re-build the strings in sorted order.

Assume that the trie uses hash tables to store the children of each node.

Best case for building trie:

Worst case for building trie:

If each trie node offers the ability to iterate through its children in alphabetical order, what kind of tree traversal would we use to produce the correct sorted output?

- ○ Pre-order
- ○ In-order
- ○ Post-order
- ○ Level-order

Best case for traversing trie:

Worst case for traversing trie:

## Question 4: Choose Your Own ADT

What ADTs, if any, are needed to implement the following algorithms? Ignore the algorithm inputs and outputs; consider only the types used internally.

For each ADT you select, name a data structure implementation that could be used to implement the algorithm with good asymptotic runtime.

### Q4.1 Prim's Algorithm

☐ List ADT ☐ Stake ADT ☐ Queue ADT ☐ Deque ADT ☐ Set ADT ☐ Map ADT
☐ Priority Queue ADT ☐ Graph ADT ☐ Disjoint Sets ADT ☐ None

Data structure implementation(s):

### Q4.2 Kruskal's Algorithm

☐ List ADT ☐ Stake ADT ☐ Queue ADT ☐ Deque ADT ☐ Set ADT ☐ Map ADT
☐ Priority Queue ADT ☐ Graph ADT ☐ Disjoint Sets ADT ☐ None

Data structure implementation(s):

### Q4.3 Dijkstra's Algorithm

☐ List ADT ☐ Stake ADT ☐ Queue ADT ☐ Deque ADT ☐ Set ADT ☐ Map ADT
☐ Priority Queue ADT ☐ Graph ADT ☐ Disjoint Sets ADT ☐ None

Data structure implementation(s):

### Q4.4 Dual-Pivot QuickSort

☐ List ADT ☐ Stake ADT ☐ Queue ADT ☐ Deque ADT ☐ Set ADT ☐ Map ADT

☐ Priority Queue ADT ☐ Graph ADT ☐ Disjoint Sets ADT ☐ None

Data structure implementation(s):

| |
|---|
| |

### Q4.5 In-Place InsertionSort

☐ List ADT ☐ Stake ADT ☐ Queue ADT ☐ Deque ADT ☐ Set ADT ☐ Map ADT
☐ Priority Queue ADT ☐ Graph ADT ☐ Disjoint Sets ADT ☐ None

Data structure implementation(s):

| |
|---|
| |

## Question 5: Maze Generation

Assume you are generating some mazes (defined as 2D grids with exactly one path from the “start” room to the “end” room). Describe how you would use the Disjoint Sets ADT to do this.

| |
|---|
| |

## Question 6: Sorting

Review the following statements about Java's QuickSort and select the ones that are true.

- ☐ It is only used for reference types; a different sorting algorithm is used for primitive types
- ☐ It uses introspection to switch between MergeSort and QuickSort
- ☐ It uses introspection to switch between InsertionSort and QuickSort
- ☐ Its pivot selection is the median-of-three algorithm
- ☐ Its partitioning algorithm is Hoare Partitioning
- ☐ Its partitioning algorithm can partition the array in a single pass, but the tradeoff is that it is not stable
- ☐ It uses InsertionSort for small input
- ☐ It uses InsertionSort for small subarrays
- ☐ Its runtime has optimal asymptotic growth for a comparison sort
- ☐ It has the best runtime of any known sorting algorithm, as determined by empirical tests
- ☐ When its input is mostly sorted, its runtime is Theta(N^2)

## Question 7: MergeSort: Three's a Party!

Recall that merge sort works by taking an array, splitting it into two pieces, recursively sorting both pieces, then combining both pieces in O(n) time. Suppose we split the array into three equally sized pieces instead of two. And after we finish recursively sorting each of the three pieces, we first merge two of the three pieces together, then we merge that with the third piece to get the final sorted result.

### Q7.1 Recurrence

Complete the recurrence for the runtime of this variation of MergeSort:

| | |
|---|---|
| T(n) = { | 1 if _______<br><br>__________ |

Condition for base case:

| |
|---|
| |

Expression for recursive case:

| |
|---|
| |

### Q7.2 Evaluating Variations

Compare the runtime recurrence and the asymptotic runtime of this variation with that of standard merge sort. Is this variation better, the same, or worse than the standard merge sort? Justify your answer in 1–2 sentences.

| |
|---|
| |

## Question 8: 😍😷🇨🇦🥦🚝⛲3️⃣7️⃣3️⃣

What is your favorite emoji, and why?

| |
|---|
| |