# CSE 373: 24 Wi Final

| | | |
|---|---|---|
| Name: Yafette | UW Email: yafettek | @uw.edu |

## Instructions

- The allotted time is **50** minutes. Please do not turn the page until the staff says to do so.
- This is an open-book and open-notes exam. You are NOT permitted to access electronic devices including calculators.
- We can only give partial credit for work that you've written down.
- If you want to use an algorithm discussed in class, just cite the name of the algorithm, like "run BFS to…"
- If you use a data structure that requires data to be comparable (e.g. a search tree or heap), you should explain how the data is compared. For example, "I will use a binary min heap of (`name, count`) pairs where pairs are **compared by `count` in ascending order**"
- If you use a map, explain what the keys and values are.
- Unless otherwise noted, when we ask for algorithm runtime, it must be **simplified and tight**.
- **You may assume that all hash functions and operations (find, add, remove) are O(1).**
- If you run out of room on a page, indicate where the answer continues. Try to avoid writing on the very edges of the pages: we scan your exams and edges often get cropped off.

## Advice

- If you feel like you're stuck on a problem, you may want to skip it and come back at the end if you have time.
- Relax and take a few deep breaths. You got this :-)

| Questions |
|---|
| 1. Heaps |
| 2. Sorting |
| 3. Graphs |

## Problem 1 (Heaps):

Yafqa is assigning each problem on the final to a TA to grade. Some TAs have already volunteered to grade certain problems, so Yafqa needs to assign the remaining problems. He wants the workload to be as evenly distributed as possible.

To be more specific, there are `k` problems and `n` TAs, where `k > n`. You are given an array `problemAssignments` of length `k`, where `problemAssignments[i]` is either the name of the TA who volunteered for the `ith` problem or `null` if no TA has been assigned to it yet. You are also given an array `tas` of all the TA names.

Let `min` be the fewest number of problems that any TA has been assigned, and let `max` be the greatest. Your task is to assign the remaining problems so that the difference between `max` and `min` is as small as possible. The method header would look something like this:

```
public void assignProblems(String[] problemAssignments, String[] tas)
```

---

### Example

Suppose Yafqa has 7 problems to assign, and some TAs have volunteered as below:

`problemAssignments`

| Problem 0 | Problem 1 | Problem 2 | Problem 3 | Problem 4 | Problem 5 | Problem 6 |
|---|---|---|---|---|---|---|
| Azita | null | Simon | Simon | null | Josh | null |

And suppose we have the following TAs:

`tas`

| TA 0 | TA 1 | TA 2 | TA 3 |
|---|---|---|---|
| Azita | Simon | Josh | Emily |

Then to make the workload as even as possible, we would assign the remaining problems like so:

`problemAssignments`

| Problem 0 | Problem 1 | Problem 2 | Problem 3 | Problem 4 | Problem 5 | Problem 6 |
|---|---|---|---|---|---|---|
| Azita | Emily | Simon | Simon | Azita | Josh | Josh |

There may be multiple valid solutions. For example, problem 6 could have been assigned to Emily instead.

1) Describe how you would compute and store the number of problems each TA in `tas` **initially** volunteered for, **before assigning any of the remaining problems**. If a TA didn't volunteer for any problems, you should record the fact that they volunteered for 0 problems. Whatever you store this data in, it should **enable constant time lookup of a TA's initial count**, in practice. For an **upper bound**, your algorithm should run in O(n + k) time. Your answer should contain a precise description of what data structure(s) you're using, **following the instructions on the front page**, as well as **an algorithm in either plain English or pseudocode**. No justification necessary. (≤ 5 sentences)

Construct a `HashMap` where the key is a TA name and the value is the number of problems the TA initially volunteered for. To initialize the map, iterate over the ta names and set each of their counts to 0 in the map. Then iterate over the problemAssignments array; for each non-null TA name, increase the corresponding count in the map by 1.

2) For this and the following questions, assume you have a data structure that enables constant-time lookup of a TA's initial counts, as in (1). Describe any additional data structures you would use to assign the remaining problems (as described in the problem statement above). Your response should contain a precise description of your additional data structure(s), **following the exam instructions on the front page**. You can use any data structure described in class. You **do NOT need to** justify your choices. (<= 4 sentences)

The solution is on the front page of the exam: "I will use a binary min heap of (`name, count`) pairs where `name` is the name of TA, `count` is the number of problems that TA signed up for, and pairs are **compared by `count` in ascending order**".

3) Now describe how you would assign the remaining problems in `problemAssignments` as described above. Your response should contain an algorithm in either plain English or pseudocode. For an **upper bound**, your algorithm should run in O(n + klog(n)) time in the worst-case. You can abbreviate variable names if you'd like, but they should either be explained or obvious. You **do NOT need to** justify the runtime. (<= 5 sentences).

Build the heap by iterating over the map from 1), adding the (`name, initial_count`) pairs to a list, and running floyd's `buildHeap` algorithm to heapify it. Then iterate over the problemAssignments array; for each unassigned problem, pop off the top (`name, count`) pair in the binary min heap described in 2), assign `name` to this problem, then insert (`name, count + 1`) into the heap.

## Problem 2 (Sorting):

For each of the following situations, name **one sorting algorithm** from lecture that fits the constraints. Choose from among `insertion, selection, heap, quick, merge, radix,` and `bucket`. Explain how your choice meets all constraints. If you make any assumptions about any sorting algorithm, you should explain those too, e.g. if you assume that quicksort always picks good pivots. (≤ 4 sentences each)

1. As an employee at Suzzallo Library, you have been tasked with sorting a recent delivery of electronic books. These books are already arranged in alphabetical order by author. You must now sort them by genre, such that books within the same genre remain sorted by author. However, given the constraints of an older library computer, you must use a sorting algorithm that utilizes limited memory.

One solution: insertion sort. We must retain the relative order of authors, so we need a stable sort. We also have limited memory, so an in-place sort would be ideal. Insertion sort is both stable and in-place, so it satisfies the constraints of the problem.

Notes: some submissions mentioned the fact that since the array is already sorted by author, sorting by genre would be more efficient. This is not true: consider [("Zombies", "Azita"), ("Young Adult", "Balsara"), ("X-rated", "Catherine"), ("Weird", "Dory")]. The genres are in reverse sorted order, which is the worst-case for insertion sort. We ignored this when grading.

Another note: A lot of submissions contained the phrase "iterating through the elements of the list and repeatedly inserting each element into its correct position in the sorted part of the list", which appears to be from ChatGPT. We also ignored this when grading.

2. As an employee at Tema, you have been asked to sort the usernames of all Tema users. However, you realize that the data set is incredibly large, so you want to ensure an efficient worst-case runtime.

One solution: heap sort. Heap sort has a worst-case runtime of O(nlog(n)), so it ensures an efficient worst-case runtime. (doesn't matter if in-place or not; you just need to satisfy the constraints of the problem).

Another solution: Merge Sort. I would use the merge sort algorithm. Merge sort has a worst-case runtime of O(nlogn), which guarantees a relatively fast sorting time for large data sets. By dividing the data into smaller sublists and then merging them in sorted order, merge sort avoids the bottleneck of comparing every element against every other element, making it a suitable choice for sorting a large number of usernames.

Note: Solutions that used radix sort also had to specify that they were assuming that the lengths of usernames were bounded, since otherwise the worst-case runtime would be undefined.

3. As a data analyst at the University of Washington, you have conducted a survey to gather information from students about their favorite courses. The collected data includes the names of students, their student IDs, and their preferred courses. Now, you want to sort the list of students by their ID numbers, as quickly as possible. (You can assume that student ID numbers are integers from 0 to 1,000,000)

One solution: bucket sort. Since the ids are between 0 and 1,000,000, we can use bucket sort with an array of 1,000,001 buckets to quickly sort the ids. The runtime would be linear in the size of the input, meeting the “quick as possible” constraint.

Another Solution: To quickly sort the list of students by their ID numbers, I would use radix sort. This algorithm is efficient for sorting integers within a specific range and can be implemented in linear time.

## Problem 3 “Spill the Tea” (Graphs) :

Mr. Meow Meow decided to reveal their secret identity. They tell some students, those students tell some other students, and so on. You want to figure out how long it takes until the entire school knows their true identity.

Suppose you are given a list of triplets of the form (`gossipGiver`, `gossipReceiver`, `time`), indicating that it takes `time` minutes for `gossipGiver` to spread gossip to `gossipReceiver` (once `gossipGiver` has something to say to `gossipReceiver`). Note that just because `gossipGiver` gossips to `gossipReceiver` doesn’t mean that `gossipReceiver` spreads gossip to `gossipGiver`. You are also given a list of all student names at UW. At time 0, Mr. Meow Meow starts revealing the news. You need to find the earliest time **t** in which everyone on campus knows about it, or output -1 if this is not possible.

You will solve this problem using a graph:

---

In one word, what should each vertex represent in the graph?

People/Students

In one short sentence, what should each edge represent in the graph?

There is an edge from a student *u* to a student *v* iff *u* gossips to *v*.

Should the edges in the graph be directed or undirected?

Select the best answer:

[✓ Directed] Undirected

Should the edges in the graph be weighted or unweighted?

Select the best answer:

[✓ Weighted] Unweighted

If you answered “Weighted” to the previous question, in one short sentence, what should each edge weight represent? If you answered “Unweighted” to the previous question, leave it blank.

The amount of time it takes for one student to spread gossip to another.

Explain how you can determine the earliest time `t` in which everyone knows Mr. Meow Meow's identity, or whether no such time exists. If you plan to use one of the graph algorithms you've learned in class, you do not need to explain how the algorithm is implemented. For example, if you want to use breadth-first search in your answer, simply say "use BFS to achieve…" but if you plan to adapt an algorithm, be sure to explain any changes you would make. (≤ 4 sentences).

Run Dijkstra's algorithm, with Mr. Meow Meow as the source, to get the SPT. Iterate over the list of students on campus to ensure that everyone is contained in the SPT; if not return -1. Otherwise, iterate over the SPT, find the person with the greatest distance from Mr. Meow Meow, and return this distance.

Note: this is a reskin of this leetcode problem:
https://leetcode.com/problems/network-delay-time/description/

Many of these exam problems are reskins of leetcode problems. If you still believe that some modification of Kruskal's/Prim's can solve this problem, we encourage you to try it and see if it passes the tests. Let us know if it is successful.