

## Homework 6: Hash Maps, Proofs, and Trees

Due: Friday, March 6 at 5pm on Canvas

**Concepts:** pre-processing, hash maps, trees, proof techniques

**NOTE:** There is **no** implementation required for this assignment.

1. (9 points) This question builds on problem 2 on Homework 4, in which you designed an algorithm to find a list of all possible patients starting with patient zero. You may want to review the solution to this homework before starting.
  - (a) (6 pts) In this question, you will complete the runtime analysis for your algorithm. Suppose that you are given a list of all students at Olin, and for every class at Olin, you are given a list of students in that class. Further, suppose that at most 25 students are in a class and every student takes at most 5 classes (so these are constants that do not depend on the number of students at Olin). Update your algorithm from HW4 to pre-process these class lists so that your algorithm now runs in  $O(n)$  time, where  $n$  is the number of students at Olin (**Hint:** you may want to use hash maps).
  - (b) (3 pts) Previously, you argued why every student added to the list was a potential patient and why the returned list contains all such students. This question will formalize that last part using a proof by contradiction. That is, suppose that a student caught academitis but was not added to the list. Prove that this leads to a contradiction.
2. (4 points) In class, we argued why Merge Sort always returned a fully sorted list. Formalize that argument using a proof by induction. Your proof should contain your inductive hypothesis, a base case, and an induction step.
3. (12 points) Union-Find is an abstract data type and supports the following operations
  - (a) `make_set(i)`: Creates a set with a single node  $i$ .
  - (b) `union(A,B)`: Merges two sets  $A$  and  $B$ .
  - (c) `find(i)`: Given a node  $i$ , returns the head element of  $i$ 's set.

Here, a set is a collection of distinct elements with a defined head element. Union-find data structures are often used in applications in which you want to maintain a partition of items.

- (a) (6 pts) Explain how to construct a Union-Find DS using doubly linked lists with the following runtimes:
  - `make_set`:  $O(1)$ .
  - `union(A,B)`:  $O(\min(n_A, n_B))$ , where  $n_A$  is the number of elements in  $A$  and  $n_B$  is the number of elements in  $B$ .
  - `find(i)`:  $O(1)$ .

Make sure to clearly explain how the operations work and analyze the runtime. **Hint:** You may want to modify the list's node class to contain one extra member.

- (b) (6 pts) We can improve the runtime of `union` by having slower `find` operations. Explain how to construct a Union-Find DS using a tree of elements with the following runtimes:

- `make_set`:  $O(1)$ .
- `union(A,B)`:  $O(1)$ .
- `find(i)`:  $O(\log n)$ .

Make sure to clearly explain how the operations work and analyze the runtime. **Hint:** Your tree does not have to be a binary tree. That is, a node can have more than two children.