Problem 4 is worth 40% of this problem set, and problems 1-3 constitute the remaining 60%.

For each problem where you are asked to give an algorithm, more points are given for asymptotically faster algorithms. In judging the number of points to award a correct solution, we only consider the running time in asymptotic notation, i.e. a writeup of an algorithm taking 1000n^2 steps versus one taking .01n^2 steps would receive the same number of points — both would be simply treated as \( \Theta(n^2) \)-time solutions.

Problem 1

In class we saw that with the disjoint forests data structure, if you use both “union by rank” and “path compression”, a sequence of \( n \) make-set and \( m \geq n \) find and union operations takes time \( O(m \log^* n) \). As you probably noticed, the proof for that bound is non-trivial, which might leave you wondering, could I possibly simplify the data structure and still achieve this bound? Along that train of thought, suppose we use only one of “union by rank” or “path compression”, but not both. Show that if we only use union by rank, there is an infinite family of operation sequences (with \( m, n \) going to \( \infty \)) such that the total runtime to serve a sequence in this family is \( \Omega(m \log n) \). Similarly, show that if we only use path compression, we can also make the total time \( \Omega(m \log n) \). In both of your example sequences, \( m \) should be \( \Theta(n) \). **Hint:** you may find the following sequence of trees useful to think about:
Problem 2

You know what can get tough when running an automated factory? Checking that each robot is up and running! Suppose each robot has a set schedule on which it runs. For simplicity, a robot can only run once and must run continuously until some end time. We can then represent such a schedule as \( n \) horizontal line segments in the plane. The \( i \)th segment has some height \( h_i \) (which may be negative and represents the equity on robot \( i \)) and runs from the start time, \( x = a_i \), to the end time, \( x = b_i \) (\( a_i < b_i \)). For some bizarre reason, the line segments are half-open: they contain their end time, but not the start time. For simplicity, “checking in” at the factory can be represented by a vertical line (note line and not line segment), and can occur at non-integer times. The only limitation that exists is that each robot (horizontal line segment) is checked on (intersected by a vertical line) at least once.

(a) Help the manager solve this problem! Give a greedy algorithm which minimizes the total number of “check-ins” and prove its correctness. The running time should be \( O(n \log n) \).

(b) However, the robots actually form part of a union, and are rather wary of being constantly checked on. Suppose the objective was instead to minimize the maximum number of times that any robot is checked on. That is, each robot should be checked on at least once, and at most \( z \) times such that \( z \) is minimized. Show that if the optimal solution for a given instance achieves a value of \( z_{opt} \), then there is a greedy solution which is optimal for part (a) while achieving \( z \leq z_{opt} + 1 \) for this new objective.

(c) Well, the robots really do insist that you try your best to absolutely achieve \( z_{opt} \). Assuming that the statement of (b) is true, derive an algorithm which achieves the objective of (b) but with \( z = z_{opt} \). Any algorithm running in polynomial time (i.e. \( O(n^c) \) for some constant \( c \)) will receive full credit because the robots are on the verge of mutiny.

Problem 3

You are given a rooted tree on \( n \) nodes as input, where the root is vertex 1. The input is specified as follows: you are given an array \( A[1..n] \) such that \( A[i] \) is a linked list containing the children of vertex \( i \). Each vertex in this tree corresponds to a computational task that must be carried out by one of your servers. However, there is a constraint that a vertex cannot be processed by a server until all its children have already been processed. You have \( k \) servers available to do the processing, meaning that you can only process up to \( k \) tasks in parallel. Every task takes one unit of time to complete. Your goal is to devise a greedy algorithm to compute the optimal time needed to complete the root task of the tree.

For example in Figure 1 with \( k = 2 \), a valid ordering is to process vertices 2 and 5, then 6 and 7, then 8 and 9, then 3 and 4, then 1, taking 5 units of time. This is optimal.
Figure 1: Example input to Problem 3.

(a) (4 points) Show that if the number of leaves is at most $k$, then the minimum time needed to complete all tasks is equal to the height of the tree plus 1 and give a greedy algorithm to calculate an optimal scheduling of the tasks. **Note:** we interpret a tree with a single node as having height 0.

(b) (3 points) Consider the general case, where the number of leaves may be larger than $k$. Let $H$ be the height of the tree, and show that for any $0 \leq h \leq H$, $h + 1 + \lceil \frac{n_h}{k} \rceil$ is a lower bound on the processing time for any schedule of processing tasks. Here $n_h$ denotes the number of nodes at depth $h + 1$ or deeper in the tree, where the “depth” of a node is its distance to the root.

(c) (3 points) Devise a greedy algorithm to find an optimal scheduling of tasks even in the general case where the number of leaves may be larger than $k$. In your correctness proof, you may wish to show that your algorithm achieves processing time matching the lower bound from (b) for some $h$.

**Programming Problem**

Solve **ODDEMNITY** on the programming server ([https://cs124.seas.harvard.edu](https://cs124.seas.harvard.edu)). **Hint:** Suppose we say $u \sim v$ if vertex $u$ has an even length path to vertex $v$. Then note $\sim$ is an equivalence relation for undirected graphs, i.e. it provides a partition of the vertices into disjoint equivalence classes ($u, v$ are in the same partition iff $u \sim v$). Another definition and fact about undirected graphs that may be useful for you to know: define a graph to be **bipartite** if its vertices can be labeled “white” or “black” (or “0” or “1”) such that for any two vertices $u, v$, if $(u, v)$ is an edge then $u, v$ have different colors. Then it is a known theorem that an undirected graph $G$ is bipartite iff $G$ has no odd-length cycles.