This homework is due Wednesday, April 28, at 3:30pm MST. Please upload a single PDF file containing your submission (ensuring scans of handwritten work are legible) on Gradescope by then.

The questions are drawn from the material in the lectures (and Chapter 19 of the text) on Fibonacci heaps, and graph matchings.

The homework is worth a total of 100 points. When point breakdowns are not given for the parts of a problem, each part has equal weight.

In grading the required problems on the homework, only a subset of two required problems will be graded, whose points together add up to a total of 100 points.

Please remember to start each problem on a new page, and mark the corresponding pages in your submission for each homework problem on Gradescope. Conciseness counts!

(1) **(Lower-bounding heap operations)** (50 points) Suppose only comparisons are allowed on heap keys. Prove that

Insert or (Extract and (Delete or Minimum))

must take $\Omega(\log n)$ amortized time on a heap of $n$ elements for all heap implementations.

(Note: This shows the amortized times for Fibonacci heaps are optimal for comparison-based heaps. In particular, it shows that Insert and Extract cannot both take $O(1)$ amortized time.)

(Hint: Make use of the classic lower bound on the worst-case time for sorting by comparisons. This lower bound is given in Section 8.1 of the text.)

(2) **(Structure of Fibonacci heaps)** (50 points) Prove the following.

Lemma  For any Fibonacci heap $H$ and any $k \geq 0$, one can construct $H$ so that the Fibonacci tree $T_k$ is rooted at any specified node of $H$ of degree $k$.

(Note: This shows that the analysis from class of the maximum degree in a Fibonacci heap is tight.)

(Hint: First show how to construct $T_k$ for any $k \geq 0$ by a series of Fibonacci heap operations.)

(3) **(Fibonacci heaps on a pointer machine)** (50 points) The implementation of the Extract operation for Fibonacci heaps uses array indexing to efficiently find nodes of equal degree when consolidating roots. A pointer machine is a restricted model of computation equivalent to a random access machine without arrays. The memory on a pointer machine may hold data structures that contain pointers, but array indexing is not allowed.

Show how to implement a Fibonacci heap on a pointer machine so all its operations, except for $\text{Union}$, run in the same amortized time as on a random access machine. In particular, show how to implement Extract without using an array so it still runs in $O(\log n)$ amortized time.

(Note: When you implement Extract without an array, you may need to add new pointer fields to the Fibonacci heap data structure. If other operations besides Extract have to maintain these additional pointers, you will need to discuss how the implementation of the other operations changes as well. Since the solution modifies the data structure, you may need to modify the potential function as well to achieve the same amortized times for all operations. A consequence of your solution to this problem should be that the implementation of a Fibonacci heap does not need to know $f(n)$, the maximum degree in a heap of size $n$.)
(4) **Path and cycle cover** (50 points) Given a directed graph \( G = (V, E) \), a *path and cycle cover* of \( G \) is an edge subset \( C \subseteq E \) such that the subgraph \( (V, C) \) consists of vertex-disjoint directed paths and directed cycles. In other words, in \( C \), every vertex has both in-degree and out-degree at most 1.

Given a directed graph \( G \) with edge weights \( \omega \), design an algorithm that finds a path and cycle cover \( C \) of maximum total weight \( \sum_{e \in C} \omega(e) \) in \( O(mn + n^2 \log n) \) time, where \( n \) and \( m \) are the number of vertices and edges of \( G \).

(Hint: Reduce the problem of computing a maximum weight path and cycle cover to the problem of computing a maximum weight matching in a bipartite graph.)

(Note: If graph \( G \) is acyclic, then an algorithm for path and cycle cover will find an optimal covering of \( G \) by disjoint paths.)