Guidelines: You may brainstorm with others, but please write up the answers by yourself. Acknowledge all collaborations and external resources used.

1. Linked list heaps.
   (a) Describe how to use a linked list to implement a heap that supports delete, find-min, and delete-min in $O(1)$ worst-case time, and insert and decrease-key in $O(n)$ worst-case time.
   (b) Compared to a Fibonacci heap, a linked list heap supports faster delete-min but slower inserts. Are there any sequences of operations where the benefits of a faster delete-min ($O(1)$) outweighs the disadvantages of a slower insert ($O(n)$)? Why or why not? Explain.

2. A min-heap is a heap that supports the find-min and delete-min operations. A max-heap is a heap that supports the find-max and delete-max operations instead. A min-max-heap supports both.
   (a) Describe how to implement a min-max-heap, which supports insert, meld, find-min, and find-max in $O(1)$ time, and both delete-min and delete-max in $O(\log n)$ amortized time. Base your data structure on Fibonacci heaps. Please describe all node fields and all operations.
   (b) Explain the why your operations run in the time specified. To make your arguments more concise, you may assume the results on Fibonacci heaps that we proved in class.
   (c) Can you implement decrease-key in $O(1)$ amortized time? Why or why not? Explain.

3. We can construct many interesting variants of the Fibonacci sequence by changing the initial values, $F_0$ and $F_1$. Specifically, consider the following sequence:

\[
\begin{align*}
G_0 &= a \\
G_1 &= b \\
G_k &= G_{k-1} + G_{k-2}
\end{align*}
\]

where $a$ and $b$ are positive constants. Show that for any $a, b > 0$, $G_k$ is $\Omega(\phi^k)$.

NOTE 1: $a$ and $b$ are non-negative, but they do not have to be integers.

NOTE 2: Although $O()$ and $\Omega()$ notation is often used to analyze the time and space complexity of algorithms and data structures, here you are only asked to analyze the growth of the function $G_k$. No algorithms, just math.

4. (Grads only) In the Fibonacci heap paper, on page 606-607, Fredman and Tarjan describe a variant of Fibonacci heaps that supports a kind of lazy deletion with “good trees” and “bad trees.” This allows delete and delete-min to take amortized time $O(\log(n/l) + 1)$ for a sequence of $l$ operations without a find-min. Prove that this bound is correct.

NOTE: The description is somewhat confusing. My interpretation is that delete min always begins with a call to find min (forcing a consolidate, if necessary), and that the minimum pointer is set to null after the minimum element is deleted.