These problems are recommended. Do not turn them in. Solutions will be posted (except for the first one.)

1. (Note: the purpose of this problem is to help you make sure that you understand the mechanics of the various data structures listed below. If you are convinced you understand how they work you can skip this problem, but be aware that you might get some version of this problem on the midterm and/or the final exam.)

Choose a few small integers (somewhere between 5 and 10). For each of the following data structures, insert your numbers into the data structure and then work through the various operations.

(a) binary heaps. Find-delete min, delete, change-priority. Also be sure that you understand how heapify works.

(b) k-ary heaps. Find-delete min, delete, change-priority.

(c) Fibonacci heaps. Find-delete min, change-priority. Also be sure that you understand the potential function. (I won’t ask you to reproduce the analysis for find-delete-min and change-priority, but you should be able to tell me the value of the potential function for a Fibonacci heap in a given state.)

2. Suppose that we have a binary heap with \( n \) numbers in it, with smaller numbers meaning higher priority. We know that the minimum of the numbers is in position 0 of the heap. How many different positions could possibly hold the maximum of the numbers? (You should assume that all the numbers are different from each other.)

3. Consider the following naive representation of a priority queue: we represent the queue as a dynamic array of the data values, with no constraints on how they are ordered. To find the minimum value, we search through the whole array.

(a) What are the amortized time bounds for the add, decrease priority, and delete min operations in this structure? Give your answers in O-notation as a function of \( n \), the number of values in the priority queue.

(b) If we are using a priority queue to implement Dijkstra’s shortest path algorithms, are there graphs for which this naive method would lead to a total time bound that is the same as or better than the bound given by using Fibonacci heaps? If yes, describe these graphs. If no, explain why not.
4. (In each of the parts of this question, you do not need to specify the priorities of the items stored in the tree or forest. Just draw the shape of the tree or the shapes of all the trees in the forest.)

(a) Draw an example of a tree that cannot be formed as part of a Fibonacci heap.

(b) Draw an example of a forest that can be a Fibonacci heap but cannot be a Fibonacci heap immediately after an insert operation.

(c) Draw an example of a forest that can be a Fibonacci heap but cannot be a Fibonacci heap immediately after a delete min operation.

5. Suppose that in the implementation of the delete min operation on Fibonacci heaps, we eliminate the step where we make all root nodes have a different number of children. (So all we do is delete the node pointed to by the min pointer, add all of its children to the list of root nodes, and traverse the list of root nodes to update the min pointer.) Show that in this modified implementation of the Fibonacci heap, the amortized cost of the delete min operation is no longer $O(\log n)$.

6. Suppose that we perform the following two operations on a Fibonacci heap:

- First, create a new heap from a given set of $n$ values; and
- Second, perform a single delete-min operation.

When we are finished, how many trees will there be in the structure as a function of $n$? (Hint: First consider the special cases where $n$ is a power of 2 and one more than a power of 2. Then for general $n$, think about binary representations of integers.)