CIS 413/513 Midterm
DUE: 10:00am on Wednesday, November 2nd

Instructions: You must complete this exam independently, with no outside resources of any kind except for: the Tarjan book, CLRS, the splay tree article, the “Scapegoat and Splay Trees” notes referenced in the last homework, and any notes you took in class. You may not discuss the questions with other students in the class. If you find a question confusing or suspect that it is inaccurate, please contact me ASAP by email.

1. For the following questions, we define the depth of a Fibonacci heap as the maximum depth of any tree in the root list. A tree that is a single node has a depth of 1. Operations in a Fibonacci heap may increase or decrease this maximum depth.

   (a) (15 points) What is the largest increase in the depth from a single operation in a Fibonacci heap with \( n \) nodes, and under what circumstances does it occur?
   
   (b) (15 points) What is the largest decrease in the depth from a single operation in a Fibonacci heap with \( n \) nodes, and under what circumstances does it occur?

Additional Details: Select the operation and initial structure that give you the largest increase or decrease in depth. Please specify:

- the operation you are performing and which node you are performing it on
- the structure of the heap before and after the operation
- the exact change in depth, as a function of \( n \).

The structure you describe must be possible to construct using normal Fibonacci heap operations, but you do not need to describe how to construct it.

To be perfectly clear on the expectations, here is an example of how to answer a question like this using a binary heap:

Q: What is the largest increase in the depth of a binary heap?

A: The largest increase in depth in a binary heap with \( n \) items is obtained by inserting a single item, which increases the depth by 0 or 1, depending on \( n \). When \( n = 2^k - 1 \) (for some \( k \)) the lowest level of the binary heap is full and it will create a new level with one node in it. This increases the maximum depth by one. Otherwise, inserting an item will not increase the maximum depth – it will only add one node to the lowest level.

2. Suppose a crazy hacker releases an internet worm that corrupts all splay tree implementations whenever it infects a computer. In the infected implementation, the code performs a splay operation on a node of its choosing before performing any requested operation, in an attempt to slow down your computer.

   (a) (10 points) Explain why a sequence of \( m \) splay tree accesses that we request can still be performed in \( O(m \log n) \) amortized time, where \( n \) is the total number of nodes in the splay tree, no matter which nodes the infected code chooses to splay. (NOTE: \( O(m \log n) \) is only the amortized time, not the total worst-case time, since we’re ignoring a possible decrease in the potential function. You only need to worried about the amortized time, not the total worst-case time.)
(b) (10 points) Suppose that instead of performing an extra splay operation, the hacked code performs a modified locate, which finds a node but does not splay it to the root. (The locate commands that we issue are splayed as normal.) Including the running time of the adversary’s operations, what is the new asymptotic running time of \( m \) operations? Use a worst-case sequence of operations. Justify your answer.

3. (10 points) (Take off every zig!) Let a semi-splay be a splay-like operation that moves a node up in the tree via “zig-zig” or “zig-zag” operations until it has no grandparent. This could either happen when the node becomes the root or when it becomes the child of the root. In other words, this is just like a regular splay, except without the “zig” operation.

Suppose that we redefine the find operation to perform a semi-splay operation instead of a splay operation before returning the desired node \( x \). This means that, after performing a find, the accessed node could either be the root of the tree or a child of the root.

Prove that the amortized cost of a single find operation is still \( O(\log n) \). Use the potential function from class: \( \sum x \lg(size(x)) \), where all nodes have weight 1. (Note that we are only discussing amortized time, which ignores possible drops in the potential function. You do not need to address drops in the potential function in your answer.)

(HINT: It is possible to answer this question in three sentences. Longer answers are fine, but if your answer is getting excessively long and complicated, then you may want to search for an easier way.)

4. Consider the following lazy variant of a binary heap. The lazy binary heap consists of a heap list, an insert list, and a pointer to the minimum element. The insert operation appends the item to be inserted to the insert list and updates the min pointer, if necessary. The delete-min operation builds a new binary heap from the items in the insert list and appends this heap to the heap list. Then it removes the element referenced by the min pointer from whichever binary heap it’s in using the standard delete-min operation on binary heaps. Finally, it updates the min pointer by looking at the roots of all heaps in the heap list (since the new minimum could be in any of these heaps).

(a) (10 points) Suppose that the lazy binary heap currently contains \( n \) elements and delete-min has already been called \( k \) times. What is the worst-case running time of the next insert or delete-min operation, as a function of \( n \) and \( k \)? Justify your answers.

(b) (10 points) What is the amortized running time of insert and delete-min as a function of \( n \) and \( k \)? Justify your answers.

(c) (20 points for grads, 5 points extra credit for undergrads) Suppose you have an algorithm that requires \( n \) inserts and \( m \) delete-mins. For what values of \( m \), relative to \( n \), will the lazy binary heap be asymptotically faster than the traditional binary heap? Assume a worst-case sequence of inserts and delete-mins in your analysis. Express your answer as \( m = o(f(n)) \) or \( m = \omega(f(n)) \) for some function \( f \) that you determine. Justify your answers.