Individual Assessment 4: Amortized Efficiency

This is an individual assessment, and, as the name suggests, must be completed individually. Specifically, you're not allowed to work with a partner, and you should not discuss these problems with other students in CS166. However, the course staff are happy to answer clarifying questions on Piazza (if you do, please post the question privately) or in our office hours.

Due Tuesday, May 19th at 2:30PM Pacific.
Problem One: Palos Altos

The order of a tree in a Fibonacci heap establishes a lower bound on the number of nodes in that tree (a tree of order $k$ must have at least $F_{k+2}$ nodes in it). Surprisingly, the order of a tree in a Fibonacci heap does not provide an upper bound on how many nodes are in that tree.

Prove that for any natural numbers $k > 0$ and $m$, there’s a sequence of operations that can be performed on an empty Fibonacci heap that results in the heap containing a tree of order $k$ with at least $m$ nodes (that is, the tree of order $k$ contains at least $m$ nodes). Illustrate the sequence of operations you get with $k = 2$ and $m = 8$ and show the resulting tree. You don’t need to draw out each step, but should provide enough detail to convince us (and yourself!) that your sequence indeed produces the correct tree.

Problem Two: Building B-Trees

Suppose you have a sorted sequence of keys $x_1 < x_2 < \ldots < x_n$ from which you’d like to construct a B-tree of order $b$. To do so, you could insert each key into an empty tree one at a time. If you were to do this, though, you’d find that you were doing a lot of unnecessary work traversing the tree top-down, since each search is guaranteed to terminate in the rightmost leaf. (Make sure you see why this is, but you don’t need to prove this.) This top-down searching would mean that the runtime of inserting the keys would be $\Omega(n \log_b n)$.

There’s a much faster way to build a B-tree from scratch. Maintain a pointer to the rightmost leaf node in the B-tree, and annotate each node in the B-tree with the number of keys it contains. Then, insert each key, one at a time and in sorted order, using the following procedure:

1. Place the key in the next free slot in the rightmost leaf.
2. If the rightmost leaf overflows, split that leaf and propagate any further splits higher up in the tree using the usual node-splitting procedure.

Prove that the amortized cost of inserting each element this way is $O(1)$ in two ways, first using the banker’s method and second using the potential method. This shows that the cost of building a B-tree from a sorted sequence using this algorithm is $O(n)$, independent of the order of the B-tree.

For the banker’s method, if an operation places down credits, tell us where those credits are placed, and if an operation spends credits, tell us which specific credits it spends and why the credits you’re spending are guaranteed to be there. For the potential method, clearly define what your potential $\Phi$ is in terms of the shape of the B-tree and explain why your potential begins at zero and is nonnegative.

Although B-trees are typically stored on-disk and analyzed in terms of disk reads and writes, for the purposes of this problem please analyze this algorithm in terms of the total number of operations performed, not the number of block transfers.