CSCI 104L Lecture 21: Log-Structured Merge Trees

Consider the following data structure:

- You have a linked list of *arrays of* integers.
  - The first node has an array of size one.
  - The second node has an array of size two.
  - The third node has an array of size *four*
  - The fourth node has an array of size *eight*
  - The ith node has an array of size 2^{i-1}
- Each array of integers is *sorted.*
- We maintain the invariant that each array is either full or empty.

Question 1. If we have 7 elements, which nodes will be full?

Question 2. If we have 12 elements, which nodes will be full?

Question 3. How would you write Find for this data structure?

Question 4. What values of n (number of elements) produce the worst-case runtime for Find?

Question 5. What is the worst-case runtime for Find?

Question 6. If the first 3 nodes are full, and the 4th node is empty, what will be the state of the first 4 nodes after we insert something into the data structure? Which values will be in the 4th node?

Insert

1. Try to insert the new value into the first node. If it is currently empty, you're done.
2. Otherwise, merge the contents of the first node with the new value to produce a sorted array of size 2.
3. If the next node is empty, place the array in the node, and you're done.
4. Otherwise, merge the contents of the current node with your array to produce a sorted array of double the size, and return to the previous step.

Question 7. What values of n (number of elements) produce the worst-case runtime for Insert?

Question 8. What is the worst-case runtime for Insert?

Question 9. What is the worst-case **amortized** runtime for Insert?

Question 10. Would this amortized runtime still hold if you were allowed to Remove from the data structure as well?