Discussion Prompt Questions
Module 6
Brian Loughran
Johns Hopkins Data Structures

1. How is a priority queue different from a traditional queue? What applications would make the best use of such a data structure?
2. Is there an implicit prioritization in an traditional queue? If so, what is it?
3. What data structures lend themselves to effective implementation of a priority queue?
4. How does the order of of a tree (a general tree with no special properties) affect the size of the tree and layout of the nodes in the tree. How do you decide which child path to follow?
5. How would a Huffman tree look different if we used different tie-breakers. and how would that impact the potential compression.

ANSWERS:

1. A priority queue is a data type where each element has an associated priority, or ranking. The queue is then ordered by that priority. This is different than a regular queue, since a regular queue is organized as a FIFO structure, where the first item out is the first item in. In a priority queue the first item out is the item with the highest priority, regardless of who was first. My high school accounting teacher used to make a joke when the physics teacher would be in front of him to use the printer. He would kid that the printers were prioritized by SAT score. This was not actually true, this comment was made in jest, but a printing queue prioritized by SAT score would be a good example of a priority queue. Those who had the highest score on the SATs would bet pushed to the top of the queue, and those with lower scores would get bumped to the back of the line.
2. There is implicit prioritization in a traditional queue. The first item in is the item which gets the highest priority. This is analogous to when you go to a busy restaurant, and you have to take a number. The highest priority is the number which was taken the longest time ago, or the first number.
3. A heap would lend itself to a priority queue quite well. The root of a max heap is always the highest value in the heap. So if you wanted to get the top priority item, it would take only O(1) to access, and O(lgn) to pop() the top priority and reorder the heap. Insertion into the heap is also efficient, O(lgn).
4. The order of a tree with no generalized properties does not tell us a whole lot about the tree. For example, the tree could be perfectly balanced, with n nodes and n/2 leaves. Or a tree could be perfectly imbalanced (for example, all nodes have a right child, but no left child. Then the tree would have n nodes and 1 leaf. Insert and delete would take much longer for imbalanced trees. You can decide which child node to follow based on the type of tree. If it is a binary tree, for example, if the value is higher than the current node, you could traverse the tree to the right. If the value is lower than the current node, you can traverse to the right. Different types of trees may have different strategies for searching a value. Some trees may not be efficient for traversing, for example, if you are trying to check if a max heap has a specific value. That operation would take O(n), which is not efficient.

Discussion Prompt Questions
Module 6
Brian Loughran
Johns Hopkins Data Structures

5. Using different tie breaker would change slightly the organization of the tree, and therefore the encoding of the letters. Some examples of different tie breakers can be alphabetic tie breakers, putting smaller nodes to the left, etc. This should theoretically not change the compression of the message in a large way, while you may see a small change in the size of the message, it should be pretty insignificant.