Kyle Neubarth

Data Structures Final Project: Priority Queues

Purpose:
Determine the efficiency of different data structures when implemented as a priority queue under different conditions, namely the number of nodes per queue.

Procedure:

Linked List:
A linked list is made up of nodes that contain a value and a pointer to the next node. The first node in the list is called the root. To enqueue a node, we start at the root and go down the list one by one. If the insertion node has greater priority than the node we are looking at, the insertion node replaces it and the replaced node becomes the next node after the insertion. Dequeuing is easy, as one can simply return the value of root, remove the root node and set the next node as root.

Enqueue: O(n)
Find Max:O(1)
Dequeue: O(1)

Binary Heap:
A Binary heap is an array based tree structure, we are using a max heap for our queue, meaning the root will always be the highest priority node. Lower nodes on the tree are not guaranteed to be ordered. To insert a node, add the node at the next open location, then compare the node with its parent. If the node has a higher priority, switch them. Continue this until the node stops. Viewing the highest priority node is easy, simply return the root value, but dequeuing is a bit more complex. The root node is replaced with the last node in the tree, and then the replacing node is compared with its children. If it is smaller than either, it switches places with the highest child. This repeats until the node stops moving.

Enqueue: O(log(n))
Find Max:O(1)
Dequeue: O(log(n))

Standard Library Priority Queue:
The Standard Library Priority Queue is implemented with a binary heap

Same big O notation as Binary Heap

Test results taken will be time to enqueue and dequeue an array of nodes. The times will be given in milliseconds as an average of all tests of the given type. Standard deviation of each set

of tests will also be recorded. Tests will be run 2000 times each for every implementation for different numbers of nodes(0-880 in intervals of 50).
Data:

The data set used is an array of patients. Each patient has a name, priority, and an expected treatment time. Patients with higher priority will be put higher in the queue, and in case of a tie, the patient with the lowest treatment time will go first.

Results:

(Look in 'Graphs' folder for data plots)

Linked lists are very slow when enqueueing compared to binary heaps, but have extremely fast dequeuing. This can be explained by the big O notation of our linked list. Enqueuing a node has a complexity of O(n), and dequeuing always will be O(1) where n is the length of the queue.

The time for enqueuing a linked list gets more and more predictable as we add more nodes. This explains the curve in the standard deviation of the enqueue times for the Linked List

The curve in standard deviation for enqueue times in a linked list can be explained by the distribution of data and the sorting method of the Linked List. The distribution is roughly bell shaped, so as the list has more and more elements the time taken to enqueue a new element will even out, as most likely it will be going to the center of the list.

Binary heaps are faster than linked lists for enqueueing but much slower for dequeuing. As their enqueuing and dequeuing functions have a complexity of up to O(log(n)).

The standard library priority queue has equal complexity to our Binary Heap. This is because the STD queue is implemented through a binary heap. The standard library implementation is also slower than our binary heap. I suspect this is because it uses vectors, which have to be resized frequently where we use a simple array. It also might have to do with our Binary Heap being very bare bones with minimal functions.

In summary, the Binary Heap implementation of a priority queue is always superior to the Linked List implementation, except for the fringe case of needing a queue with a small number of nodes. The Standard Library Priority Queue, while slower than our implementation of a Binary Heap, is still preferable to a Linked List given the required queue is large enough. (In our case <600 nodes)

Using a linear and logarithmic regression from the data obtained, constructing a Priority Queue and enqueuing all citizens in the United States would take:
202.4 seconds using a linked list implementation
3.848 seconds using a binary heap implementation