Kirk Saechao

Riggins

CS310

28 October 2014

Program #2 Report and Analysis

**1. Ordered Array**

*Insert:* For insertion into an ordered array, we cannot simply tack the new addition onto the end. Order matters, so we must find the correct location for insertion. We use binary search (O (log n)) to find the insertion point and then we must shift every element from that position to the right (O(n)) to make room. In the end, we have **O(log n) + O(n); which resolves to O(n).**

*Remove:* For removal in an Ordered Array Priority Queue, we already know where the where the highest priority lies. Our insertion method stored the incoming objects in descending order. New objects of the same priority inserted to the array were stored to the left of currently existing objects of the same priority. This means the object of the highest priority that has been in the Queue the longest is located at index (currentSize-1). Because we know the exact location of the highest priority object, a removal of this object should be **O(1).**

**2. Unordered Array**

*Insert:* Insertion into an Unordered Array is simple; we simply add the newest object onto the end of the list. We do not calculate priority while during the insertion. Calculation of priority comes during the removal method. The complexity of this algorithm should be **O(1).**

*Remove:* Removal of an object in an Unordered Array Priority Queue requires you to go through every element in that array. You will not know an object is of the highest priority unless you compare it to every single element. First we set a temp variable to the first element in the array, and compare to every other element. This takes time O(n). After we find the element that we want, we must shift every element thereafter down by a slot. This also requires time O(n) in the worst case. In the end, we have O(n) for the comparisons plus O(n) for the shifts. **O(n) + O (n) = O(n).**

**3. Ordered Linked List**

*Insert:* Similar to insertion into an Ordered Array, order matters. We must compare the inserted object to (at the worst case) every element in the linked list. This results in a complexity of (at worst, but not often) **O(n).** Unlike an ordered array, shifting is not necessary.

*Remove:* Removal of the highest priority object in an Ordered Linked List is simple. Our insertion algorithm has automatically put that object in the first node of the linked list. To remove this object, simple pop it off the top **(O(1))** and remove the link.

**4. Unordered Linked List**

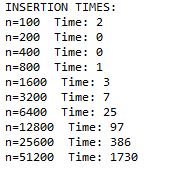
*Insert:* Insertion into an unordered linked list works much like an unordered array. We simply create a new node with the given data, and attach it to the end. Priority is not determined at this time. We simply attach to the end and move on **(O(1))**, no shifting necessary as usual for linked lists.

*Remove:* Removal of the highest priority object in an unordered linked list requires you to check every element in the list. We must set a variable equal to the head, and then compare that variable to every node in the list; while retaining the lowest value inside of our variable along the way. This behavior is equal to **(O(n)).**

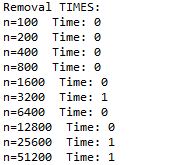
**Graphs**

**Ordered Array**

*Insertion*

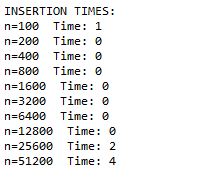
****

*Removal*

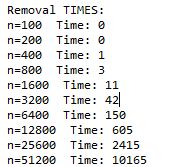
****

**Unordered Array**

*Insertion*

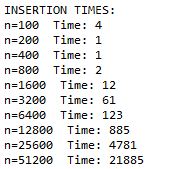
**

*Removal*

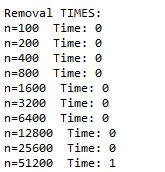
**

**Ordered Linked List**

*Insertion*

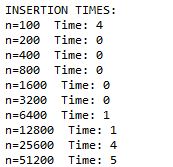
**

*Removal*

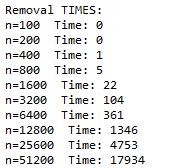
**

**Unordered Linked List**

*Insertion*

**

*Removal*

**

**Conclusion**

Overall, I believe that the Ordered Linked List implements a priority queue the most effectively. The main advantage of a linked list over an array is that a linked list cannot be filled up unless the computer runs out of memory; therefore an insertion will never fail. This makes it dynamic and able to handle any situation. In addition, removal times are O(1) which is very fast. The only disadvantage is that it takes O(n) to insert an element. However, I would still choose an Ordered Linked List over an Unordered Link List because it displays the data more neatly and you can easily tell how many elements of the same priority that a list contains.