CSCE 221 Cover Page

Programming Assignment #2

First Name: Anthony Last Name: Shao UIN: 725004846

**Any assignment turned in without a fully completed cover page will receive ZERO POINTS.**

Please list all below all sources (people, books, webpages, etc.) consulted regarding this assignment:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CSCE 221 Students | Other People | Printed Material | Web Material (URL) | Other |
| Vishaal Makani |  |  | Stack Overflow | CSCE 221 PowerPoint Slides |
| Suvedh Srikanth (former CSCE 221 student) |  |  |  |
| Hongyi Zhang |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Recall that University Regulations, Section 42, define scholastic dishonesty to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. Disciplinary actions range from grade penalties to expulsion. Please consult the Aggie Honor System Office for additional information regarding academic misconduct – it is your responsibility to understand what constitutes academic misconduct and to ensure that you do not commit it.

I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received nor given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment.

Today’s Date: 4/19/2018

Printed Name (in lieu of a signature): **Anthony Shao**

**Introduction**

The objective of this assignment is to implement three ways of a priority queue. First, one way is to implement a linked list that is not sorted. Second, another way is to implement a linked list that is sorted by order, decreasing in priority (smallest key at the front of the linked list). Third is to implement an array-based heap.

**Theoretical Analysis**

The first and second strategies are implementing linked lists. For the first implementation (unsorted linked list), inserting an element into the list takes O(1) time cost because no comparisons is need as the list is not sorted. Removing an element (the minimum or the “highest” key) from the list takes O(n) time cost because n comparisons are made to find the smallest key as the list is not sorted by order. For the second implementation (sorted linked list), inserting an element into the list takes O(n) time cost because it has to iterate through n nodes to sort by decreasing priority. Removing an element with the smallest key from the list takes O(1) time cost because the minimum is already at the head of the priority queue where the queue is implemented with the smallest key to be at the head.

The third implementation uses an array-based implementation. For this implementation, the data structure represents a heap. Inserting an element into the heap takes O(log n) time cost because the height of the tree is log n with n nodes already in the heap as the element moves its way down until conditions are satisfied. Removing the smallest key from the heap, which is at the top of the tree (first index of the array), takes O(log n) time cost as the last element replaces the minimum key and moves its way down until the heap tree conditions are satisfied.

**Experimental Setup**

Regarding machine specification, I used a Lenovo Yoga Pad 2 with Windows 10 running Microsoft’s Visual Studio 2017 and Window’s Command Prompt to implement, code, and run the three types of priority queues for this assignment.

Inputs into the queues were read from a text file called “numbers.txt”. The first number is read as the choice of priority queue to use (“0” for unsorted linked list priority queue, “1” for sorted linked list priority queue, and “2” for heap priority queue). The second number is read as how many numbers to inserts into the priority queue. Random integers are taken a random number generator found online, copied and pasted onto “numbers.txt” file. Random integers are between 0 and 100000. A for loop was used to call the insert function to insert the integers into the priority queue. Test sizes were in degrees of 10 (1 thousand and 10 thousand). This was to test the efficiency of each algorithm over longer periods of time. The initial priority queue size for each implementation was chosen to be 100000. The data structure used to implement the list is are singly-linked lists.

Each experiment was tested at least 10 times. However, most of my experiment time was spent debugging and implementations.

**Experimental Results**

In the long-run, the best performing insert operation of the three priority queue implementations is the array-based heap by rank. Second is unsorted linked list implementation. And last is sorted linked list implementation. The heap algorithm is the fastest as the inserts took 0.046875 seconds to insert everything. Next, unsorted linked list took 32.3324 seconds, which was faster than the sorted link list that took 44.8868 seconds. In most of the experiments, the long-run ranks are used as presented in the results. The theoretical analysis of *only* the insert algorithms of all three priority queues strongly agrees with the experimental results.

For the unsorted linked list, my remove function was unable to be successfully implemented.

In the long-run, the best performing insert *and* remove operations of the three priority queue implementations is the array-based heap by rank. Second is the sorted linked list implementation (not shown since remove implementation for unsorted linked list was unsuccessful). And last *should be* unsorted linked list implementation. The heap algorithm is the fastest as the inserts took 0.078127 seconds to insert and remove everything. Next, sorted linked list took 46.7793 seconds. In most of the experiments, the long-run ranks are used as presented in the results. The theoretical analysis of the insert *and* remove algorithms of all three priority queues somewhat agrees with the experimental results.