Name: Carlos Salazar

Assignment: Final Project

TA: Abhidip Bhattacharyya

Date: 5/4/2018

**Purpose** - What is the purpose of this project? What are you evaluating?

In my opinion the purpose of this project is to implement software for a real world application. This was an example of the many reasons why there is software development to solve real world problems. Even we start our day with the application like Alarm Clock or social networking apps and emails in the morning. This project allowed us to understand the network that we all live under on a daily basis. As with most things in programming, there are many ways to solve a problem. To make things interesting, you decide to implement this priority queue using three different implementations and compare their respective performances in order to determine the best data structure for the job which we included Heap, Linked List, and STL queue.

**Procedure** - What data structures are you using for your implementations? This section should include a description of how each implementation works. Also, include what metric you will be using to measure the runtime differences between the implementations.

Within my implementation we would include the data structures: Min-Heap, Linked List, and Queue. A min binary heap is an efficient data structure based on a binary tree. A min binary heap can be used to find the C (where C <= n) smallest numbers out of n input numbers without sorting the entire input. However, we must understand that:

1.The right child of node in index i is: 2\*i+1

2.The left child of node in index i is: 2\*i

3.The parent of the node in index i is: ((i)/2)

Then we have linked list that we used for this project. There are two types of linked list; singly-linked list and doubly-linked list. In a singly-linked list, every element contains some data type and a link to the next node. On the other hand, every node in a doubly-linked list contains some data type, a link to the next node and a link to the previous node. In this project we implemented a single linked node which would allow us to sort the data from smallest to greatest sort.

Lastly, we have STL C++ priority queue which was offered within the library. Priority queues are designed such that the first element of the queue is the greatest of all elements in the queue. This was the fastest way to sort the data from either greatest to smallest or vice-versa.

In order to find the runtime between the implementations we would use the Clock() function which would allow us to find the difference in the runtime for the algorithm. However, in order to output a precise measurement we converted to a float which allows for more precision in the digits printed unlike an integer type. We could even use a Double type for more precision.

**Results** – Describe how each implementation performed. This section needs to have at least one graph showing the performance differences. You also need to include the mean and standard deviations of the runtimes for each implementation.

Using the clock runtimes for each implementation I was able to use Excel spreadsheet in order to analyze the data speed for each data structure. Visually it is clear that STL and HEAP are the more efficient performer.

Time complexity of heapify is O(Logn). Time complexity of create And BuildHeap() is O(n) and overall time complexity of Heap Sort is O(nLogn).

Time complexity of Priority Queue will be O(nlog(n))O(nlog⁡(n)), but if all elements are known in advanced and stored we can build the heap in O(n)O(n) time using the [heapify](https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/DemoHeapify.pdf) method.

Time complexity of Linked List Before deletion you need to search the linked list, which is O(n). Once you have searched the node to be deleted, you can delete in O(1) for both singly-linked list.

I would argue that STL should have been the fastest due to O(1) for all elements.



