CSCI 2270 Final Project Description

Sitong Lu

1) Purpose

The purpose of this project is to evaluate the time of dealing the same problem while using different approaches. We have so many ways of solving the same problem by using knowledge from Data Structures and the most important thing about data structure is the time the program had processed and the number of storage spaces the program had used during the whole process. We need to get all the different runtimes for different ways if we want to find out the fastest and the easiest way of dealing a problem.

2) Procedure

I am using three different types of data structures in this project: Linked List, Binary Heap, and STL Priority Queue.

Linked List:

First of all, I need to get the patient's name, priority time, and treatment time while reading the file lines by lines. After going through the first line (which is "patient, priority, treatment"), I will call a split function to separate the line into three important information: patient name, priority time, and treatment time.

After that, a node will be created by taking all the three information above. The node will also have a pointer node "next" which will be initialized to NULL. Now I need to check the root of the linked list (which will also be initialized to NULL). If the root is NULL (which means the list contains 0 node) then the node I am holding will be the root. Otherwise I will check the priority of my current with nodes in the list from the beginning to the end. After I got the first node in the list that has the same priority time with my current node using a while loop, the treatment time of those two nodes will be compared and the node with smaller treatment time will be placed in front of the other node. Then I end this loop and read the next line in the file.

If there is no node in the list which has the same priority time with my current node, the first node I got in the list which has larger priority time than the current node will be located. Now I just need to place my node in front of the located node in the list and end this loop.

If none of the situation above happened then I just need to put my node at the end of the linked list.

After reading the whole file (which means I already put all nodes into the linked list), I just need to print each node's name, priority time, and treatment

time in a line with its arrangement in the list lines by lines to check whether my list is having the right arrangement.

Binary Heap:

Create an array to store all the nodes (starts from its second space, which is array[1]). Read the file lines by lines and create nodes just like the process in Linked List part. Place the node at the end of the array and check its parent's (if the current node locates at i then the parent node locates at i/2) priority and treatment time. Exchange the two nodes if the current node's priority time is smaller than its parent's or the treatment time is smaller than its parent's while the priority time is the same. Keep checking and exchanging the current and its parent node until the current node has no parent node. Finally just use a while loop to keep deleting and printing the heap (using the delete heap algorithm, which is replacing the end node as the new heap and keep checking and exchanging its children's (located at 2*i and 2*i+1) priority and treatment node until the children reach the end of the array) in the array until there is no nodes in the array.

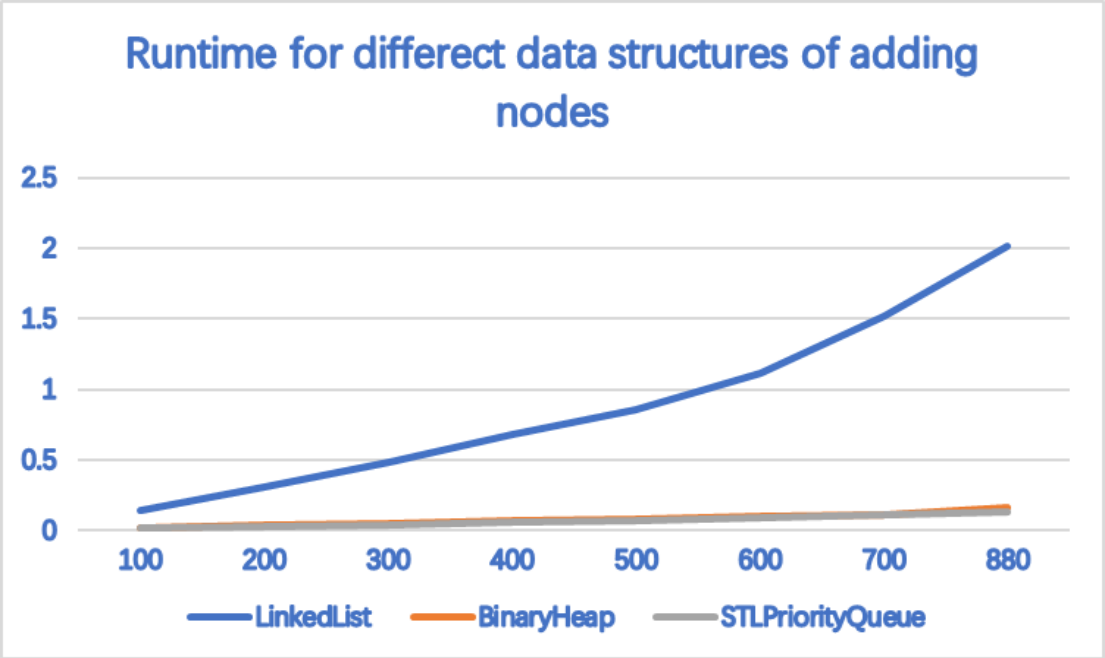
STL Priority Queue:

The whole process is super easy. I just need to create a compare function that will place the current node in the front of a queue if it has a smaller priority number or if it has a smaller treatment number (while comparing with other nodes with the same priority number). Then I create a priority queue using STL (with compare function that I had just defined above) and start that boring process (read file, split line into name, priority time, and treatment time. Put those information into a node and push the node into the priority queue). After reading the whole file I will get a perfect priority queue. Finally, I just need to print the queue's top out and pop the queue to get my final arrangement of the patient list on the screen.

3) Data

The data file contains three information: patient's name, patient's priority time, and patient's treatment time. To arrange all the patients in order, we need to first check patients' priority time. Patients whose having shorter priority time should be placed in the front of the list. If there are more than 2 patients whose holding the same priority time but with different treatment time then the patient with shorter treatment time will be in front of patients whose holding longer treatment time.

4) Results

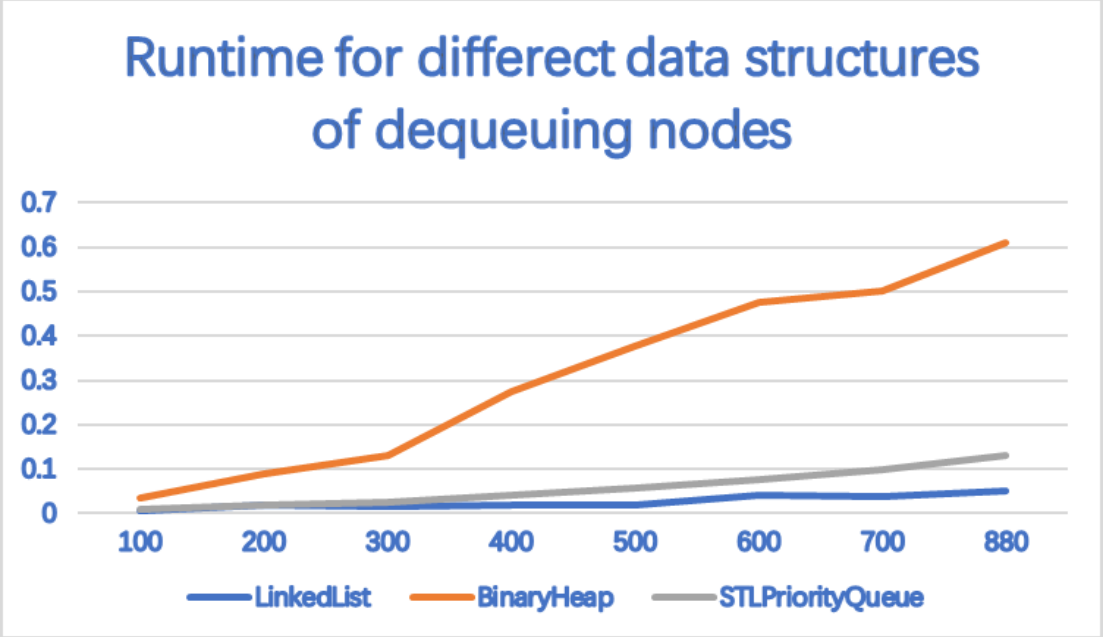


Standard Deviations:

Linked List: 0.635 ms

Binary Heap: 0.046 ms

STL Priority Queue: 0.04 ms

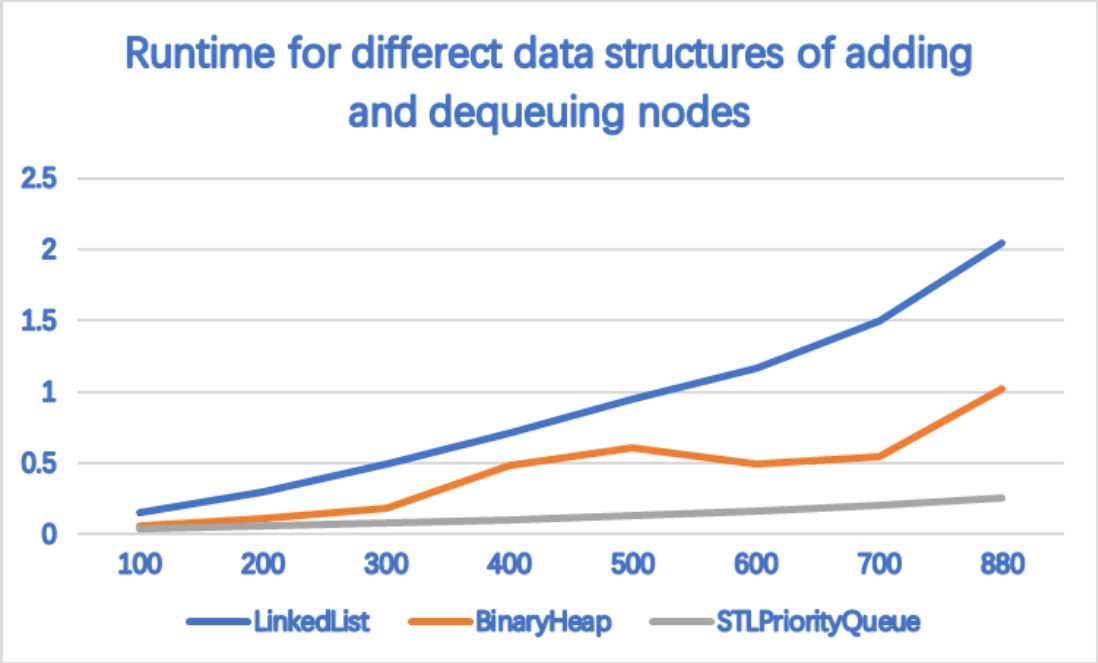


Standard Deviations:

Linked List: 0.015 ms

Binary Heap: 0.213 ms

STL Priority Queue: 0.042 ms



Standard Deviations:

Linked List: 0.638 ms

Binary Heap: 0.316 ms

STL Priority Queue: 0.074 ms

x-axis: first x lines of patients' information read from the file
y-axis: milliseconds

From the first graph, we can say that using linked list takes the longest time to get all the nodes added into the list but it takes the shortest time for binary heap (similar to STL Priority Queue) to just built the array. If the data is big enough (more than 1 billion lines of patients' data) then the binary heap will be easier than the linked list algorithm to build an array (based on the slope in the second graph). The linked list algorithm will be the fastest algorithm if we just want to dequeue all the nodes in the list (results from the second graph). If we count the total building and dequeuing time for each algorithm (data shown in the third graph) then the STL priority queue algorithm will be the fastest way to deal with this problem.

In all, the Linked List algorithm will be the worst choice to get the expected output; and the STL priority queue will be the fastest way to build the expected array. The binary heap algorithm works in a constant speed and its running time just lies between the linked list algorithm and the STL priority queue algorithm.