ALGORITHMS AND DATA STRUCTURE FINAL PROJECT REPORT

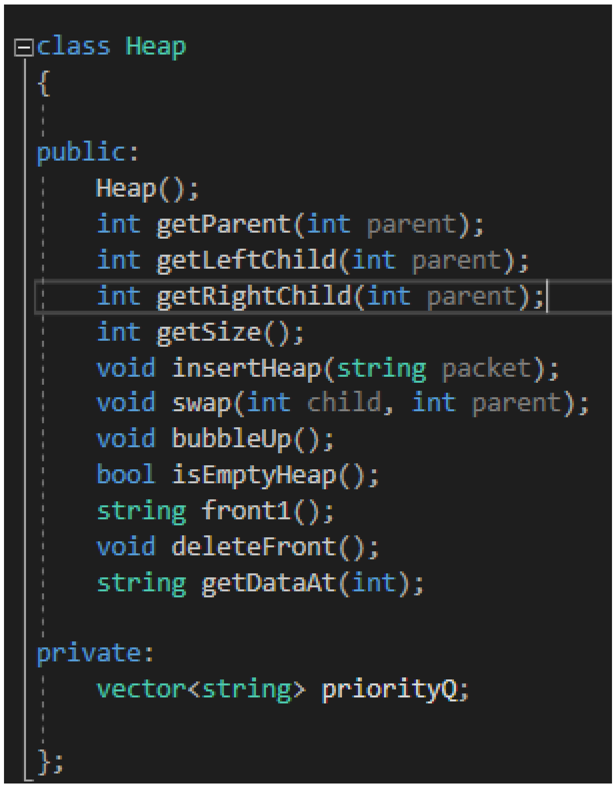
Rushi Patel Yash Patel Jeet Desai

Rushi Patel
Yash Patel
Jeet Desai

Algorithms and Data Structures Final Project Report

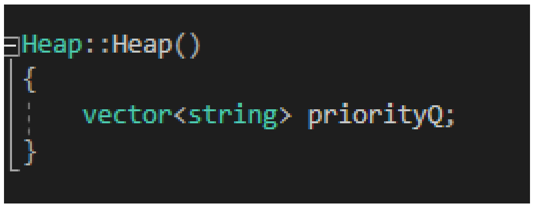
1. Implement the priority Q's using a heaps

Header class to define functions used to create a heap



```
class Heap
{

public:
    Heap();
    int getParent(int parent);
    int getLeftChild(int parent);
    int getRightChild(int parent);
    int getSize();
    void insertHeap(string packet);
    void swap(int child, int parent);
    void bubbleUp();
    bool isEmptyHeap();
    string front1();
    void deleteFront();
    string getDataAt(int);

private:
    vector<string> priorityQ;

};
```

Constructor which defines a vector which was used to hold all the packets from FIFO queue and sort them into priority queues 100, and 101 according to their priority

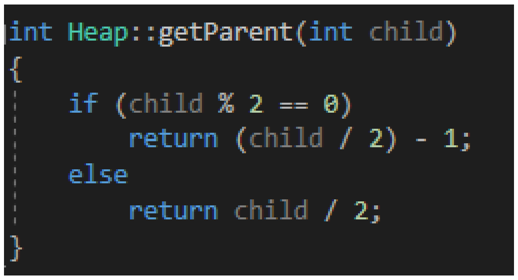


```
Heap::Heap()
{
    vector<string> priorityQ;
}
```

Rushi Patel
Yash Patel
Jeet Desai

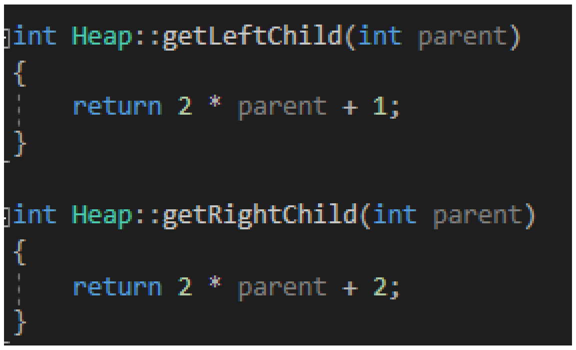
Algorithms and Data Structures Final Project Report

Parent function to get parent node from the tree



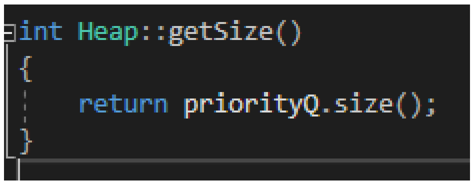
```
int Heap::getParent(int child)
{
    if (child % 2 == 0)
        return (child / 2) - 1;
    else
        return child / 2;
}
```

These functions are used to get left child and right child of a parent node



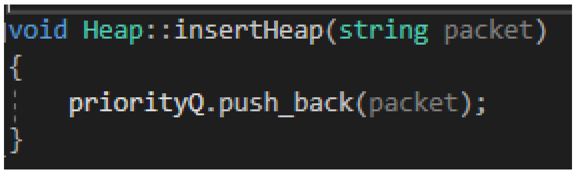
```
int Heap::getLeftChild(int parent)
{
    return 2 * parent + 1;
}

int Heap::getRightChild(int parent)
{
    return 2 * parent + 2;
}
```

getSize function returns priority queue size



```
int Heap::getSize()
{
    return priorityQ.size();
}
```

Insert function takes a packet as a parameter and pushes it back in the priority queue

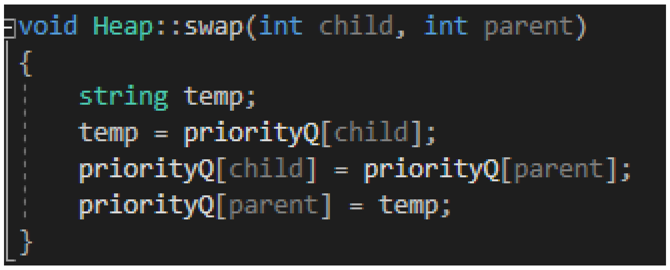


```
void Heap::insertHeap(string packet)
{
    priorityQ.push_back(packet);
}
```

Rushi Patel
Yash Patel
Jeet Desai

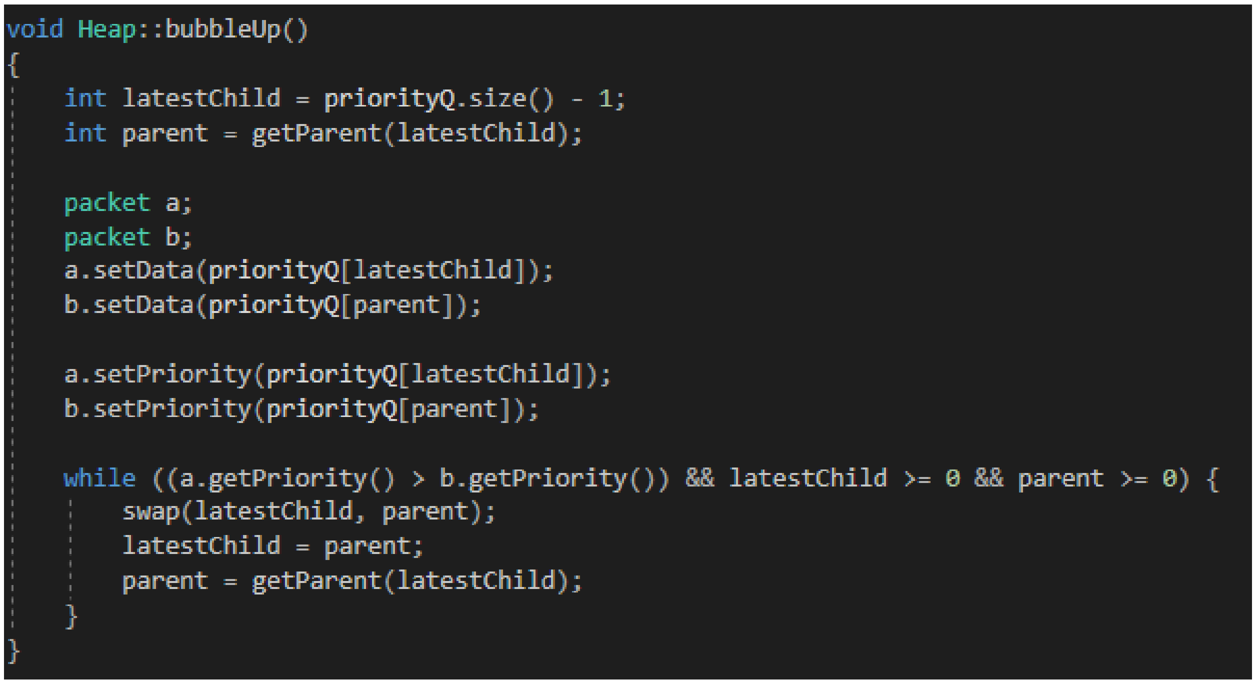
Algorithms and Data Structures Final Project Report

Swap function takes two parameters child and parent to swap parent with left or right child if any of them is greater than parent node.



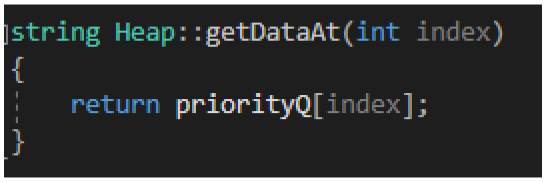
```
void Heap::swap(int child, int parent)
{
    string temp;
    temp = priorityQ[child];
    priorityQ[child] = priorityQ[parent];
    priorityQ[parent] = temp;
}
```

Bubbleup function takes latest child and keeps comparing with the parent node until latest child reach to the point where it is equal to the highest value or it is first in the queue. Packet with priority 2 will keep swapping until it is compared with a node which is holding a packet with priority 2.



```
void Heap::bubbleUp()
{
    int latestChild = priorityQ.size() - 1;
    int parent = getParent(latestChild);

    packet a;
    packet b;
    a.setData(priorityQ[latestChild]);
    b.setData(priorityQ[parent]);

    a.setPriority(priorityQ[latestChild]);
    b.setPriority(priorityQ[parent]);

    while ((a.getPriority() > b.getPriority()) && latestChild >= 0 && parent >= 0) {
        swap(latestChild, parent);
        latestChild = parent;
        parent = getParent(latestChild);
    }
}
```

This function takes in an index as a parameter and returns a value from that specific index



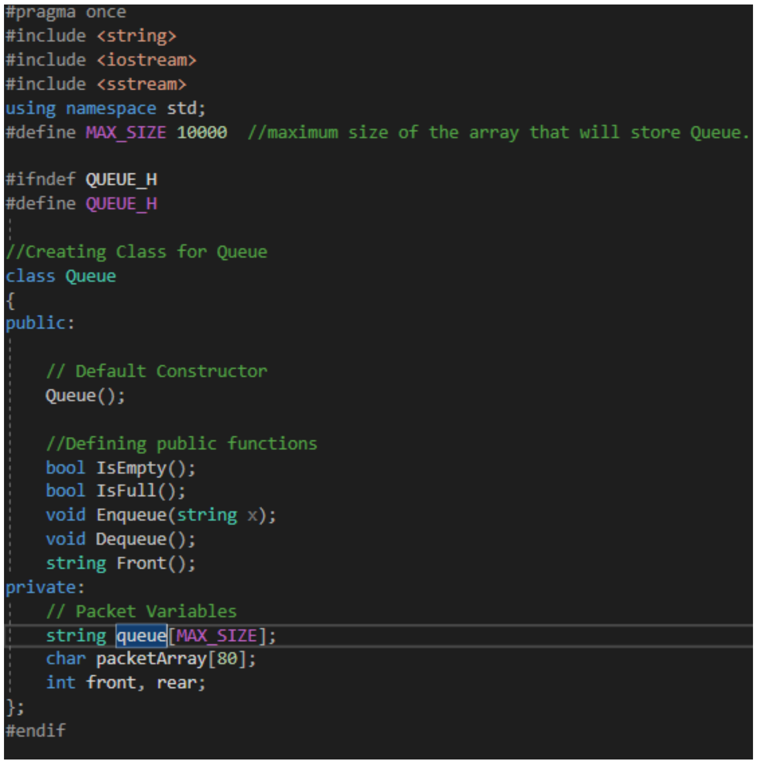
```
string Heap::getDataAt(int index)
{
    return priorityQ[index];
}
```

Rushi Patel
Yash Patel
Jeet Desai

Algorithms and Data Structures Final Project Report

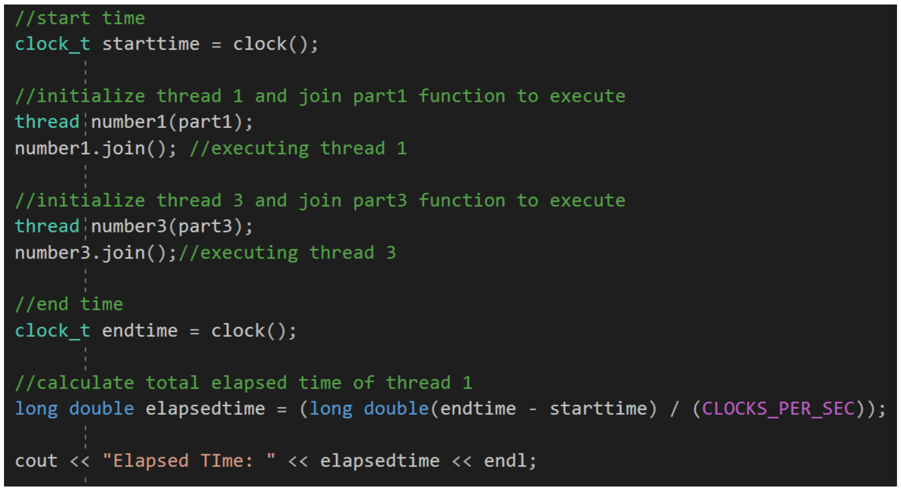
Queue initializations

Array-based Implementation was used to create packet class to handle packets going in/out of queue



```
#pragma once
#include <string>
#include <iostream>
#include <sstream>
using namespace std;
#define MAX_SIZE 10000  //maximum size of the array that will store Queue.

#ifndef QUEUE_H
#define QUEUE_H

//Creating Class for Queue
class Queue
{
public:

    // Default Constructor
    Queue();

    //Defining public functions
    bool IsEmpty();
    bool IsFull();
    void Enqueue(string x);
    void Dequeue();
    string Front();
private:
    // Packet Variables
    string queue[MAX_SIZE];
    char packetArray[80];
    int front, rear;
};
#endif
```

2. Evaluate the performance for different sizes of the input files.



```
//start time
clock_t starttime = clock();

//initialize thread 1 and join part1 function to execute
thread number1(part1);
number1.join(); //executing thread 1

//initialize thread 3 and join part3 function to execute
thread number3(part3);
number3.join();//executing thread 3

//end time
clock_t endtime = clock();

//calculate total elapsed time of thread 1
long double elapsedtime = (long double(endtime - starttime) / (CLOCKS_PER_SEC));

cout << "Elapsed TIme: " << elapsedtime << endl;
```

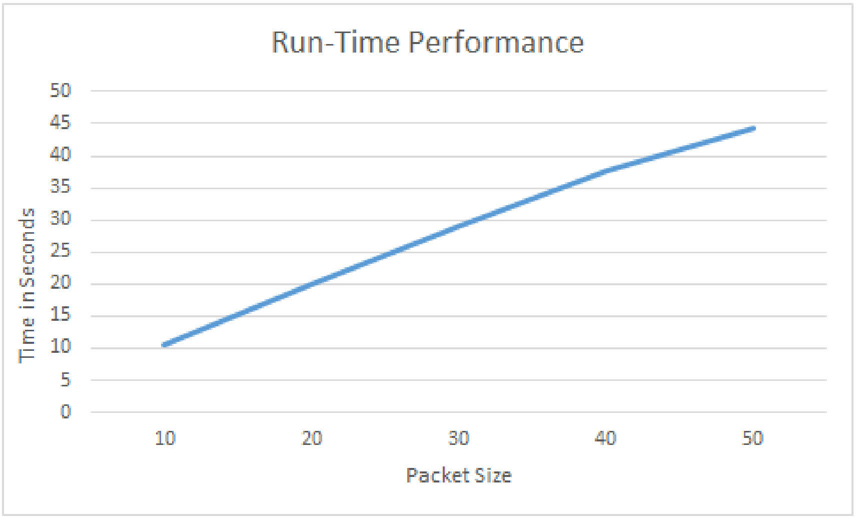
Rushi Patel
Yash Patel
Jeet Desai

Algorithms and Data Structures Final Project Report

Packet Size	Run Time (seconds)
10	10.526
20	19.86
30	29.166
40	37.606
50	44.203

The reason why we took such packet sizes and limited till only 50 packets is because of the 1 second delay in thread 3 when everytime a packet from a queue is read. If we were to test with larger number of packets there would obviously be a larger run-time. So with this analysis we can assume that, the run-time would be approximate of how many packets are in the file.

Run-Time performance graph



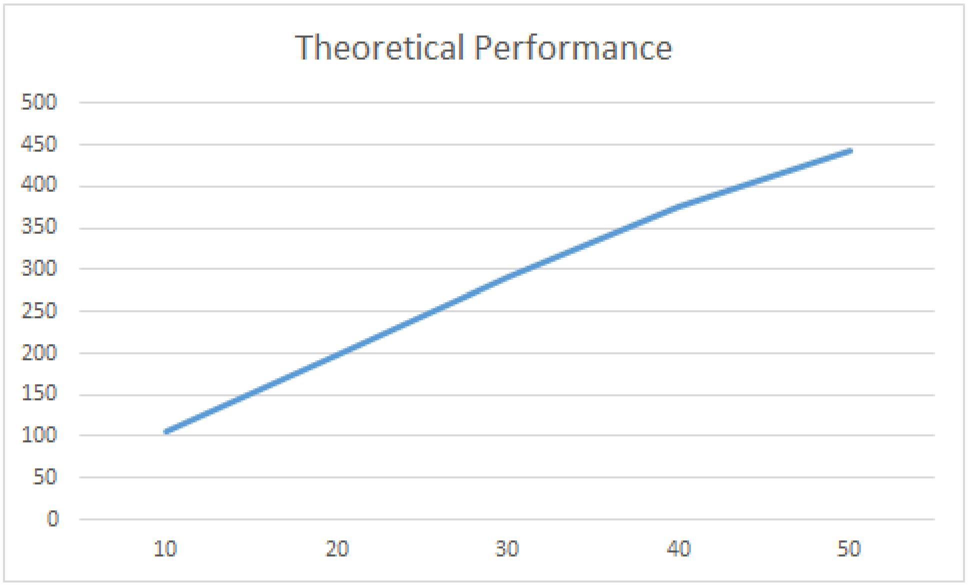
Rushi Patel
Yash Patel
Jeet Desai

Algorithms and Data Structures Final Project Report

3. Compare the run-time performance to the theoretical performance

Packet Size	Run-Time Performance	Theoretical Performance $T(n) = c_{op}.C(n)$
10	10.526	T(n) = 10 * 10.526 = 105.26
20	19.86	T(n) = 10 * 19.86 = 198.6
30	29.166	T(n) = 10 * 29.166 = 291.66
40	37.606	T(n) = 10 * 37.606 = 376.06
50	44.203	T(n) = 10 * 44.203 = 442.03

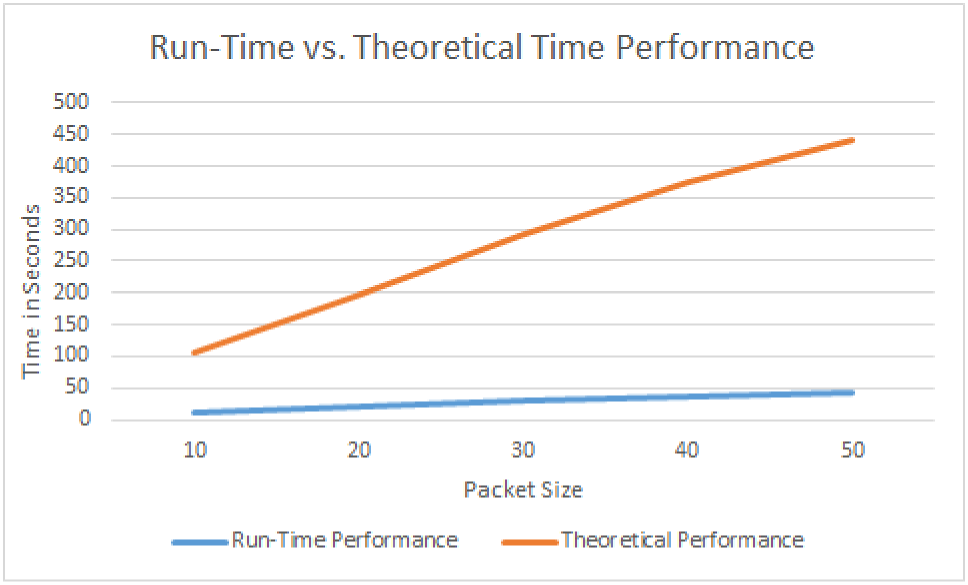
Theoretical performance graph



Rushi Patel
Yash Patel
Jeet Desai

Algorithms and Data Structures Final Project Report

Run-Time and Theoretical Time comparison


Run-Time vs. Theoretical Time Performance
Time in Seconds
Packet Size
Run-Time Performance
Theoretical Performance