**CSE 330 Lab 9 Report**

Daniel Meyer

Data Structures

Fall 2017

**Status:** 100%

**Time Complexity:** O(logn)

**Storage Complexity:** O(n)

**Source Code:** Pages 2 - 3

**Sample Run:** Page 4

/\*

Daniel Meyer

11-20-17

CSE 330

Fall 2017

Lab 9: Priority Queue

Problem: Implement a Prioirty\_queue class that represents the data structure heap.

Algorithm: Created functions for push() which puts item on the bottom of the tree

where it percolates upward until sorted. Pop() function replaces top with the

smallest element and lets it fall through the binary tree until it is sorted.

Status: 100%

Time Complexity: O(logn) Storage Complexity: O(n)

\*/

#ifndef PRIORITY\_QUEUE\_H

#define PRIORITY\_QUEUE\_H

// Priority\_queue.h -- a priority\_queue implemented as a heap

#include <vector>

using namespace std;

template <class T>

class Priority\_queue {

public:

Priority\_queue() : c() { }

Priority\_queue(const Priority\_queue & p) : c(p.c) { }

Priority\_queue & operator=(const Priority\_queue & p) { c = p.c; return \*this; }

// vector destructor is automatically called when an obj of type Priority\_queue is destroyed

bool empty() { return c.empty(); }

unsigned int size() { return c.size(); }

void push(const T & x);

void pop();

T & top() { return c.front(); }

private:

vector<T> c;

};

//Pushes new element to the back of the vector and percolates up until sorted

template <class T>

void Priority\_queue<T>::push(const T &x)

{

c.push\_back(x);

unsigned int n = c.size() - 1;

unsigned int parent = (n - 1) / 2;

while (n > 0 && c[parent] < c[n]) {

swap(c[n], c[parent]);

n = parent;

parent = (n - 1) / 2;

}

}

//Sets first element to last element and pops last element in vector where it falls through tree until sorted

template <class T>

void Priority\_queue<T>::pop()

{

if (empty()) {

return;

}

c[0] = c.back(); //root = last value

c.pop\_back(); //remove last

unsigned int n = 0; //root

unsigned int lc = 1; //left child

unsigned int rc = 2; //right child

unsigned int largest;

while(lc < c.size()) {

largest = lc; //assume left, accounts for only 2 elements

if(rc < c.size() && c[lc] < c[rc]) { //check if right child exists and left < right

largest = rc; //largest = right

}

if(c[n] > c[largest]) {

return;

}

else {

swap(c[n], c[largest]);

}

//Setup for next run in loop

n = largest;

lc = (2 \* n) + 1;

rc = (2 \* n) + 2;

}

}

#endif

**Sample Run:**

Script started on Mon 27 Nov 2017 07:44:41 PM UTC

To run a command as administrator (user "root"), use "sudo <command>".

See "man sudo\_root" for details.

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ g++ Priority\_queue.h

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ g++ Priority\_queue\_test.cpp

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ ./a.out

All tests passed.

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ exit

Script done on Mon 27 Nov 2017 07:45:06 PM UTC