**CSCE 221 Assignment 5 Cover Page**

First Name: **Shah** Last Name: **Kaivan** UIN: **522006920**

User Name: **kks6920**  E-mail address: **kaivan29@tamu.edu**

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor.

System Office website: http://aggiehonor.tamu.edu/

|  |  |  |  |
| --- | --- | --- | --- |
| Type of Sources |  |  |  |
| People | Kyle Almeida |  |  |
| Web Pages (provide URL) | http://www.cplusplus.com/ | others |  |
| Printed Material | Data Structures and Algorithms in C++ |  |  |
| Other Sources |  |  |  |

I certify that I have listed all the sources that I used to develop the solutions/codes to the submitted work.

On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Shah Kaivan Date 11/21/14

**Program Description**

For this programming assignment, I had to implement a minimum priority queue based on two different data structures: an unsorted array and a binary heap. For both of these priority queues, I had to create member functions isEmpty, print\_PQ, insertItem, createPriorityQueue, minKey, removeKey, and decreaseKey. I also had to measure the number of comparisons performed in the removeKey and decreaseKey functions so that I could estimate running time of these functions.

For the input data, I had to create a file of 40 U.S. cities and their distances from a source city. My source city is Dallas. I stored these cities into a priority queue, and sorted them by repeatedly calling minKey and removeKey until the queue is empty.

**Purpose of the Assignment**

The purpose of this programming assignment is to learn about data structures and algorithms.More specifically, I learned about Minimum Priority Queues and Heaps.

**Data Structures Description**

The main data structure in this assignment is the Minimum Priority Queue. The priority queue contains a variable that is the size of the priority queue, and a pointer to an Item. The priority queue is an array of Items, and every Item contains a key value (distance) and an element value (city name).

**Algorithm Description**

isEmpty( ) – returns true if the priority queue is empty

print\_PQ( ) – every item in the priority queue

insertItem(Item i) – inserts i into the priority queue, and reorganizes queue if necessary (only reorganizes in the heap implementation)

createPriorityQueue( ) – reads input from a file and creates a priority queue with the data

minKey( ) – returns the item with the smallest key value

removeKey( ) – removes the item with the smallest key value, and reorganizes queue if necessary (only reorganizes in the heap implementation)

decreaseKey(string e, int k) – changes the key value of the item with elem e to the value k, and reorganizes queue if necessary (only reorganizes in the heap implementation)

**Comparisons/ Running Time**

In the unsorted array implementation, removeKey( ) always required size-1 comparisons.

In the heap implementation, removeKey( ) required log(size) comparisons in the worst case.

In the unsorted array implementation, decreaseKey( ) required size comparisons to find the desired Item in the worst case, and reorganizing took 0 comparisons.

In the heap implementation, decreaseKey( ) required size comparisons to find the desired Item in the worst case, and reorganizing took log(size) comparisons.

In total, sorting the list using the unsorted array takes n(n+1)/2 comparisons while using the heap version takes nlog(n) comparisons. Where n is the size.

|  |  |  |
| --- | --- | --- |
|  | Unsorted Array | Binary Heap |
| insertItem | O(1) | O(logn) |
| minKey | O(n) | O(1) |
| removeMin | O(n) | O(logn) |
| decreaseKey | O(n) | O(n) |

**Real-life Applications**

Three real life applications of a minimum priority queue are: simulating traffic, determining the order to treat patients in a hospital, and Huffman coding.

**Instructions to Compile and Run my Program**

Compile the MPQ\_array implementation with **g++ -std=c++11 -o Main \*.cpp** and run it with **./Main**

Same for the MPQ\_heap implementation, just make sure you are in the folder of the one you would like to compile and run

**Input and Output Specifications**When changing the key of a city, capitalize the first letter of each word. Ex: Los Angeles

Make sure the new distance is an integer; otherwise the program will change its distance to 0.

**Logical Exceptions**

If the city entered is not in the priority queue, the program will tell the user it was not found.