**Lab 5 Report**

The main problem that I am trying to solve is to implement different methods as heapsort, percolate up, and percolate down in order to sort the list of numbers from smallest to largest element in the list. As mentioned before one of the solutions that I found was to implement the percolate down and the percolate up methods in order to maintain the proper order between the children and the parent. In order to solve the problem, I first struggle when going through the different methods. What I had to do is break down the problem and work each method independently and trace different examples of just a few numbers to analyze in a better way how the algorithm was going to work. I also had a difficult time when I was determining whether I needed both methods percolate down and percolate up I first though I just needed one which was percolate up and that it would be enough to solve the problem. However, it did not solve the problem and had to re-analyze the problem where then I figured I needed both in order to maintain the proper order of the heap while analyzing or inserting different elements or numbers. Also, I had the issue since the file was not being read appropriately and it indicated that the line could not be read and an error that said that string could not be converted into integer which I found quite confusing. Another solution was to use the example that the professor provided us with in order to understand and then implement them myself after gathering more information on resources like YouTube, Google, and Zybooks.

Once I was done and the process finished with an exit code of 0 my methods were running effectively and I tested first by using a small list of numbers. What I learned from this lab is that heaps are used to find either the smallest or largest element and then once it is found stored it in an array. I definitely learned from this project how to implement heaps and heapsort along with other methods.

The Running time is 0(nlogn)

# CS2302 Data Structures  
# Programmed by Luis Garcia.  
# Last modified December 5, 2018.  
# Instructor Diego Aguirre.  
# Implementation of heaps and heap sort in order to sort a list  
# and return the unsorted list followed by the sorted list  
# Finding the smallest method and returningrom smallest to biggest.  
  
# Class and method given by professor.  
# Some lines were missing of the code and  
# I implemented whatever was left incomplete and necessary.  
class Heap:  
 def \_\_init\_\_(self):  
 self.heap\_array = []  
  
 def insert(self, k):  
 self.heap\_array.append(k)  
 self.percolateUp()  
  
 def extract\_min(self):  
 if self.is\_empty():  
 return None  
 min\_elem = self.heap\_array[0]  
  
 newValue = self.heap\_array.pop()  
 if len(self.heap\_array) > 0:  
 self.heap\_array[0] = newValue  
 self.percolate\_down(0)  
 return min\_elem  
  
 def is\_empty(self):  
 return len(self.heap\_array) == 0  
# This method is used to compare the parent and child node and if the comparision  
# violates the heap property then they will be switched and done so until heap  
# property is maintained.  
# this is used when the min value in the heap is extracted.  
 def percolate\_down(self, node\_index):  
 child\_index = 2 \* node\_index + 1  
 value = self.heap\_array[node\_index]  
  
 while child\_index < len(self.heap\_array):  
 min\_value = value  
 min\_index = -1  
 i = 0  
 while i < 2 and i + child\_index < len(self.heap\_array):  
 if self.heap\_array[i + child\_index] < min\_value:  
 min\_value = self.heap\_array[i+child\_index]  
 min\_index = i + child\_index  
 i = i + 1  
  
 if min\_value == value:  
 return  
 else:  
 temp = self.heap\_array[node\_index]  
 self.heap\_array[node\_index] = self.heap\_array[min\_index]  
 self.heap\_array[min\_index] = temp  
 node\_index = min\_index  
 child\_index = 2 \* node\_index + 1  
# this method is used to compare the parent and child node and if the comparision  
# violates the heap property then  
# they will be switched and done so until heap property is maintained at the root of  
# the heap.  
# this is used when a new node is added to the heap  
 def percolateUp(self):  
 node\_index = len(self.heap\_array) - 1  
 if node\_index == 0:  
 return  
 while node\_index > 0:  
 parent\_index = (node\_index-1)//2 # Compute the parent node's index  
 if self.heap\_array[node\_index] < self.heap\_array[parent\_index]:  
 temp = self.heap\_array[node\_index]  
 self.heap\_array[node\_index] = self.heap\_array[parent\_index]  
 self.heap\_array[parent\_index] = temp  
 node\_index = parent\_index  
  
 else: # Heap property is maintained.  
 break # Nothing else is required and program breaks  
# A unsorted array is the expected input and the sorting is done by extraction of the min node,  
# if percolate down is done correct then the result should be sorted with the min being at the beginning of the array  
 def heap\_sort(self):  
 res = []  
 while not self.is\_empty():  
 res.append(self.extract\_min())  
 return res  
  
 def print\_heap(self):  
 for i in range(len(self.heap\_array)):  
 print(self.heap\_array[i])  
  
def main():  
 smallestHeap = Heap()  
 file\_name = input("What's the name of the file that we are sorting today?\n")  
 file = open(file\_name, "r")  
 print("Here are the results:")  
 for line in file:  
 curr = line.split(",")  
 for number in curr:  
 smallestHeap.insert(int(number))  
 print("")  
 print("This is the unsorted list:")  
 smallestHeap.print\_heap()  
 sorted\_min\_heap = smallestHeap.heap\_sort()  
 print("This is the sorted list:")  
 print(sorted\_min\_heap)  
main()

Time O(nlogn)

“I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in class.”