**Introduction**

For this lab we are given certain part of the Min-Heap Implementation, we have to complete the rest of the implementation. Furthermore, we are going to use the “UnitTest” library to make sure that the heap is producing the correct outputs. Finally, once the min-heap implementation is finish we are going to use it to create the heap-sort method. We are going to use this method and the heap to sort the numbers that are in a separate text file under the same directory. The final result should be the numbers in the text file sorted in the console where we are going to print the result.

**Proposed Solution**

I approached this problem by first drawing the tree and creating the methods that are going to retrieve the parent and the children of the given index. I am going to use these methods for reordering the nodes of the heap after an insertion or an extraction of the min element. There are going to be two methods that are going to reorder the heap; one method is going to reorder the heap from the bottom up and the other one is going to reorder it from the root to the leaf, each one following a specific path. I am going to need two different ways of reordering because in inserting you insert the element in the next available leaf, that means that you have to compare the node to its parents all the way to the root. If one parent is bigger that the node, then I need to swap both nodes, this is to keep the heap from going smallest to largest. However, if extract the min node from the root and replace it with the last leaf, then I have to compare the root to the children; swapping the nodes that violate the rule. After I completed these two methods, I then completed the insert method by appending the given node to the list and reordering the heap from the bottom to the top. Finally, the last method for the heap class was the extract-min method. This method returns the root, replaces the root with the last leaf, and makes the heap shorter by one index. I use the reordering method that operates from the root to the leaf nodes; once again swapping the nodes that violate the rule. Now that the heap class is done, I implemented the heap-sort method. This method should insert all the numbers to a heap and extract them; once extracted, they are going to be sorted. The final part of the project was just reading the numbers from a text file and giving them to the heap-sort method.

**Experimental Results**

I first used the insert method to insert a few numbers in random order to the heap. Then I used the heap-sort method that extracted the numbers already sorted. Additionally, the other experiment that I did was using a file that contain several numbers. I first read the file and convert it to a list of integers, finally, I give the list to the heap-sort method. The end result is going to be the numbers in the file sorted, displayed in the console. Furthermore, I also used the unit testing library to test that the extracted number is indeed the min. number of the heap.



Figure 2 The output of the program

Figure 1 The input of the program

**Conclusion**

In this project I have learn a new sorting algorithm, heap-sort. In my opinion this is one of the easiest yet efficient algorithms that we can use to sort items. Additionally, I learned about unit testing and how to use it in python in order to automate the testing in the code.

**Appendix**

import unittest

class Heap:

def \_\_init\_\_(self):

*self*.heap\_array = []

*#Going to insert the item into the heap*

def insert(self, k):

*self*.heap\_array.append(k)

*self*.reorderUp(k)

*#Going to reorder the heap from the top of the path to the root*

def reorderUp(self, k):

i = *self*.heap\_array.index(k)

while i > 0:

p = *self*.get\_parent(i)

if(p >= 0):

if(*self*.heap\_array[i] < *self*.heap\_array[p]):

*self*.swap(p, i)

i = p

*#Going to reorder the heap from the top of the path to the leaf, following only one path*

def reorderDown(self, k):

i = *self*.heap\_array.index(k)

while i < len(*self*.heap\_array):

l, r = *self*.get\_childs(i)

if(r < len(*self*.heap\_array)):

*#Going to see if the left or the right is smaller*

*#We are going to swap the parent with the smallest of either child*

if(*self*.heap\_array[l] < *self*.heap\_array[r]

and *self*.heap\_array[l] < *self*.heap\_array[i]):

*self*.swap(i, l)

i = l

continue

elif(*self*.heap\_array[r] < *self*.heap\_array[l]

and *self*.heap\_array[r] < *self*.heap\_array[i]):

*self*.swap(i, r)

i = r

continue

i = r

*#Going to swap all the items given*

def swap(self, i0, i1):

t = *self*.heap\_array[i0]

*self*.heap\_array[i0] = *self*.heap\_array[i1]

*self*.heap\_array[i1] = t

*#Going to get the child of the given index*

def get\_childs(self, k):

return (k \* 2) + 1, (k \* 2) +2

*#Going to get the parent of the given index*

def get\_parent(self, k):

return (k - 1) // 2

*#Goind to extrac the min from the heap*

def extract\_min(self):

if *self*.is\_empty():

return None

min\_elem = *self*.heap\_array[0]

l = len(*self*.heap\_array)

*self*.heap\_array[0] = *self*.heap\_array[l - 1]

*self*.reorderDown(*self*.heap\_array[0])

*self*.heap\_array = *self*.heap\_array[:l-1]

return min\_elem

*#Going to return if the heap is empty*

def is\_empty(self):

return len(*self*.heap\_array) == 0

*#Peek at the top number in the tree*

def peek(self):

if not *self*.is\_empty():

return *self*.heap\_array[0]

else:

return None

*#Going to insert all the items in heap and extrac them all ordered.*

def heapSort(l):

h = Heap()

for num in l:

h.insert(num)

while not h.is\_empty() :

n = h.extract\_min()

if(not h.is\_empty()):

assert n < h.peek() *#Testing that the value extracted is the minumum*

print(n),

*#Getting the numbers from the number file*

def getFileNums(fileName):

f = open(fileName, "r")

numbers = []

for l in f:

numbers.extend(list(map(int, l.split(","))))

return numbers

*#Going to sort all the numbers in the numbers file.*

heapSort(getFileNums("numbers.txt"))

“I certify that this project is entirely my own. 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 the class”

Eduardo Herrera