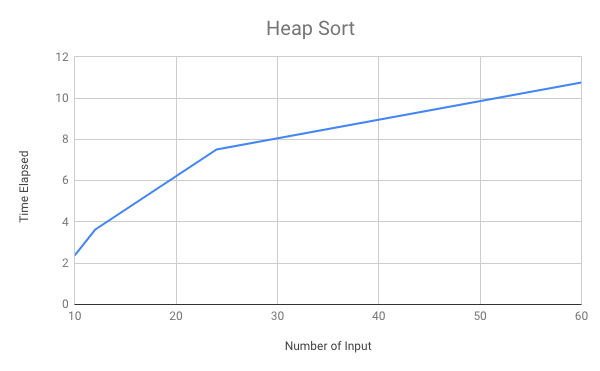
**Lab 5 – Heap Sort**

The objective of this lab was to complete the Min Heap Class and implement Heap Sort. In order to complete the Heap Class, the **extract\_min(),** insert(), and **heapSort()** methods were added.

In order to implement the heap sort, the methods added to the class were used to extract the “root” or the minimum element in the Heaps created by the insertion method and depositing them into a new array which will naturally sort the elements in ascending order, since the smallest possible element will always exist at the root of any Heap (or should). After each extraction, the element found at the farthest most leaf is placed at the root and is then compared to its children, being swapped by the smallest child to ensure that the root is populated with the minimum. Since the Heap properties dictate that parents should always contain values smaller than the children, this ensures that every root extraction will be the next necessary element to complete the sorted list. Since this lab was not really the solving of a particular problem but the implementation of a Heap Sort, not a lot more can be said in regards the different approaches used to tackle the problem.

The **extract\_min ()** method calls on a **percolate\_down()** method that performs the comparisons after each root extraction and ensures that the smallest element will be at the root for the next extraction.

The results provided by my implementation of Heap Sort appear to have fallen into the expected **O(n\*logn)** that comes with this type of sort. The following table provides a visual representation of the time elapsed for each execution of Heap Sort as the input increases.



This was more than likely due to the Heap’s binary search tree nature, which organizes the data within it in such a way as traversal through the array take place, no element at the left child or element at the right child will ever be smaller than the parent. Given the formulas for the parent, left and right children, this binary “nature” can be implemented even within an array.

In conclusion, this lab was relatively straight forward and it required the implementation of the Heap class and the Heap sort within that class. Since implementation had already been discussed in class and even included in the last exam, this lab was relatively simple but still crucial to my understanding of Min Heaps and how their very nature can be used to sort data. Again, since no complex problem or data analysis was required for this particular lab, the report is brief.

**Appendix**

#!/usr/bin/env python3  
# -\*- coding: utf-8 -\*-  
"""  
File: Main.py  
Name: Angel Villalpando  
Date: 11/27/2018  
Course: CS 2302 - Data Structures  
Description: This file contains the program that turns a list of numbers into a Min Heap and then performs Heap sort by  
performing root extraction and pushing in into a final array, where the numbers are sorted in ascending order.   
"""  
  
*from* HeapClass *import* Heap  
*import* time  
  
*def* list\_to\_heap(*list*):  
  
 heap = Heap()  
 *for* i *in* range(len(*list*)):  
 heap.insert(*list*[i])  
  
 *return* heap  
  
*def* main():  
  
  
 testList = [8, 4, 3, 2, 1, 5, 6, 7, 12, 15, 20, 25]  
 print("\n\nThe first, original list is: ", testList)  
 testHeap = list\_to\_heap(testList)  
  
 testList2 = [2, 5, 4, 1, 7, 8, 0, 14, 9, 15]  
 print("\nThe second, original listt is: ", testList2)  
 testHeap2 = list\_to\_heap(testList2)  
  
 testList3 = [7, 55, 15, 12, 90, 95, 4, 87, 100, 49]  
 print("\nThe third, original list is: ", testList3)  
 testHeap3 = list\_to\_heap(testList3)  
  
 testList4 = [66, 17, 88, 79, 90, 21, 72, 43, 34, 45, 76, 27, 98, 89, 0, 31, 8, 2, 5, 3, 44, 55, 65, 78]  
 print("\nThe third, original list is: ", testList4)  
 testHeap4 = list\_to\_heap(testList4)  
  
 testList5 = [19, 30, 61, 12, 83, 74, 85, 16, 67, 38, 29, 40, 71, 22, 93, 84,  
 95, 26, 77, 48, 39, 50, 81, 32, 3, 94, 5, 36, 87, 58, 49, 60, 91,  
 42, 13, 4, 15, 46, 97, 68, 59, 70, 1, 52, 23, 14, 25, 56, 7, 78,  
 69, 80, 11, 62, 33, 24, 3]  
 print("\nThe third, original list is: ", testList5)  
 testHeap5 = list\_to\_heap(testList5)  
  
 # testList5 = [66, 17, 88, 79, 90, 21, 72, 43, 34, 45, 76, 27, 98, 89, 0, 31, 8, 2, 5, 3, 44, 55, 65, 78]  
 # print("\nThe third, original list is: ", testList3)  
 # testHeap5 = list\_to\_heap(testList4)  
  
  
  
  
 print("\nList 1 turned into MinHeap: ")  
 testHeap.print\_heap()  
  
 print("\nList 2 turned into MinHeap: ")  
 testHeap2.print\_heap()  
  
 print("\nList 3 turned into MinHeap: ")  
 testHeap3.print\_heap()  
  
 print("\nList 4 turned into Minheap: ")  
 testHeap4.print\_heap()  
  
 print("\nList 5 turned into a MinHea: ")  
 testHeap5.print\_heap()  
  
  
  
  
 print("\nHeapSort for first list: ")  
 t0 = time.time()  
 sorted = testHeap.heapSort()  
 t1 = time.time()  
 total = t1 - t0  
 print(sorted)  
 print("Time: ", total)  
  
 print("\nHeapSort for second list: ")  
 t2 = time.time()  
 sorted2 = testHeap2.heapSort()  
 t3 = time.time()  
 total2 = t3 - t2  
 print(sorted2)  
 print("Time: ", total2)  
  
 print("\nHeapSort for third list: ")  
 t4 = time.time()  
 sorted3 = testHeap3.heapSort()  
 t5 = time.time()  
 total3 = t5 - t4  
 print(sorted3)  
 print("Time: ", total3)  
  
 print("\nHeapSort for fourth list: ")  
 t6 = time.time()  
 sorted4 = testHeap4.heapSort()  
 t7 = time.time()  
 total4 = t7 - t6  
 print(sorted4)  
 print("Time: ", total4)  
  
 # print("\nHeapSort for fifth list: ")  
 # t8 = time.time()  
 # sorted5 = testHeap5.heapSort()  
 # print(sorted5)  
 # t9 = time.time()  
 # total5 = t9 - t8  
 # print("Time: ", total5)  
  
  
  
main()

* Main Class

*class* Heap:  
  
 *def \_\_init\_\_*(self):  
 self.heap\_array = []  
 self.current\_size = 0  
  
 *def* percolate\_up(self, *index*, *size*): # method percolates the minimum value up after each insertion  
  
 current = (2 \* *index*) + 1  
 value = self.heap\_array[*index*]  
  
 *while* current < *size*:  
 min = value  
 min\_index = -1  
 i = 0  
 *while* i < 2 *and* i + current < *size*:  
 *if* self.heap\_array[i + current] < min:  
 min = self.heap\_array[i + current]  
 min\_index = i + current  
 i += 1  
  
 *if* min == value:  
 *return* temp = self.heap\_array[*index*]  
 self.heap\_array[*index*] = self.heap\_array[min\_index]  
 self.heap\_array[min\_index] = temp  
  
  
  
 *def* insert(self, *k*): # the breadth of this method is done by the percolate method, which ensures MinHeap properties  
  
 self.heap\_array.append(*k*)  
 self.current\_size += 1  
 *for* i *in* range(len(self.heap\_array)):  
 self.percolate\_up(i, self.current\_size)  
  
 *def* percolate\_down(self, *index*): # method percolates larger numbers down when each min root is "extracted"  
  
 *while* (*index* \* 2) < self.current\_size:  
 *if index* \* 2 + 1 > self.current\_size:  
 minChild = *index* \* 2  
 *else*:  
 *if* self.heap\_array[*index* \* 2 + 1] < self.heap\_array[*index* \* 2 + 2]: # chooses the smaller child of the two  
 minChild = (*index* \* 2) + 1  
 *else*:  
 minChild = (*index* \* 2) + 2  
  
 *if* self.heap\_array[*index*] > self.heap\_array[minChild]: # swaps actual values when min heap properties have been violated  
 temp = self.heap\_array[*index*]  
 self.heap\_array[*index*] = self.heap\_array[minChild]  
 self.heap\_array[minChild] = temp  
 index = minChild  
  
  
  
 *def* extract\_min(self):  
  
 *if* self.is\_empty():  
 *return None* min\_elem = self.heap\_array[0]  
 self.heap\_array[0] = self.heap\_array[self.current\_size-1]  
 self.current\_size = self.current\_size - 1  
 self.percolate\_down(0)  
  
 *return* min\_elem  
  
 *def* is\_empty(self):  
 *return* len(self.heap\_array) == 0  
  
 *def* heapSort(self): # method returns a new list with the values now sorted  
 result = []  
 *while* self.current\_size != 0:  
 currMin = self.extract\_min()  
 result.append(currMin)  
  
 *return* result  
  
 *def* print\_heap(self):  
 print(self.heap\_array)

* Heap Class

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

-Angel Villalpando

11/27/2018