Elizabeth Barragan
CS 2302 - Fall 2018
Instructor: Diego Aguirre
TA: Saha, Manoj Pravakar
Lab 5 - Report

[Heap Sort / Min Heap]

Introduction:

The purpose of this lab was to implement heap sort utilizing a min heap. In order for heap sort to work I had to create a class that contained the attributes and functions of the min heap. Such functions include: insert, extract min, percolate up, and percolate down.

Proposed Solution:

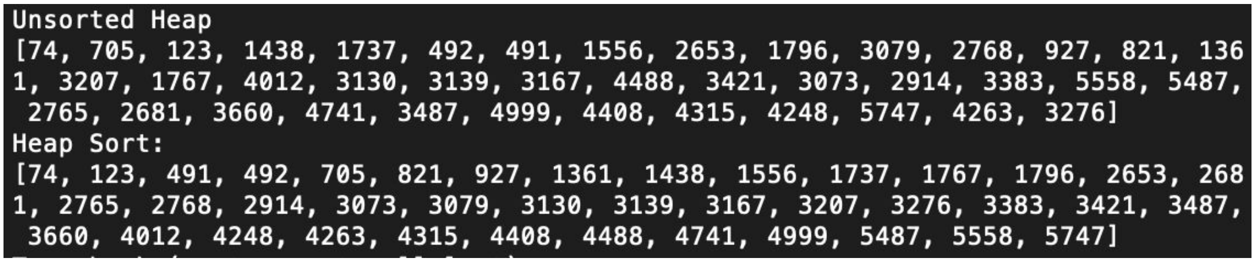
To begin with, my program reads a file of integers, each number separated by line, I then populate the min heap with the integers from the file. After the min heap is created I send the heap to the heap sort function.

Heap sort :

Utilizing the min heap function, extract_min() I used a loop to extract the minimum elements in my heap, then reorganizing the heap to meet the min heap requirements, until the original heap was empty.

To keep the min heap properties, I utilized the percolate_up and percolate_down codes from Zybooks. Since they implemented a max heap, I had to change some parts of the code to be able to make it into a min heap. In the extract_min() method I had to extract the value of the root, and I replaced it with the last element of the heap. Then I utilized percolate_down starting from the new root to reorganize my min heap.

To demonstrate my results I printed the original min heap, then after applying heap sort I then printed the sorted heap. The numbers that are sorted were given by the text file.



```
Unsorted Heap
[74, 705, 123, 1438, 1737, 492, 491, 1556, 2653, 1796, 3079, 2768, 927, 821, 136
1, 3207, 1767, 4012, 3130, 3139, 3167, 4488, 3421, 3073, 2914, 3383, 5558, 5487,
 2765, 2681, 3660, 4741, 3487, 4999, 4408, 4315, 4248, 5747, 4263, 3276]
Heap Sort:
[74, 123, 491, 492, 705, 821, 927, 1361, 1438, 1556, 1737, 1767, 1796, 2653, 268
1, 2765, 2768, 2914, 3073, 3079, 3130, 3139, 3167, 3207, 3276, 3383, 3421, 3487,
 3660, 4012, 4248, 4263, 4315, 4408, 4488, 4741, 4999, 5487, 5558, 5747]
```

The standard running time for heap sort is O(nlogn).

Appendix:

```
import Heap

#Heap Sort Implementation
def heap_sort(self):
  heap_sorted = Heap()
  while len(self.heap_array)>0:
    min_dummy = self.extract_min()
    heap_sorted.insert(min_dummy)
  return heap_sorted


unsorted_heap = Heap()

g = open("numbers.txt")
num_list = g.readlines()



#Populating the heap from the list of numbers
for ln in num_list:
    ln = int(ln.replace('\n',''))
    unsorted_heap.insert(ln)
print(unsorted_heap.heap_array)

unsorted_heap = unsorted_heap.heap_sort()

print("Sorted heap: ")
print(unsorted_heap.heap_array)

class Heap:
    def __init__(self):
        self.heap_array =[]

    '''
    Method that simply inserts a value to the Heap utilizing the percolate_up
    method to traverse the heap to insert to the appropriate position.
    '''
```

```
def insert (self,k):
    self.heap_array.append(k)
    self.percolate_up(len(self.heap_array) - 1)


'''
 Method that removes the last value from the heap in support of
 another method to swap (percolate_down) and position the values
 where they should be.
'''
def extract_min(self):
    if self.is_empty():
        return None

    min_elem = self.heap_array[0]   #last value of the heap would be in index 0
    min = self.heap_array.pop()

    if len(self.heap_array)>0:
        self.heap_array[0] = min
        self.percolate_down(0)
    return min_elem

def is_empty(self):
    return len(self.heap_array)==0


'''
Auxiliary methods to maintain the properties of a min heap
         - percolate_up to insert a new value in the heap
         - percoulate_down to remove a value from the heap
        '''
def percolate_up(self, i):
    while i > 0:

        parent_i = (i - 1) // 2    # computes the parent node's index

        #checks the value of the index and its's parent to see if there is no violation
        if self.heap_array[i] >= self.heap_array[parent_i]:
            return
            else:
                        #swaps the two values
                        self.heap_array[i] , self.heap_array[parent_i] = self.heap_array[parent_i] ,
self.heap_array[i]
                        #continue to loop from the parent node
                        i = parent_i
    '''
             Method that traverses until violation is found and then it fixes it
            '''
    def percolate_down(self, node_i):
        child_index = 2 * node_i + 1
        value = self.heap_array[node_i]

        while child_index < len(self.heap_array):
            # Find the min among the node and all the node's children
            min_value = value
            min_i = -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_i = i + child_index
                i = i + 1

            # check for a violation of the min heap property
            if min_value == value:
                return
            else:
                        self.heap_array[node_i] , self.heap_array[min_i] = self.heap_array[min_i] ,
self.heap_array[node_i]
                #continue loop from the min index node
            node_i = min_i
            child_index = 2 * node_i + 1

    '''
    Heap sort implementation
    '''
    def heap_sort(self):
        heap_sorted = Heap()
        while len(self.heap_array)>0:
            min_dummy = self.extract_min()
            heap_sorted.insert(min_dummy)
        return heap_sorted


'''
Creation of the heap along with reading a txt file to obtain numbers
to work with the different operations
'''
unsorted_heap = Heap()

g = open("numbers.txt")
num_list = g.readlines()


#reading the list to the Heap
for ln in num_list:
    ln = int(ln.replace('\n',''))     #numbers will be sepearated by a space
    unsorted_heap.insert(ln)                # insert the numbers(ln) into the heap

print("Unsorted Heap ")
print(unsorted_heap.heap_array)

unsorted_heap = unsorted_heap.heap_sort()

print("Heap Sort: ")
print(unsorted_heap.heap_array)
```

“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 the class.”