Govinda KC
80540998
Instructor: Diego Aguirre
TA: Manoj Sah

CS 2302 Lab 5A: Report

Problem Description:

We have asked to implement the heap data structure and use this data structure to implement the heap sort algorithm to sort the multiple arrays that are populated by random integers. We have to test the implementation by doing at least one of the following options:

(i) Units Tests (II) Reading a file (iii) using hard coded list.

Purposed Solution:

I purposed the solution of the given problem by first creating a heap class will insert the items into it. It also has ability to look up, retrieve and then to keep the properties of a min-heap.

So, **Heap class** defines the properties of the Heap, such as how to insert and also the swapping.
Insert method will insert the element into the heap that takes the value to be inserted as an argument.
Move_up method will work to swap elements from last item until we get the properties of the min heap is met. The relation between the parent and child is: if i is the child, parent = (i-1)//2.
Move_down method will work just like above, only difference is it will swap the item from the first item until the properties of the min heap is met.

Heap_sort method will implement heap sort as it finds the maximum element and then swaps until the array is sorted. The sorted heap is in the ascending order.

Running Time complexity:

Insertion occurs in the cost of O(log(n)).
Heap sort occurs in O(n*log(n)).
Read: it reads the file in O(n).

Output Results:

Among the given different options to implement the code, I used the two implantations. And the following are the outputs.

```
gvin@Govinda:~/Desktop/Lab_5_final$ python lab_5_heaps.py
Testing the hard-coded values.

Before sorting:  [7, 43, 35, 23, 75]
After sorting:  [7, 23, 35, 43, 75]

Testing by using the text file

The list before heap sort:  [11, 44, 23, 2, 55, 34, 56]
The list after heap sort  [2, 11, 23, 34, 44, 55, 56]
gvin@Govinda:~/Desktop/Lab_5_final$
```

We can see the two implementations of the code with its output. The output is sorted heap.

Appendix:

Heapy.py:

```
################################################################################
#########################
#   Govinda KC
#   CS 2302 TR 10:30 - 11:50
#   Lab Assignment 5 Option A
#   Instructor: Diego Aguirre
#   Teaching Assistant: Manoj Saha
################################################################################
#########################

# Class that creates the conditions that can be done to the heap
class Heap:
    def __init__(self): # initializes the object
        self.heap_array = [] # Initialize to empty

# Method will insert the items from the array list into the heap array. This is done in O(log n)
time
    def insert(self, k):
        self.heap_array.append(k)
```

```
      self.move_up(len(self.heap_array) - 1) # percolate up from the last index because we want to keep the property of a min-heap

# Method will begin to swap elements from last item in until we get the properties of a min-heap (parent root is smaller than children) are met. This is
   # done in O(log n)
   def move_up(self, node):
      while node > 0:
         parent_node = (node - 1) // 2 # Used to find parent of current node
         if self.heap_array[node] >= self.heap_array[parent_node]: # Used to check if the max heap is present
            return
         else: # This is where we need to swap elements to meet property to min-heap
            #print("Swapping: %d <-> %d" % (self.heap_array[parent_node], self.heap_array[node]))
            temp = self.heap_array[node]
            self.heap_array[node] = self.heap_array[parent_node]
            self.heap_array[parent_node] = temp

            node = parent_node

# Method will swap the elements from the first item in the heap until the properties of a min-heap are met. This is done in O(log n)
   def move_down(self, node, heap_list, size):
      child_index = (2 * node) + 1
      element = heap_list[node]

      while child_index < size: # Find the max among the node and all the node's children
         max_value, max_index = element, -1
         i = 0

         while i < 2 and i + child_index < size:
            if heap_list[i + child_index] > max_value:
               max_value = heap_list[i + child_index]
               max_index = i + child_index
            i = i + 1

         if max_value == element:
            return

         # We will swap the current node index with the max index using a temp variable
         temp = heap_list[node]
         heap_list[node] = heap_list[max_index]
         heap_list[max_index] = temp
```

```
        node = max_index
        child_index = 2 * node + 1

# Method will check if the heap is empty by comparing the lenght of the array is 0.
   def is_empty(self):
      return len(self.heap_array) == 0
```

```
################################################################################
#########################
#   Govinda KC
#   CS 2302 TR 10:30 - 11:50
#   Lab Assignment 5 Option A
#   Instructor: Diego Aguirre
#   Teaching Assistant: Manoj Saha
################################################################################
#########################

import Heap, time, random


# Method will implement heap sort to sort the list into an ascending order. This is done in O(n
log n)
def heap_sort(heap_list):
   h = Heap.Heap()
   i = len(heap_list) // 2 - 1
   while i >= 0:
      h.move_down(i, heap_list, len(heap_list))
      i = i - 1

   i = len(heap_list) - 1
   while i > 0: # Will be used for swapping elements
      temp = heap_list[0]
      heap_list[0] = heap_list[i]
      heap_list[i] = temp
      h.move_down(0, heap_list, i)
      i = i - 1

#---------------------------------------------------------------------
# Method will read a text file and then add rows to it's own list. This is done in O(n)
try:
   def read_file():
```

```
        heap_list = []
        file = open("test_file.txt", "r+")
        line = file.read().split(",") # Separates the elements in the text file by a comma

        for num in line: # Appends the elements to the heap
            heap_list.append(num)

        heap_list = list(map(int, heap_list))
        return heap_list
except FileNotFoundError:
    print ("File not found, please check the directory and try again.")

#---------------------------------------------------------------------
# Testing with hard-coded values
def main():
    print("Testing the hard-coded values.")
    hard_list = [7,43,35,23,75]
    print("\nBefore sorting: ", hard_list)
    heap_sort(hard_list)
    print("After sorting: ", hard_list)

# Testing with the text file values

    print("\nTesting by using the text file")
    read_list = read_file()
    print("\nThe list before heap sort: ", read_list)
    heap_sort(read_list)
    print("The list after heap sort ", read_list)

main()
```

For the extra problems code, please refer the github account link.

"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."

GKC