DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
Faculty of Engineering
University of Moratuwa

CS 2022: Data Structures & Algorithms

Lab 5: Heap Data Structure

Objectives: The aim of this lab session is for you to explore the use of Heap data structure for problem solving.

Learning Outcomes: After completing this lab the students should be able to;

- Implement a priority queue using a heap data structure.
- Use heaps for solving problems.

In this lab session you will;

1. Complete several operations of a Heap implementation.
2. Complete the implementation of a Priority Queue using the Heap implemented in task 1.
3. Write a program to extract the k largest numbers in a list of numbers.
4. Evaluate the time complexity of the solution of the task 3.

You are expected to use python programming language for completion of this lab exercise. You should upload your completed code in separate .py files to the assignment in the Moodle page. The expected effort required is 30 minutes for the first task, 15 minutes for the second task, 45 minutes for the third task and 60 minutes for the fourth task.

Task 1: *Complete the Heap Implementation*

In the first task, you are given a partially completed implementation of a Heap data structure in Python. The representation uses a list to store the heap and the items in the heap are stored from the second element (with index 1) of the list. For example the heap which contains the items 16, 14, 10, 8, 7, 9, 3, 2, 4 and 1 will be represented by the list ['Empty', 16, 14, 10, 8, 7, 9, 3, 2, 4, 1]. The current implementation has the following method implemented.

- `HeapParent(i)` - Returns the index of the parent node of node ***i***.
- `HeapLeft(i)` - Returns the index of the left child node of node ***i***.
- `HeapRight(i)` - Returns the index of the right child node of node ***i***.
- `HeapMax(A)` - Returns the maximum element in the heap.
- `HeapMaxHeapify(A, i)` - Converts the tree rooted by the node ***i*** into a Heap. As the pre-condition the sub trees routed by left child and right child of the node ***i*** should be Heaps.
- `HeapBuildMaxHeap(A)` – Converts the list ***A*** into a Heap.

Your first task is to complete the following functions for the Heap implementation.

- `HeapExtractMax(A)` – Extracts the maximum element in the Heap and returns it.
- `HeapIncreaseKey(A, i, key)` – Increase the key value of the ith node in the Heap to value specified by the input parameter key and return the new Heap.
- `HeapMaxHeapInsert (A, key)` – Insert the key value specified by the input parameter key to the Heap and return it.

The code with the completed functions and the templates for incomplete functions are available in the "Heap_150000X.py" file. A sample test file which is used to test the some of the current implementations are available in "HT_150000X.py" file.

You should complete the task and verify that your code works using suitable test cases. However you are required to only upload the completed Heap implementation with the filename "Heap_<your_index_number>.py". **You will be evaluate based on the correctness of your code and the quality of the code.**

Task 2: ***Complete the implementation of a Priority Queue***

You are given a partially completed implementation of a Priority Queue implementation in Python. The current implementation has the following method implemented.

- `MPQCreate()` – Creates an empty Max Priority Queue and returns it.
- `MPQCreate(A)` – Converts the list A into a Max Priority Queue and returns it.
- `MPQEnqueue(A, key)` – Inserts the element with value key to the Max Priority Queue A.
- `MPQDequeue(A)` – Removes an element from the Max Priority Queue A.
- `MPQIncreasePriority(A, i, newKey)` – Increase the priority of the element at the i^{th} location of the list to the value specified by `newKey`. (Note that the i^{th} element may not be the i^{th} largest element in the list.)

Your second task is to complete the following function for the Max Priority Queue implementation.

- `MPQDecreasePriority(A, i, newKey)` – Decrease the priority of the element at the i^{th} location of the list to the value specified by `newKey`. (Note that the i^{th} element may not be the i^{th} largest element in the list.)

The code with the completed functions and the templates for incomplete functions are available in the "MPQ_150000X.py" file.

You should complete the task and verify that your code works using suitable test cases. However you are required to only upload the completed Max Priority Queue implementation with the filename "MPQ_<your_index_number>.py". **You will be evaluate based on the correctness of your code, the proper reuse of the code already available and the elegance of the solution and the quality of the code.**

Task 3: ***Write a program to extract the k largest numbers in a list of numbers***

The third task of the lab is to extract the ***k*** largest numbers from a given list of ***n*** numbers. You are required to implement a Python program that will accept a list of (***n***) numbers and a ***k*** value, and return a list with the ***k*** largest numbers of the input list. It is given to you that ***n*** is very large compared to ***k*** (***n*** > > > ***k***). The signature of the method to extract the ***k*** largest numbers should be;

```
def ExtractMaxK (NumberList, k)
```

You should complete the task and verify that your code works using suitable test cases. However you are required to only upload the completed solution (with completed `ExtractMaxK` function) with the filename "EMK_<your_index_number>.py". **You will be evaluate based on the correctness of your solution, efficiency of your solution (time complexity), elegance of the solution and the quality of the code.**

Note: Consider the Heapsort algorithm and see whether you can get a good solution for the above problem using a modified version of Heapsort algorithm.

Task 4: ***Evaluate the Time Complexity of the Solution***

As the forth task, you are expected to evaluate the time complexity of the solution of the third task. For this task;

A. Evaluate the time complexity of the solution theoretically and present the results.
B. Evaluate the time complexity of your solution (time taken by the program to run) using sample data of suitable size and present the results.
C. Compare the theoretical results and the practical results and present your observations.

For the fourth task, you are expected to upload a file with the answers to parts A, B and C in PDF format. The filename should be "TC_<your_index_number>.py".