DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
CONCORDIA UNIVERSITY
COMP 352: Data Structure and Algorithms
Summer 2019
ASSIGNMENT 3
Due date and time: Monday June 10th by 11:59 pm

***Written Questions (50 marks)**:*

Q1. Design an algorithm to print a general tree. A general tree is one where each internal node can have any number of children. At the visit of a node of the tree, it prints the node based on its (x, y) coordinates where x = the visiting rank of the node (i.e., for the i^{th} node visited, its visiting rank is i) and y = depth of the node. Your algorithm must be **linear in time**, i.e., the tight bound on time complexity must be O(n) where *n* is the number of nodes. You must provide the pseudocode only (no Java code).

Q2. Design an algorithm for **non-recursive** inorder traversal of a binary tree. At each visit of a node, it prints the node's label. Provide the pseudocode only (no Java code). What is the big-Oh time complexity of your algorithm? You must explain your answer.

Q3. Show (diagrammatically) all the steps involved in the **in-place** heap sort on the following input sequence: 22, 45, 20, 2, 37, 33, 4, 7, 10, 6. You should sort the sequence in the increasing order.

***Programming Questions (50 marks)**:*

Q4. In class, we discussed the priority queue (PQ) ADT implemented using min-heap. In a min-heap, the element of the heap with the smallest key is the root of the binary tree. On the other hand, a max-heap has as root the element with the biggest key, and the relationship between the keys of a node and its parent is reversed of that of a min-heap. We also discussed an array-based implementation of heaps.

In this assignment, your task is to implement your own **adaptable and flexible priority queue (AFPQ)** ADT using both min- and max-heap. The specifications of the AFPQ are as follows: the heap(s) must be implemented from scratch using an array that is dynamically extendable. You are not allowed to use any list (including arraylist), tree, vector, or heap implementation already available in Java. You must not duplicate code for implementing min- and max-heaps (i.e. **the same code must be used for constructing min or max heaps. Hence think of a flexible way to parameterize your code for either min- or max heap**).

The priority queue is also **adaptable** meaning that any key or value of any entry of the priority queue can be modified, where the entry is passed as a parameter. An entry can also be removed from anywhere in the priority queue. The following are the access methods of the AFPQ ADT:

removeTop(): removes and returns the entry object (a key, value pair) with the smallest or biggest key depending on the current state of the priority queue (either Min or Max).

insert (k,v): Insert (k,v), which is a key(k) and value(v) pair, to the priority queue and returns the corresponding entry object in the priority queue.

top(): returns the top entry (with the minimum or the maximum key depending on whether it is a Min- or Max-priority queue) without removing the entry.

remove (e): Removes entry object e from the priority queue and returns the entry
replaceKey (e, k): replace entry e's key to k and return the old key.
replaceValue (e, v): replace entry e's value to v and return the old value.
toggle() transforms a min- to a max-priority queue or vice versa.
state (): returns the current state (Min or Max) of the priority queue.
isEmpty(): returns true if the priority queue is empty.
size(): returns the current number of entries in the priority queue

You have to submit the following deliverables:

a) Pseudocode of your implementation of the AFPQ ADT using a parameterized heap that is implemented using dynamic array. Note that Java code will not be considered as pseudo code. Your pseudo code must be on a higher and more abstract level.
b) Tight big-Oh time complexities of toggle (), remove (e), replaceKey (e, k), and replaceValue (e, v) operations, together with your explanations. **These methods should be as efficient as possible**.
c) Well documented Java source code with 20 different but representative examples demonstrating the functionality of your implemented ADT. These examples should demonstrate all cases of your ADT functionality (e.g., all operations of your ADT, several cases requiring automatic array extension, sufficient number of down and up-heap operations, changing keys, and removing entries from anywhere of heap).

Submit all your answers to written questions in PDF format only. For the Java programs, you must submit the source files together with the compiled executables. The solutions to all the questions should be zipped together into two separate .zip files (one for Theory and one for Programming) and submitted via EAS (Refer to the course outline for more details on submission guidelines).