CONCORDIA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING
COMP5511 Fall 2022

ASSIGNMENT 3

Due date : Monday November 7 2022

PART I – Theory questions.

1. Draw a single binary tree T, such that :
   • each internal node of T stores a single character;
   • a preorder traversal of T yields H L G D A P M B C F;
   • an inorder traversal of T yields G D L H M P A C B F.

2. Assume that the binary tree from question 1 is stored in an array-list as a complete binary tree as discussed in class. Specify the contents of such an array-list for this tree.

3. Draw the min-heap that results from running the bottom-up heap construction algorithm on the following list of values: 15 12 27 20 19 16 13 10 17 11.

   (a) Show the intermediate steps and the final tree representing the min-heap.
   (b) Afterwards perform the operation removeMin( ) 3 times and show the resulting min-heap after each step.

4. Create again a min-heap using the list of values from question 3 but this time you have to insert these values step by step using the order from left to right as shown in Question 3. Show the tree after each step and the final tree representing the min-heap.

5. Assume that the heap resulting from question 4 is stored in an array-list as a complete binary tree, specify the contents of such an array-list for this heap.

6. Consider the sequence of numbers 15 12 27 20 19 16 13 10 17 11. Write two algorithms in pseudo-code for the selection and insertion sorts. Use these 2 sorting algorithms to sort the given numbers in non-decreasing order.
   For each sorting algorithm specify :
   - Lower complexity bound (Ω) as a function of n;
   - Upper complexity bound (O) as a function of n;
   - Number of steps according to your pseudo code required to sort the sequence given above;
   - Best-case input (sequence of 10 integer values) requiring the smallest number of steps;
   - Worst-case input (sequence of 10 integer values) requiring the biggest number of steps.

   For all five points shown above you have to justify your answers.

PART II – Programming question.

In class, we discussed priority queues implemented with min-heaps. In a min-heap the element of the heap with the smallest key is the root of a binary tree. A max-heap always has as root the element with the biggest key and the relationship between the keys of a node and its parent is less than or equal (≤). Apparently, for a max-heap we will have the operation removeMax( ) (instead of removeMin( )).

Your task is to develop your own binary tree ADT and your own max-heap ADT. The max-heap ADT must be implemented using your binary tree ADT. You have to implement these two ADTs in Java. The max-heap ADT supports two additional operations:

- increaseKey(v,k) : assigns to vertex v in the max-heap the new key value k which cannot be smaller than the old key value. Be advised that after this operation the max-heap order must possibly be restored.
- heapSort(heap1) : sorts the heap in non-decreasing order and stores the sorted elements in a one-dimension array. The array can later be used by the calling code to print the sorted values.

Binary trees must be implemented with a dynamic, linked data structure similar to what we discussed in class. You are not allowed to implement trees with arrays or to use any list, queue, vector, (binary) tree, or heap interface already available in Java. All heap operations must be either in O(1) or $O(\log_2 n)$. You may safely assume for the binary tree and max-heap ADTs that keys are of type integer and values are of type character. So, the use of generics is not required.

- **Requirements.**

a) Specification of the binary tree and max-heap ADTs including comments about assumptions and semantics (especially about the added two heap operations and the change to maxheaps).
b) Pseudo code of your implementation of the binary tree and max-heap ADTs. Keep in mind that Java code will not be considered as pseudo code. Your pseudo code must be on a higher and more abstract level.
c) Well-formatted and documented Java source code and the corresponding executable jar file with at least 20 different but representative examples demonstrating the functionality of your implemented ADTs. These examples should demonstrate all cases of your ADT functionality (e.g., all operations of your ADTs, sufficient sizes of heaps, sufficient number of downheap, upheap, increaseKey, and heapSort operations). You must have separate tests for each ADT but the majority of tests should cover heaps because they are implemented using binary trees.