Data Structures
BITL3

Worksheet 8

Fall 2022

3 November 2022

1. **Warm up:** Answer the following True / False questions.
   (a) Binary search may be performed on a sorted or unsorted list.
   (b) The number of unique paths in a binary tree is equal to the number of its leaves.
   (c) The opposite of "predecessor" is "postdecessor".
   (d) In an algorithm, all the code that is not in a loop does not contribute to the computational complexity of the algorithm.

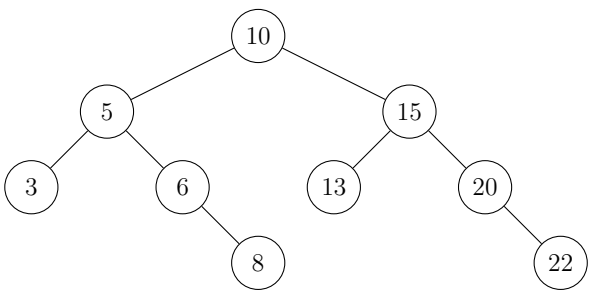
2. Consider the pseudocode on the left, which takes as input a set of numbers $X = \{x_1, \ldots, x_n\}$.

```
1  for i = n, n − 1, . . . , 2:
2    for j = 1, 2, . . . , i − 1:
3      x = x_j
4      if x > x_{j+1}:
5        x_j = x_{j+1}
6        x_{j+1} = x
```

step 0:	3	7	4	1	2
step 1:					
step 2:					
step 3:					
step 4:					
step 5:					
step 6:					
step 7:					

   (a) How many times is line 3 called?
   (b) What is an upper bound on the number of times line 5 is called?
   (c) What is the running time, using Big-O notation, of this algorithm?
   (d) In the boxes on the right above, starting with X as given in step 0, write what X looks like every time the order of its elements changes.
   (e) What do you think the code does to X?

3. In the *binary search tree* below left, perform the operations below right, in the given order. For each insert operation, show the path taken when searching for the spot to insert the key.



   (a) insert 7
   (b) insert 14
   (c) delete 15
   (d) insert 15
   (e) delete 10
   (f) insert 10

   (g) Give the sequence of the nonrecursive preorder traversal of the tree after all the steps above have been applied.

4. This question uses nodes from binary search trees. A file `binary_tree_node.cpp` with this definition and function names in the tasks below is given in ORTUS.

```
1   template <class elemType>
2   struct nodeType {
3       elemType info;
4       nodeType<elemType> *leftLink;
5       nodeType<elemType> *rightLink;
6   };
```

Given a node N in a binary search tree T, the *subtree* at N is the binary search tree with N as a root node, containg all the nodes below N (and forgetting all the nodes above it).

(a) Define a function `nodeCount` that takes in a `nodeType` and returns the number of nodes below it (the input node itself is also counted).

(b) Define a function `leafCount` that takes in a `nodeType` and returns the number of leaves below it (if the input node itself is a leaf, then it is counted).

(c) Define a function `getLevel` that takes in a `nodeType` and returns the largest level index among all nodes.

Recall the *level* of a node N is the number of steps away that N is from the root R. The level of the root node R is 0.