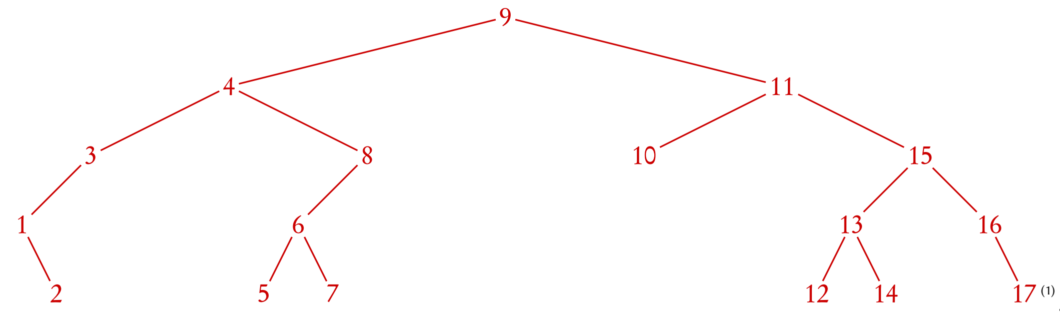
COMP122/22-16 Data Structures and Algorithms	**0 – 65 points**
Binary Search Trees	2022-03-28 *Due Date* — 2022-04-04
Class Code	
Student No.	DO NOT WRITE YOUR NAME

1. The pre-order traversal sequence of a binary search tree is:

$$9, 4, 3, 1, 2, 8, 6, 5, 7, 11, 10, 15, 13, 12, 14, 16, 17.$$

Construct this binary search tree from the pre-order traversal sequence and draw it below. $\boxed{\text{8 points}}$



(1) .

2. We insert 13 items with keys $11, 12, 13, \ldots, 22, 23$ to an empty binary search tree.

a) Write out 4 insertion sequences of these 13 numbers that, according to the insertion algorithm on Slide 10 of Lesson 15, they must all result extreme binary search trees that degenerate to linked lists. $\boxed{\text{6 points}}$

$11, 23, 12, 22, 13, 21, 14, 20, 15, 19, 16, 18, 17$ (2);

$11, 12, 13, 23, 22, 21, 14, 20, 15, 19, 16, 18, 17$ (3);

$23, 22, 11, 12, 13, 21, 14, 20, 15, 19, 16, 18, 17$ (4);

$11, 12, 13, 14, 15, 16, 23, 22, 21, 20, 19, 18, 17$ (5).

b) Write out an insertion sequence that results a binary search tree that is also a complete binary tree. $\boxed{\text{3 points}}$

$18, 14, 12, 11, 13, 16, 15, 17, 22, 20, 19, 21, 23$ (6).

3. Write a *tail-recursive* function *find_gl*(*r*, *x*, *m*) to return the greatest key that is less than x in a binary search tree with root node r. If such a key cannot be found in the tree, the function should return m. The time complexity of the function should be $\mathcal{O}(h)$, where h is the height of the binary search tree.

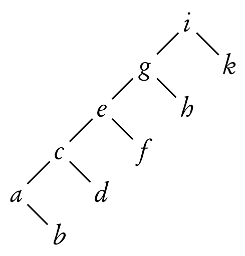
   Note: parameter m is in fact the accumulator. 8 points

```
def find_gl(r, x, m):


    if r is None:                          1
        return m                           1
    elif r.key < x:                        1
        return find_gl(r.right, x, r.key)  3 recursion, subtree, accumulator
    else:
        return find_gl(r.left, x, m)       2 recursion, subtree          (7)
```

4. Suppose r is the root node of a binary search tree. Write a generator function *biter*(*r*, *a*, *b*) to yield all the keys within the range $[a, b)$, that is, each yielded key k satisfies $a \leqslant k < b$. The keys must be yielded in increasing order. If there are m keys yielded, the time complexity of the function should be $\mathcal{O}(h + m)$, where h is the height of the binary search tree. 10 points

```
def biter(r, a, b):


    if r is not None:                      1
        if r.key < a:                      1
            yield from biter(r.right, a, b)  2 yield from recursion, subtree
        elif r.key >= b:                   1
            yield from biter(r.left, a, b)   2 yield from recursion, subtree
        else:
            yield from biter(r.left, a, b)   1
            yield r.key                      1
            yield from biter(r.right, a, b)  1                          (8)
```

5. A binary search tree T is shown below.

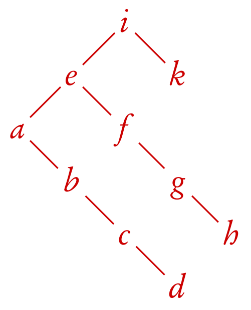


Let $LR(x)$ and $RL(x)$ denote the operations to rotate a binary tree rooted at x "from left to right" and "from right to left", respectively. We list tree rotation steps as a sequence of $LR(\ldots)$ and $RL(\ldots)$ operations.

a) Suppose tree T is rotated by the following operations,

$$LR(g), LR(c), LR(c), LR(g)$$

Draw the resulted tree. $\boxed{\text{6 points}}$

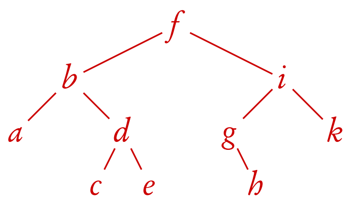


(9)

b) Suppose tree T is rotated by the following operations,

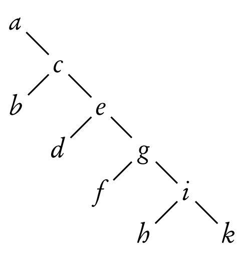
$$RL(e), LR(g), LR(i), RL(a), LR(c), LR(e), RL(c), LR(e)$$

Draw the resulted tree. $\boxed{\text{6 points}}$



(10)

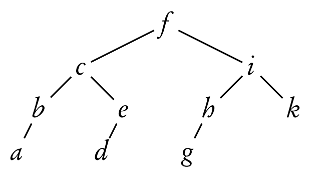
c) List the steps to rotate tree T into the tree shown below, use no more than 8 operations. $\boxed{\text{6 points}}$



$LR(i), LR(g), LR(e), LR(c)$

(11)

d) List the steps to rotate tree T into the tree shown below, use no more than 8 operations. **6 points**



$RL(e), LR(g), LR(i), RL(g), LR(e), RL(a)$ (12).

e) List the steps to rotate tree T into an AVL tree, use no more than 4 operations. **6 points**

$LR(e), LR(i)$ (13).