CSC 212 Midterm 2 - Spring 2015

College of Computer and Information Sciences, King Saud University
Exam Duration: 90 Minutes

23/04/2015

Question 1 [35 points]

1. Write the method *isIncreasing* that takes a Stack<Integer>*st* and checks whether the elements in the stack are ordered in an increasing order from the top to the bottom. If it is the case, it returns true, otherwise, it returns false. The stack *st* should not change after the method. The method's signature is *public boolean isIncreasing(Stack<Integer>st)*.

   Example 1.1. *The stack* st_1 *(top to bottom): 2, 12, 34, 54. Calling* isIncreasing(st_1) *returns true. The stack* st_2 *(top to bottom): 2, 12, 64, 34. Calling* isIncreasing(st_2) *returns false.*

2. Evaluate the following postfix expression using a stack. Redraw the stack **after every push** operation. The final stack should contain the final result. (There should be **9 push** operations): 5 5 6 + 3 2 / ×

3. Solve the following infix expression using stacks. Redraw the stacks after every push. The final stack should contain the final result. (You should redraw the stacks **13** times): 6+1×5-7>2

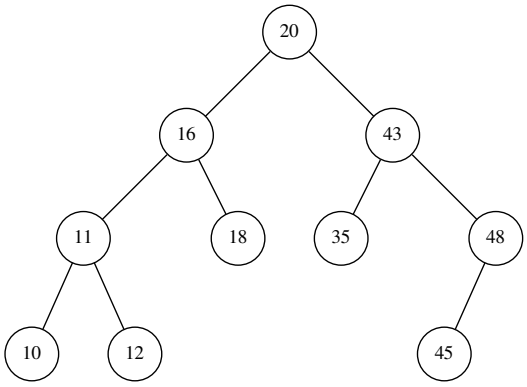
Question 2 [30 points]

1. Write a **recursive** method *printLeaf*, member of the class *BT* (binary tree), that prints the data of all leaf nodes of a non-empty subtree from left to right. The signature of the recursive method is: *private void printLeaf(BTNode<T>t)* (**non-recursive solutions are not accepted**).

   Example 2.1. *For the tree shown in Figure 1, the method displays: E, F, K, I, J.*

2. Write the member method *public int maxKey (int k)* of the class BST (binary search tree) that returns the maximum key of the sub-tree rooted at the node with key k. Assume that k exists.

   Example 2.2. *For the tree below,* maxKey(16) *returns 18,* maxKey(48) *returns 48.*



Question 3 [35 points]

1. Indicate the preorder, inorder and postorder traversals of the tree shown below (write **only** the **number** on the **answer sheet**, for example, **Preorder: 1, Inorder: 2, Postorder: 3**).

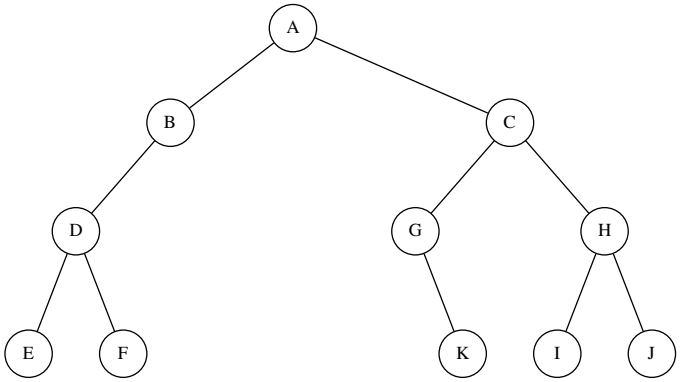


Figure 1: Binary tree

Preorder		Inorder		Postorder	
1.	E F D B A K G C I H J	**1.**	E F D B A K G C I H J	**1.**	E D F B K G I J H C A
2.	A B D E F C K G H I J	**2.**	E D F B A G K C I H J	**2.**	E F D B G K I J H C A
3.	A B E D F C G K I H J	**3.**	D E F B A G K C I H J	**3.**	A B D E F C K G H I J
4.	E F D B A K G I J H C	**4.**	E D F B A G K C H J I	**4.**	E F D B K G I J H C A
5.	A B D E F C G K H I J	**5.**	A B F D E C G K I H J	**5.**	E F D B A K G I J H C
6.	A B F D E C G K I H J	**6.**	E F D B K G I J H C A	**6.**	E F D B A K G C I H J

2. Given the initial BST shown in Figure 2, choose the resulting BST for each of the three sequences of operations shown in the table below (on the **answer sheet**, write **only** the sequence **number** and the corresponding tree **letter**, for example, **1 : a, 2 : b, 3 : c**).

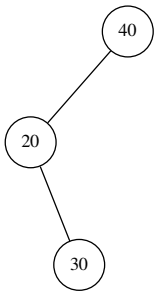
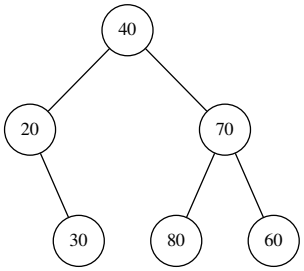
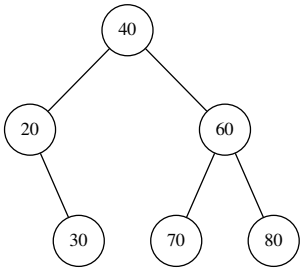


Figure 2: Initial BST.

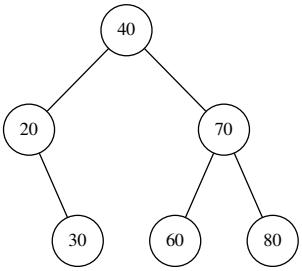
1	**2**	**3**
insert(60)	insert(70)	insert(60)
insert(80)	insert(60)	insert(70)
insert(20)	findKey(70)	findKey(60)
insert(70)	insert(80)	insert(80)



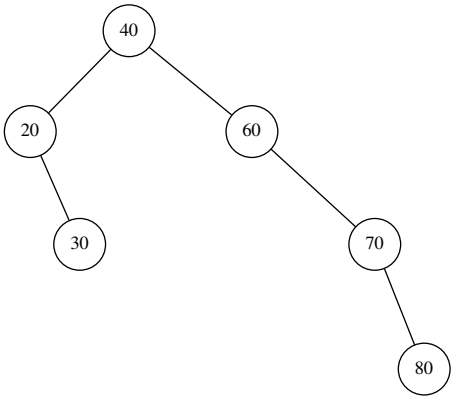
a



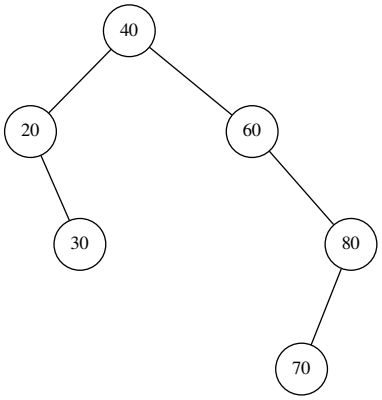
b



c

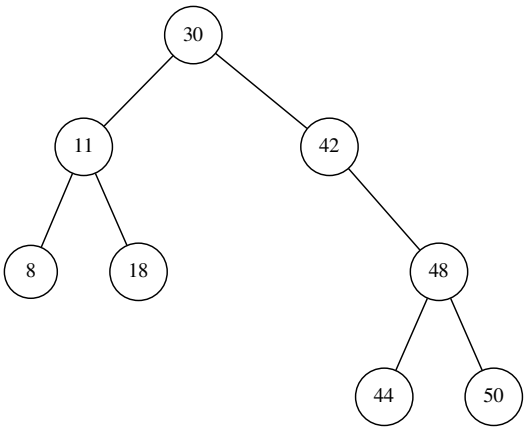


d



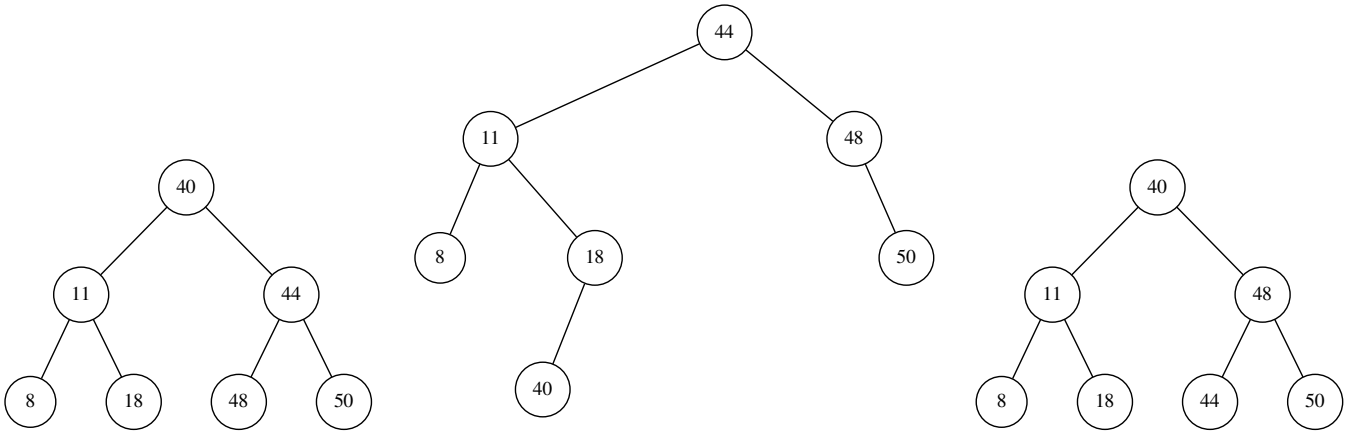
e

3. Given the initial BST shown in Figure 3, choose the resulting BST for each of the three sequences of operations shown in the table below (on the **answer sheet**, write **only** the sequence **number** and the corresponding tree **letter**, for example, **1 : a, 2 : b, 3 : c**).



1	**2**	**3**
removeKey(30)	removeKey(42)	insert(40)
insert(40)	insert(40)	removeKey(42)
removeKey(20)	findKey(42)	findKey(48)
removeKey(42)	removeKey(30)	removeKey(30)

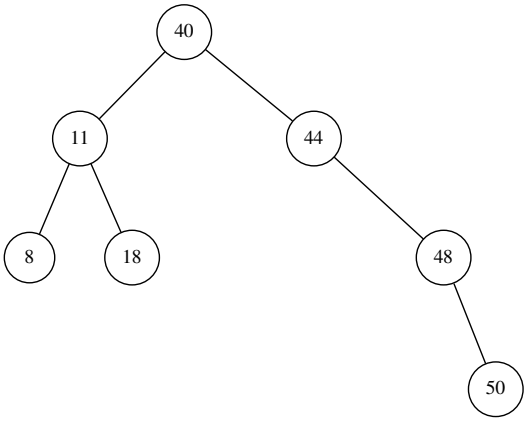
Figure 3: Initial BST.



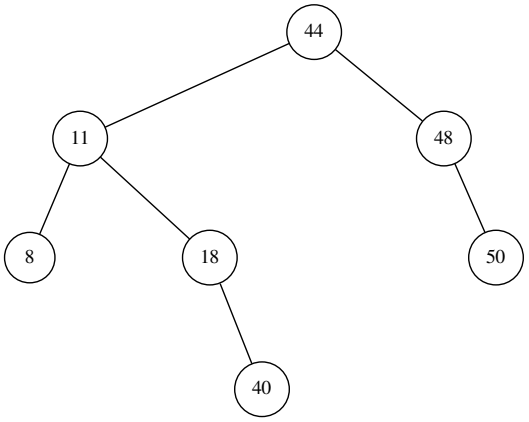
a

b

c



d



e

ADT Stack Specification

- Push (Type e): **requires**: Stack S is not full. **input**: Type e. **results**: Element e is added to the stack as its most recently added elements. **output**: none.
- Pop (Type e): **requires**: Stack S is not empty. **input**: **results**: the most recently arrived element in S is removed and its value assigned to e. **output**: Type e.
- Empty (boolean flag): **requires**: none. **input**: none. **results**: If Stack S is empty then flag is true, otherwise false. **output**: flag.
- Full (boolean flag): **requires**: none. **input**: none. **results**: If S is full then Full is true, otherwise Full is false. **output**: flag.