CSC 212 Final - Fall 2014

College of Computer and Information Sciences, King Saud University
Exam Duration: 3 Hours

03/01/2015

1. **All answers must be written on the answer sheet.**
2. **The use of calculators is prohibited.**

Question 1 [16 points]

1. Give the Big Oh notation for each of the following functions and write them in increasing order of growth rate: **(a)** $3n^2+10n\log(n)$, **(b)** $100n+n^2+n^3$, **(c)** $2^n+n^2+n^2\log(n^2)$, **(d)** $n^2+n\log(n^n)+2^{\log n}$.
2. Write the method *removeBetween*, member of the class *DoubleLinkedList*. The method takes two elements e_1 and e_2, and removes all the elements between the two elements (e_1 and e_2 not included). If e_1 or e_2 or both do not exist, no element will be removed. You can assume the elements to be unique, e_1 always comes before e_2 and that $e_1 \neq e_2$. **Do not call any methods and do not use any auxiliary data structures**. The method signature is: *public void removeBetween(T* e_1*, T* e_2*).*

   Example 1.1. *If the list:* $a \leftrightarrow c \leftrightarrow d \leftrightarrow b \leftrightarrow r \leftrightarrow x$*, then after calling* removeBetween("c", "r")*, the list becomes:* $a \leftrightarrow c \leftrightarrow r \leftrightarrow x$.

. . . . . .

Question 2 [16 points]

1. Write the method remove *removeHP*, member of the class *LinkedPQ* (linked priority queue), that removes all the elements having the highest priority (**Do not call any methods an do not use any auxiliary data structures**). The method signature is: *public void removeHP().*

   Example 2.1. *If pq:* $7 \rightarrow 7 \rightarrow 7 \rightarrow 6 \rightarrow 4 \rightarrow 1$*, then after calling* pq.removeHP()*, pq becomes:* $6 \rightarrow 4 \rightarrow 1$.

2. Write the method *isReverse*, user of the ADT Stack, that takes as input two stacks and returns true if they have the same elements in reverse order (use *equals* to test for elements equality). The two stacks **must not change** after the call. The method signature is: *public <T >boolean isReverse(Stack <T >*st_1*, Stack <T >*st_2*).*

   Example 2.2. *If* st_1 *(top to bottom):* $A \rightarrow B \rightarrow E \rightarrow C$ *and* st_2 *(top to bottom):* $C \rightarrow E \rightarrow B \rightarrow A$*, then calling* isReverse(st_1, st_2) *returns* true.

. . . . . .

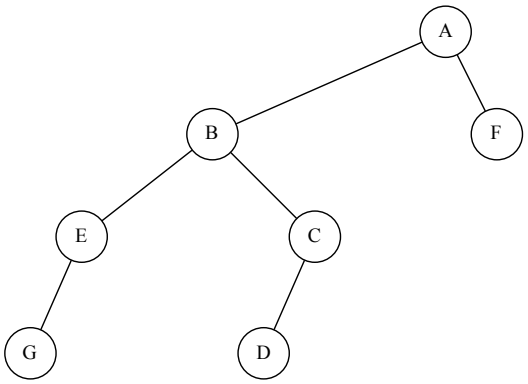
Question 3 [16 points]

1. Write the **recursive** method *twoChildren*, member of the class BT (Binary Tree), which returns the number of nodes with two children. **Do not use any auxiliary data structures and do not call any *BT* methods**. The method signature is *public int twoChildren()*. This method must call the private recursive method *recTwoChildren*. **Important**: Non-recursive solutions are not accepted.

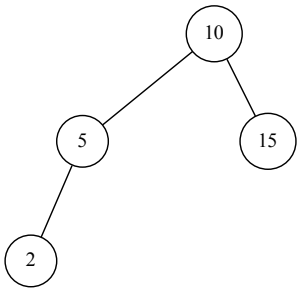
   Example 3.1. *The call* twoChildren() *on the binary tree shown below returns 2.*

2. Write the **recursive** method *isLeaf*, member of the class *BST* (Binary Search Tree), which takes as input an integer *key* and returns true if a node with a key equals to *key* is found and is a leaf node. **Do not use any auxiliary data structures and do not call any BST methods**. The method signature is *public boolean isLeaf(int key)*. This method must call the private recursive method *recIsLeaf* which has the following signature: *private boolean recIsLeaf(int key, BSTNode<T>t)*. **Important**: Non-recursive solutions are not accepted.

   Example 3.2. *On the binary search tree shown below, the call* isLeaf(5) *returns false,* isLeaf(20) *returns false and* isLeaf(15) *returns true.*



Binary tree.



Binary search tree.

. . . . . .

Question 4 [12 points]

1. Consider the following heap represented as an array: 5,10,8,16,12. Choose the correct answer for very operation (all operations are done on the above heap, "-" indicates an empty position).

1. Heap after inserting 6:				
(a) 5,10,8,16,12,6	(b) 6,10,5,16,12,8	(c) 5,6,8,16,10,12	(d) 5,10,6,16,12,8	(e) 6,5,8,16,10,12
2. Heap after inserting 10:				
(a) 5,10,8,16,12	(b) 8,10,5,16,12,10	(c) 5,10,8,16,12,10	(d) 5,10,8,16,12,-,10	(e) 5,12,8,16,10,10
3. Heap after inserting 3:				
(a) 5,10,3,10,12,8	(b) 5,3,8,16,12,10	(c) 3,10,8,16,12,5	(d) 3,5,816,12,10	(e) 3,10,5,16,12,8
4. Heap after inserting 5:				
(a) 5,10,8,16,12,5	(b) 5,10,5,12,16,8	(c) 5,10,5,16,12,8	(d) 5,5,8,16,12,10	(e) 8,10,5,16,12,5
5. Heap after inserting 8:				
(a) 5,10,8,16,12,8	(b) 5,8,8,16,12,10	(c) 5,10,8,16,8,12	(d) 5,10,8,12,16,8	(e) 5,10,810,12,8

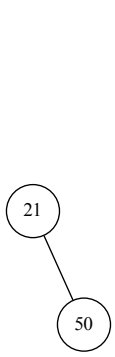
2. Consider the following heap represented as an array: 16,8,12,4,6,4,2. Choose the correct answer for very operation (all operations are done on the above heap, "-" indicates an empty position).

1. After deleting one key:				
(a) 2,8,12,4,6,4	(b) 16,8,12,4,6,4	(c) 12,8,4,4,6,2	(d) 8,4,12,2,6,4	(e) 12,8,2,4,6,4
2. After deleting two keys:				
(a) 4,8,12,4,6	(b) 16,8,12,4,6	(c) 4,8,2,4,6	(d) 8,4,4,2,6	(e) 8,6,4,4,2
3. After deleting three keys:				
(a) 16,-,8,4,6	(b) 4,6,2,4	(c) 12,8,6,4	(d) 6,4,4,2	(e) 6,2,4,4

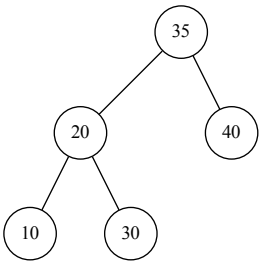
. . . . . .

Question 5 [12 points]

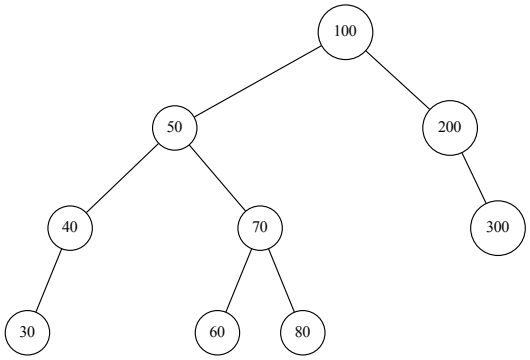
Draw the resulting AVL trees after the following operations:



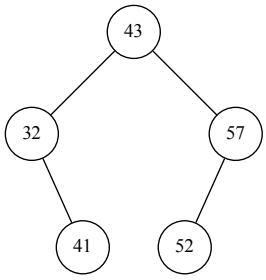
Insert 40



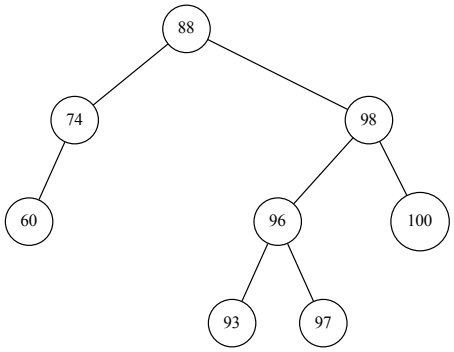
Insert 5



Insert 65



Delete 43

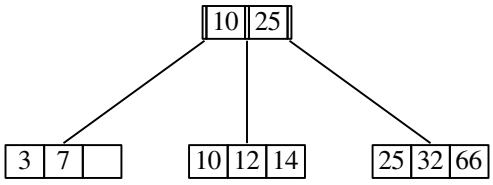


Delete 74

. . . . . .

Question 6 [10 points]

Using the following B+ Tree (m = 3), perform the following operations: Insert 2, Insert 11, Insert 50, and Delete 7 (each operation should be performed on the original tree).



. . . . . .

Question 7 [12 points]

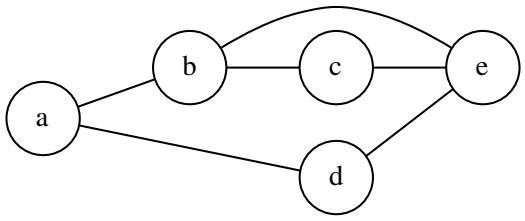
Use the hash function $H(key) = key\%11$ to store the following sequence of keys in a hash table: **98, 55, 14, 0, 60, 23, 43, 13** (the table size is 11).

1. Use linear rehashing (take c=1). **Show the number of probes**.
2. Use external chaining.
3. Use coalesced chaining with a cellar size of 2 (the size of the address region is 11). **Show clearly the links and the final location of epla**.

. . . . . .

Question 8 [6 points]

Draw the adjacency matrix and adjacency list for the following graph (order the nodes alphabetically).



. . . . . .

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.