No. of Pages	**4**
No. of Questions	**5**

BRAC University
Department of Computer Science and Engineering
FINAL EXAMINATION SPRING 2014
CSE 220: Data Structures

Total Marks: 60 **Time Allowed: 2 hrs 30 mins**

- Answer any **FOUR (4)** questions out of **FIVE (5)**
- Figure in bracket [] next to each question indicates marks for that question
- Please read all the Question Carefully
- Understanding the Question is a part of exam

Question 1 (Marks 15):

a) Suppose you have a **resizable Linear Array** of n elements. Write down the **insert()** method which inserts the given element at the mentioned index. Again, note that "n" is the number of elements in the array, which may be less than the capacity. If there is no more space for the new element, you should resize and double the capacity of the array before insertion. [5 Marks]

```
/**
* Inserts the element in the given index of the linear array
* @param a the linear array.
* @param n the number of elements in the array (n <= a.length)
* @param elem the element to be inserted
* @param idx the index to which the insertion should be done.
*/
public static void insert(Object[] a, int n, Object elem, int idx) {
      // TO DO
}
```

b) Suppose you have a **Linear Array** of n elements. Write down the **remove()** method which removes the given element from the array. Again, note that "n" is the number of elements in the array, which may be less than the capacity. [5 Marks]

```
/**
* Removes the given element from the Array
* @param a the linear array.
* @param n the number of elements in the array (n <= a.length)
* @param elem the element to be removed
* @return true if it was successfully removed, or false
* otherwise
*/
public boolean remove(Object[] a, int n, Object elem) {
      // TO DO
}
```

c) Suppose you have a singly linked linear list (NOT dummy-headed), with a reference to the head node. The Node class is shown below:

```
public class Node {
      public Object element; // The element within this Node.
      public Node next; // The reference to the next node in list.
      /**
      * Creates a new Node.
      * @param e the element within
      * @param n the next node
      */
      public Node(Object e, Node n) {
            element = e;
            next = n;
      }
}
```

Write a **recursive method** named **countHowMany()** which returns the occurrence of the given element in the list. [5 Marks]

```
/**
* recursively counts the number of occurrences of the given element in the
* list
* @param head reference to the header node of the list
* @param elem the element of whose occurrences should be counted.
* @return the number of occurrences of elem
*/
public static int countHowMany(Node head, Object elem) {
      //TO DO
}
```

Question 2 (Marks 15):

a) If a letter means **push** and an asterisk means **pop** in a sequence,

Draw the figure of the stack while you carry out the push/pop operations on the following sequence:

E A S * Y * Q U E * S T * * I O * N * A B C *

[3.5 Marks]

b) If a letter means **enqueue** and an asterisk means **dequeue** in a sequence,

Draw the figure of the queue while you carry out the enqueue/dequeue operations on the following sequence:

Q U E * S T * * I O * N * A B C * E A S * Y *

[3.5 Marks]

c) Suppose we have a circular array based Queue implementation:

```
public class Queue {
  private Object[] data;      /* The underlying array container. */
  private int size;           /* Number of elements in the queue. */
  private int front;          /* Index of the front element. */
  /**
   * Adds the given element to the back of the queue.
   * @param e the element to add to the back of the queue.
   * @throws QueueFullException if the queue is full.
   */
  public void enqueue(Object e) throws QueueFullException {
            //TO DO
  }

  * Removes the front element of the queue, and returns it.
  * @throws EmptyQueueException if the queue is empty.
  public Object dequeue() throws EmptyQueueException {
            //TO DO
    }
}
```

i) Write down enqueue () method [4 Marks]
ii) Write down dequeue () method [4 Marks]

Question 3 (Marks 15):

a) What type of data structure do we need while implementing (i) Binary Search and (ii) Insertion Sort? Explain why. [1.5 + 1.5 Marks]

b) Write down the comparison between insertion sort and selection sort [5 Marks]

c) Given array A of non-negative integers, your task is to write a key-indexed search method that takes a key and returns "true" if it exists, or "false" otherwise. You need two methods – one to setup the auxiliary array B, and other to perform the actual search (which uses the auxiliary array B). **Note that you may have multiple entries of the same integer in array A**.

```
/**
* Creates the auxiliary array B for key-indexed search.
* @param A the input array of non-negative integers.
* @return the auxiliary array B needed for key-indexed search
*/
public static int[] setup(int[] A) { // TODO }

/**
* Returns "true" if the key exists in the array, or "false" otherwise.
* @param B the auxiliary array
* @param key the key to search for
* @return true if found, false otherwise
*/
public static boolean search(int[] B, int key) { // TODO }
```

i) Write the setup method that returns the auxiliary array given the input data. [5 Marks]
ii) Write the search method that returns "true" if search key exists in the array, or "false" otherwise [2 Marks]

Question 4 (Marks 15):

a) Write down definitions: Free trees, M-ary trees, Binary Search Tree [3 Marks]
b) Following is the Array representation of a Binary Tree:

	2	7	5	3	6	4	8			9	14			12	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

i) Draw the Binary Tree [3 Marks]
ii) Write down the sequences for: Pre-order, Post-order and In-order Traversal [3 + 3 + 3 Marks]

Question 5 (Marks 15):

a) Suppose we need to insert the sequence (6, 11, 10, 3, 2, 5, 8, 20, 15) into an initially-empty binary search tree (of integers).
i) Show the tree after inserting each key. [5 Marks]
ii) Which one is the successor of key 6? Explain why. [1.5 Marks]
iii) Which one is the predecessor of key 15? Explain why. [1.5 Marks]

b) Consider the following Node class for a binary search tree, BST.

```
public class Node {
  public Object value;
  public Node left;
  public Node right;
  public Node(Object v, Node l, Node r){ value = v; left = l; right = r; }
}

public class BST {
/** The reference to the root node of the tree, null if empty.*/
private Node root;
/** The number of nodes in the tree.*/
private int size;

/**
* counts the number of nodes in the Binary Search Tree
* @param root reference to the root node of the tree
* @return the number of Nodes of the Tree
*/
public static int countHowMany(Node root) { // TO DO }


 /**
 * Finds the parent of the given node.
 * @param node the node whose parent is to be found.
 * @return the parent node, or null if node is the root or tree is empty
 */
  public Node findParent(Node node){// TODO}

}
```

i) Write the countHowMany() method. [3.5 Marks]
ii) Write the findParent() method. [3.5Marks]