INTERNATIONAL SCHOOL OF ENGINEERING

FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2190221 Fundamental Data Structure and Algorithm

Year 2, Second Semester, Midterm Examination March 7, 2019. Time 08:30-11:30

Name..Identification no.........................No. in CR58.........

Important

1. This exam paper has 6 questions. There are 11 pages in total including this page. The total mark is 40.
2. Write your name, ID, and CR 58 number on top of every page.
3. **All questions asked you to write Java code. If you write in some pseudo code and it is understandable, your score will be deducted appropriately.**
4. **Your answer must only be written in this paper.**
5. When the exam finishes, students must stop writing and remain in their seats until all question sheets are collected and the examiners allow students to leave the exam room.
6. A student must sit at his/her desk for at least 45 minutes.
7. A student who wants to leave the exam room early (must follow (5).) must raise his/her hand and wait for the examiner to collect his/her papers. The student must do this in a quiet manner.
8. **No books, lecture notes or written notes of any kinds are allowed in the exam room.**
9. **No calculators are allowed.**
10. A student must not borrow any item from another student in the exam room. If you want to borrow an item, ask the examiner to do it for you.
11. Do not take any part of the question sheet or answering books out of the exam room. All papers are properties of the government of Thailand. Violators of this rule will be prosecuted in a criminal court.
12. A student who violates the rules will be considered as a cheater and will be punished by the following rule:
    - **With implicit evidence or showing intention for cheating, student will receive an F in that subject and will receive an academic suspension for 1 semester.**
    - **With explicit evidence for cheating, student will receive an F in that subject and will receive an academic suspension for 1 year.**

I acknowledge all instructions above. This exam represents **only my own work**. I did not give or receive help on this exam.

Student's signature(.......................................)

Name…………………………………..…..ID…………..……………………………CR58……………………

- You can use methods that you have written in one of the questions in all other questions. .
- Any code not given in this exam must be written by you.
- READ AN ENTIRE QUESTION BEFORE STARTING TO WRITE ANYTHING FOR THAT QUESTION!!!
- The actual questions are on page 10 and 11.

The following codes for list, stack, queue, and binary search tree are usable in all questions:

List (some methods are not given):

```
public interface Iterator {  // This interface is also used by binary search tree
	public boolean hasNext();
	public boolean hasPrevious();

	public int next() throws Exception;
	                    // move iterator to the next position,
	                      // then returns the value at that position.

	public int previous() throws Exception;
	                        // return the value at current position,
	                              // then move the iterator back one position.

	public void set(int value);

}

public class DListIterator implements Iterator {
	DListNode currentNode; // interested position

	DListIterator(DListNode theNode) {
		currentNode = theNode;
	}

	public boolean hasNext() { // always true for circular list.
		return currentNode.nextNode != null;
	}

	public boolean hasPrevious() { // always true for circular list.
		return currentNode.previousNode != null;
	}

	public int next() throws Exception {
		// Throw exception if the next data
		// does not exist.
		if (!hasNext())
			throw new NoSuchElementException();
		currentNode = currentNode.nextNode;
		return currentNode.data;

	}

	public int previous() throws Exception{
		if (!hasPrevious())
			throw new NoSuchElementException();
		int data = currentNode.data;
		currentNode = currentNode.previousNode;
		return data;
	}

	public void set(int value) {
		currentNode.data = value;
```

```
        }
}

class DListNode {
        DListNode(int data) {
                this(data, null, null);
        }

        DListNode(int theElement, DListNode n, DListNode p) {
                data = theElement;
                nextNode = n;
                previousNode = p;
        }

        // Friendly data; accessible by other package routines
        int data;
        DListNode nextNode, previousNode;
}

public class CDLinkedList {
        DListNode header;
        int size;
        static final int HEADERVALUE = -9999999;


        public CDLinkedList() {
                size = 0;
                header = new DListNode(HEADERVALUE);
                makeEmpty();//necessary, otherwise next/previous node will be null
        }

        public boolean isEmpty() {
                return header.nextNode == header;
        }

        public boolean isFull() {
                return false;
        }

        public void makeEmpty() {
                header.nextNode = header;
                header.previousNode = header;
               size =0;
        }

        // put in new data after the position of p.
        public void insert(int value, Iterator p) throws Exception {
                if (p == null || !(p instanceof DListIterator))
                        throw new Exception();
                DListIterator p2 = (DListIterator) p;
                if (p2.currentNode == null)
                        throw new Exception();

                DListIterator p3 = new DListIterator(p2.currentNode.nextNode);
                DListNode n = new DListNode(value, p3.currentNode, p2.currentNode);
                p2.currentNode.nextNode = n;
                p3.currentNode.previousNode = n;
                size++;
        }

        // return position number of value found in the list.
        // otherwise, return -1.
        public int find(int value) throws Exception {
                Iterator itr = new DListIterator(header);
                int index = -1;
                while (itr.hasNext()) {
                        int v = itr.next();
                        index++;
                        DListIterator itr2 = (DListIterator) itr;
                        if (itr2.currentNode == header)
                                return -1;
                        if (v == value)
                                return index; // return the position of value.
                }
                return -1;
        }

        // return data stored at kth position.
        public int findKth(int kthPosition) throws Exception {
                if (kthPosition < 0)
                        throw new Exception();// exit the method if the position is
                // less than the first possible
                // position, throwing exception in the process.
                Iterator itr = new DListIterator(header);
                int index = -1;
                while (itr.hasNext()) {
                        int v = itr.next();
                        index++;
                        DListIterator itr2 = (DListIterator) itr;
                        if (itr2.currentNode == header)
                                throw new Exception();
                        if (index == kthPosition)
                                return v;
                }
                throw new Exception();
        }

        // Return iterator at position before the first position that stores value.
        // If the value is not found, return null.
        public Iterator findPrevious(int value) throws Exception {
                if (isEmpty())
                        return null;
                Iterator itr1 = new DListIterator(header);
                Iterator itr2 = new DListIterator(header);
                int currentData = itr2.next();
                while (currentData != value) {
                        currentData = itr2.next();
                        itr1.next();
                        if (((DListIterator) itr2).currentNode == header)
                                return null;
                }
                if (currentData == value)
                        return itr1;
                return null;
        }
} // end of Linked list code.
```

Stack (the code inside methods are not given, but you are allowed to call the methods):

```
public class Stack {
        public Stack(){} // a constructor that initializes the stack.
        public boolean isEmpty(){} //true if the stack has no data, false otherwise.
        public void makeEmpty(){} // remove all data from the stack.


        //Return data on top of stack.
        //Throw exception if the stack is empty.
        public int top() throws Exception{}

        //Remove data on top of stack.
        //Throw exception if the stack is empty.
        public void pop() throws Exception{}

        //Add new data on top of stack.
        //Throw exception if the operation is somehow
        //unsuccessful.
        public void push(int data) throws Exception{}
}
```

Queue (the code inside methods are not given, but you are allowed to call the methods):

```
public class Queue {//each method is assumed to perform in Θ(1)

        // a constructor which initializes the queue to an empty queue.
        public Queue(){}

        //Return the first data.
        //Throw Exception if the queue is empty.
        public int front() throws Exception{}

        //Return the last data.
        //Throw Exception if the queue is empty.
        public int back() throws Exception{}

        //Remove the first data (return its value too).
        //Throw Exception if the queue is empty.
        public int removeFirst() throws Exception{}

        //Remove the last data (return its value too).
        //Throw Exception if the queue is empty.
        public int removeLast() throws Exception{}

        //Insert new data before the first data.
        //Throw exception if the insert fails for some reason.
        public void insertFirst(int data) throws Exception{}

        //Insert new data after the last data.
        //Throw exception if the insert fails for some reason.
        public void insertLast(int data) throws Exception{}

        //Check if the queue is empty.
        public boolean isEmpty();

        //Check if the queue has no more space to store new data.
        public boolean isFull();

        //Return the number of data currently stored in the queue.
        public int size();
}
```

Binary Search Tree:

```
public class BSTNode {
        int data; // value stored in the node.
        BSTNode left; //pointer to lower left BSTNode.
        BSTNode right; //pointer to lower right BSTNode.
        BSTNode parent; //pointer to the BSTNode above.

        public BSTNode(int data){
                this(data,null,null,null);
        }

        public BSTNode(int data, BSTNode left, BSTNode right, BSTNode parent) {
                this.data = data;
                this.left = left;
                this.right = right;
                this.parent = parent;
        }
}
```

Name…………………………………..…..ID………..…………………………….CR58……………………

```
public class TreeIterator implements Iterator {
        BSTNode currentNode;

        public TreeIterator(BSTNode currentNode) {
                this.currentNode = currentNode;
        }

        public boolean hasNext() {
                BSTNode temp = currentNode;
                if (temp.right != null) {
                        return true;
                }
                BSTNode p = temp.parent;
                while (p != null && p.right == temp) {
                        temp = p;
                        p = temp.parent;
                }
                if (p == null)
                        return false;
                else
                        return true;
        }

        public boolean hasPrevious() {
                BSTNode temp = currentNode;
                if (temp.left != null) {
                        return true;
                }
                BSTNode p = temp.parent;
                while (p != null && p.left == temp) {
                        temp = p;
                        p = temp.parent;
                }
                if (p == null)
                        return false;
                else
                        return true;
        }

        public int next() throws Exception {
                // Throw exception if the next data
                // does not exist.
                BSTNode temp = currentNode;

                if (temp.right != null) {
                        temp = temp.right;
                        while (temp.left != null) {
                                temp = temp.left;
                        }
                } else {
                        BSTNode p = temp.parent;
                        while (p != null && p.right == temp) {
                                temp = p;
                                p = temp.parent;
                        }
                        temp = p;
                }

                if (temp == null) // hasNext() == false
                        throw new NoSuchElementException();
                currentNode = temp;
                return currentNode.data;
        }
        public int previous() throws Exception {
                // Throw exception if the previous data
                // does not exist.
                BSTNode temp = currentNode;
                int d = currentNode.data;

                if (temp.left != null) {
                        temp = temp.left;
                        while (temp.right != null) {
                                temp = temp.right;
                        }
                } else {
                        BSTNode p = temp.parent;
                        while (p != null && p.left == temp) {
                                temp = p;
                                p = temp.parent;
                        }
                        temp = p;
                }

                if (temp == null) // hasPrevious() == false
                        throw new NoSuchElementException();
                currentNode = temp;
                return d;

        }

        public void set(int value) {
                currentNode.data = value;
        }
}

public class BST {
        BSTNode root;
        int size;

        public BST(BSTNode root, int size) {
                this.root = root;
                this.size = size;
        }

        public boolean isEmpty() {
                return size == 0;
        }

        public void makeEmpty() {
                root = null;
                size = 0;
        }

        public Iterator findMin() {
                BSTNode temp = root;
                if(temp == null)
                        return null;
                while (temp.left != null) {
                        temp = temp.left;
                }
                Iterator itr = new TreeIterator(temp);
                return itr;
        }

        public Iterator findMax() {
                BSTNode temp = root;
                if(temp == null)
                        return null;
                while (temp.right != null) {
                        temp = temp.right;
```

```
		}
		Iterator itr = new TreeIterator(temp);
		return itr;
	}

	public Iterator find(int v) {
		BSTNode temp = root;
		if (v == temp.data)
			return new TreeIterator(temp);

		while (temp != null && temp.data != v) {
			if (v < temp.data) {
				temp = temp.left;
			} else {
				temp = temp.right;
			}
		}
		if (temp == null) // not found
			return null;
		return new TreeIterator(temp);
	}

	public Iterator insert(int v) {
		BSTNode parent = null;
		BSTNode temp = root;

		while (temp != null && temp.data != v) {
			if (v < temp.data) {
				parent = temp;
				temp = temp.left;

			} else {
				parent = temp;
				temp = temp.right;

			}
		}

		if (temp == null) {
			BSTNode n = new BSTNode(v, null, null, parent);
			if(parent == null){
				root = n;
			} else if (v < parent.data) {
				parent.left = n;
			} else {
				parent.right = n;
			}
			size++;
			return new TreeIterator(n);
		} else {
			return null;
		}

	}

	public void remove(int v) {
		BSTNode parent = null;
		BSTNode n = root;

		TreeIterator i = (TreeIterator) find(v);
		if (i == null) { // not found, we can not remove it
			return;
		}

		n = i.currentNode;
		parent = n.parent;
```

```
        // otherwise, we remove the value.
        size--;
        if (n.left == null && n.right == null) {// both subtrees are empty
               if (parent == null) {
                      root = null;
               } else if (parent.left == n) {
                      parent.left = null;
                      n.parent = null;
               } else {
                      parent.right = null;
                      n.parent = null;
               }
        } else if (n.left == null && n.right != null) {// only right child
               if (parent == null) {
                      root = n.right;
                      root.parent = null;
                      n.right = null;
               } else if (parent.right == n) {
                      BSTNode q = n.right;
                      q.parent = parent;
                      parent.right = q;
                      n.parent = null;
                      n.right = null;
               } else {// parent.left == n
                      BSTNode q = n.right;
                      q.parent = parent;
                      parent.left = q;
                      n.parent = null;
                      n.right = null;
               }
        } else if (n.right == null && n.left != null) {
               if (parent == null) {
                      root = n.left;
                      root.parent = null;
                      n.left = null;
               } else if (parent.right == n) {
                      BSTNode q = n.left;
                      q.parent = parent;
                      parent.right = q;
                      n.parent = null;
                      n.left = null;
               } else {
                      BSTNode q = n.left;
                      q.parent = parent;
                      parent.left = q;
                      n.parent = null;
                      n.left = null;
               }

        } else {// n has two subtrees
               BSTNode q = n.right;
               TreeIterator itr = findMin(q);
               BSTNode minInSubtree = itr.currentNode;

               n.data = minInSubtree.data;

               BSTNode parentOfMin = minInSubtree.parent;
               if (parentOfMin.left == minInSubtree) {
                      parentOfMin.left = minInSubtree.right;

               } else { // parentOfMin.right == minInSubtree
                      parentOfMin.right = minInSubtree.right;

               }

               if (minInSubtree.right != null) {
                      minInSubtree.right.parent = parentOfMin;
```

```
                              minInSubtree.right = null;
                        }
                        minInSubtree.parent = null;

                  }
            }
}
```

1) (4 marks) For class **CDLinkedList**, write code for method:

```
public void removeAt(Iterator p) throws Exception
```

This method removes data at position marked by p. If p marks header or the list is empty, this method does nothing.

2)

   a) (3 marks) For a linked list (created by class **CDLinkedList**) that contains n non-duplicated numbers, where each number has n digits, is it possible to sort this linked list in O(n log n)? Explain how to do it. You can use any of the given data structures. Your explanation must clearly show how you analyze the asymtotic runtime.

   b) (6.5 marks) Write code for your sort method from question 2.

3) (3 marks) We are trying to implement our own stack using class **Queue**. The class is as follows:

```
public class MyStack{
      Queue q;


}
```

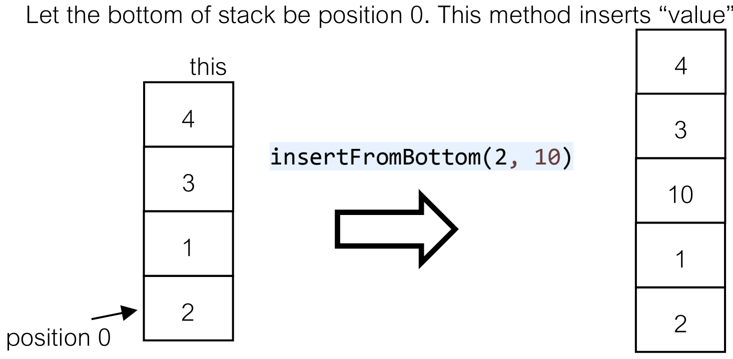
Implement methods push, pop, and top for MyStack.

Note: You are only allowed to use class Queue. No data of other classes (apart from Exception) can be created or used. Array is forbidden.

4) (6 marks) For class **Stack**, write code for method:

```
public void insertFromBottom(int p, int value)
```

Let the bottom of stack be position 0. This method inserts "value" into stack at position p. For example:



Note:

- You are not allowed to create or use any data structure except Stack. Array is forbidden.

- if position is illegal, always insert the value as the top most data in the stack.

5) (8.5 marks) class TestQueue is defined as follows:

```
public class TestQueue{
	public Queue combine(Queue q1, Queue q2){
	
	}
}
```

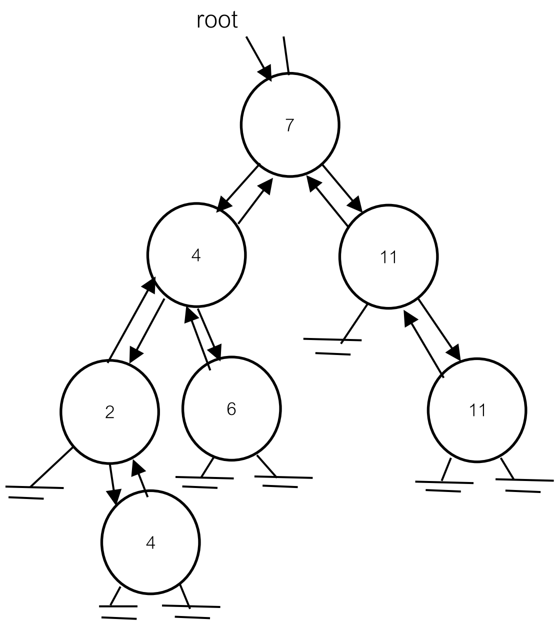
Write code for method **combine**. Method **combine** receives 2 queues. Each queue is assumed to contain sorted data (from small to large). You do not have to check whether the data is sorted, just assume it is. The method returns a sorted queue that contains all the data from both q1 and q2.

For example: if q1 has {1,3,5,8} and q2 has {0,5,7,9}, the returned queue will have {0,1,3,5,5,7,8,9}

Note:

- You are not allowed to create or use any data structure except Queue. Array is forbidden.

6) (9 marks) A modified binary search tree that can store duplicated data may look like:



Is there a better implementation (comparing memory usage and speed) of a binary search tree that allows duplicated data? Please discuss such implementation idea (draw picture!). Compare your idea with the above picture in the following aspects:

- when iterating through the tree.
- when inserting new data.
- when removing a data.