INTERNATIONAL SCHOOL OF ENGINEERING

FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2190221 Fundamental Data Structure and Algorithm

Year 2, Second Semester, Midterm Examination March 02, 2016. Time 13:00-16:00

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

Important

1. This exam paper has 5 questions. There are 10 pages in total including this page. The total mark is 32.
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 the answering book.**
5. When the exam finishes, students must stop writing and remain in their seats until all question sheets and answering books 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. Classes and their methods given to you in any of the question are usable in all questions (unless a question forbids their use).

1. (9 marks) a node for binary search tree (that stores integer) has the following definition:

```
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;
	}

	public static void main(String[] args) {
		BSTNode b = new BSTNode(9);
	}
}
```

A class for binary search tree is:

```
public class BST {
    BSTNode root;
    int size;

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

Write code for method public void insert(int v) of class BST:

- This method put a new value, v, into our binary search tree.

2. (6 marks) Assume we are using stack from class Stack, which has the code of all methods defined in the following Java interface (class Stack also has a working default constructor):

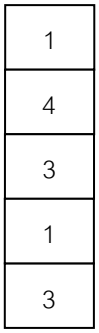
```
public interface MyStack {
	public boolean isEmpty();
	public boolean isFull();
	public void makeEmpty();

	//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;
}
```

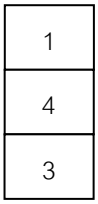
We are using stack in our own **class TestStack**, which is:

```
Class TestStack{
        Stack s;
        public void removeDup(){
                // You have to write code for this method.
        }
}
```

You are to implement method **removeDup**, which removes all duplicated data from **s**. For example, if the original data inside s is



Then stack **s** after the method is called, is



Where 1,4, and 3 can be in any order on the stack, i.e. order does not matter. In other words, method **removeDup** makes **a set** out of existing data on **s**.

- You do not know the internal workings of s, so you can only use methods provided by MyStack interface.
- You are allowed to create primitive type variables.
- You are not allowed to create non-primitive type variables, or any data structure, except Stack(s).

a. (3 marks) Explain, with illustrated example, the inner workings of **removeDup()**. Your explanation should be clear and step-by-step.

b. (3 marks) Write code for **removeDup()**.

3. (8 marks) Assume we have **Class Queue**, a double-ended queue that stores integers. This class **already implements** all methods defined in the following interfaces:

```
public interface MyQueue {
	//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;

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

	// Insert new data as the first data.
	// Throw Exception if the insert is not successful
	// for some unknown reason.
	public void insertFirst(int data) throws Exception;
}
```

We are using a double-ended queue in our own **class TestQueue**, which is:

Name……ID……CR58……

```
Class TestQueue{
        Queue q;
        public int findValue(int i){
                //return a value at i^th position in q (the position number starts from 0).
                // This method is assumed to be completed and working with runtime = O(n).
                // It assume that the value of i must be from 0 to size()-1.
                // Do not code this method!
        }
        public void swap(int p1, int p2){
                // You have to write code for this method.
        }
}
```

You are to implement method **swap**, which exchanges values stored in position p1 and p2 of **q**, where p1 and p2's value must be from 0 to size()-1. For example, if the original data inside q is: {1,2,3,4,5} and swap(1,3) or swap(3,1) is called, then the final data in the queue will be {1,4,3,2,5}.

- You do not know the internal workings of q, so you can only use methods provided by the interfaces.
- You are allowed to create primitive type variables.
- You are not allowed to create non-primitive type variables, or any data structure.

a. (4 marks) Write code for method **swap**. Leave some space at the side to do part b) and c)

b. (2.5 marks) Draw pictures and explain your code. Do the explanation and drawings at the side of your code so that each part of the code is clearly explained.

c. (1.5 marks) Analyze asymptotic runtime for each part of your code and give the overall asymptotic runtime of method **swap**.

4. (5 marks) Code for implementing a circular doubly-linked list is given below:

```
public interface Iterator {
        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;
	}

	// 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();
```

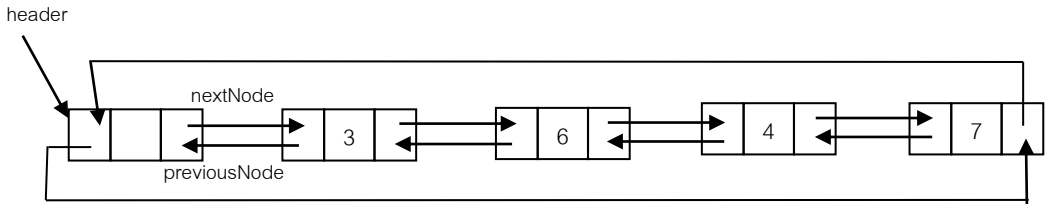
```
                index++;
                DListIterator itr2 = (DListIterator) itr;
                if (itr2.currentNode == header)
                        throw new Exception();
                if (index == kthPosition)
                        return v;
        }
        throw new Exception();
    }

    // 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;

    }

    // remove content at position just after the given iterator. Skip header if
    // found.
    public void remove(Iterator p) {
        if (isEmpty())
                return;
        if (p == null || !(p instanceof DListIterator))
                return;
        DListIterator p2 = (DListIterator) p;
        if (p2.currentNode == null)
                return;
        if (p2.currentNode.nextNode == header)
                p2.currentNode = header;
        if (p2.currentNode.nextNode == null)
                return;
        DListIterator p3 = new
DListIterator(p2.currentNode.nextNode.nextNode);
        p2.currentNode.nextNode = p3.currentNode;
        p3.currentNode.previousNode = p2.currentNode;
        size--;
    }

    // remove the first instance of the given data.
    public void remove(int value) throws Exception {
        Iterator p = findPrevious(value);
        if (p == null)
                return;
        remove(p);
    }

    // remove data at position p.
    // if p points to header or the list is empty, do nothing.
```

```
	public void removeAt(Iterator p) throws Exception{
		if (isEmpty() || p == null
			|| !(p instanceof DListIterator)
			||((DListIterator) p).currentNode == null
			|| ((DListIterator) p).currentNode == header)
			return;

		DListIterator p2 = (DListIterator)(findPrevious(p));
		remove(p2);

	}

	// Print each contact out, one by one.
	// To be completed by students.
	public void printList() throws Exception {
		Iterator itr = new DListIterator(header);
		while (itr.hasNext()) {
			Object data = itr.next();

			System.out.println(data);

		}
	}

	public int size() throws Exception {
		return size;
	}

	// Return iterator pointing to value, or null otherwise.
	// To be completed by students.
	// Not used in class.
	public Iterator findPosition(int value) throws Exception {
		Iterator itr = new DListIterator(header);
		while (itr.hasNext()) {
			int data = itr.next();

			if (data ==value) {
				return itr;
			}
		}

		return null;
	}

	//return iterator pointing to location before position.
	public Iterator findPrevious(Iterator position) throws Exception {
		if (position == null)
			return null;
		if (!(position instanceof DListIterator))
			return null;
		if (((DListIterator) position).currentNode == null)
			return null;

		DListIterator p = ((DListIterator) position);
		DListIterator p2 = new DListIterator(p.currentNode.previousNode);
		return p2;

	}

}
```

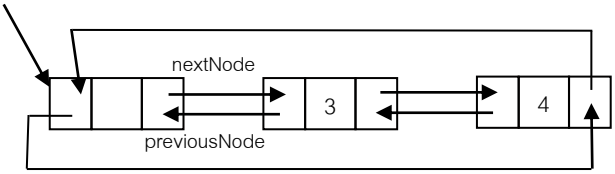
Write code for method **public CDLinkedList partition(int value)** of class **CDLinkedList**:

- This method removes all values greater than **value** from **this** list. The method returns a list containing all the removed values in order from left to right from the **this** list (or empty list of no value is removed).

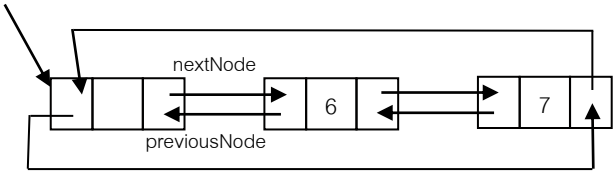
For example, if the original **this** list is:



then calling partition(5) will change **this** to:



And the returned list from the method will be:



5. (4 marks) An array of size n contains integers with values between 0 and 100 inclusive (duplicated values are allowed). Explain (use drawing example) how you sort the values in this array from small to large so that the asymptotic runtime of the sort is $\Theta(n)$. You are allowed to create any data structures and any variables to help you do the sort.