INTERNATIONAL SCHOOL OF ENGINEERING

FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2190221 Fundamental Data Structure and Algorithm

Year 2, Second Semester, Midterm Examination March 08, 2018. Time 08:30-11:30

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

Important

1. This exam paper has 4 questions. There are 9 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 the answer 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 a question are NOT usable in 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!!!

1. (6 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.
}
```

A class for binary search tree is:

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

	public BST() {
		root = null;
		size = 0;
	}
	…
	…
}
```

Class **BST** has the following methods that you can call:

- `TreeIterator findMin(BSTNode n)`: return an iterator marking a node that stores the minimum value in the subtree that has n as its root. If n is null, return null.
- `Iterator find(int v)`: return an iterator marking a node that stores v. If v is not in the tree, return null.
- `boolean isEmpty()`: return true if the tree has no node, false otherwise.
- `Iterator insert(int v)`: add data v into the tree. Existing data in the tree are not added. It returns an iterator marking a node that stores v.

Class **TreeIterator** (it implements interface Iterator) is also available.

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

You can call **TreeIterator's** methods as follows:

- `public boolean hasNext();`
- `public boolean hasPrevious();`

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

- ```
  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.
  ```

Write code for method **public BST greaterThan(int v)** of class BST:

- This method receives a value, **v**.
- It returns a binary search tree that has all its data from **this**, but with values smaller than or equal to v removed.
- **this** must remain unchanged after the method is called.
- The use of array, linked list, stack, and queue are forbidden. If you use one of them, you will not get any mark. You are only allowed to create and use binary search tree(s).

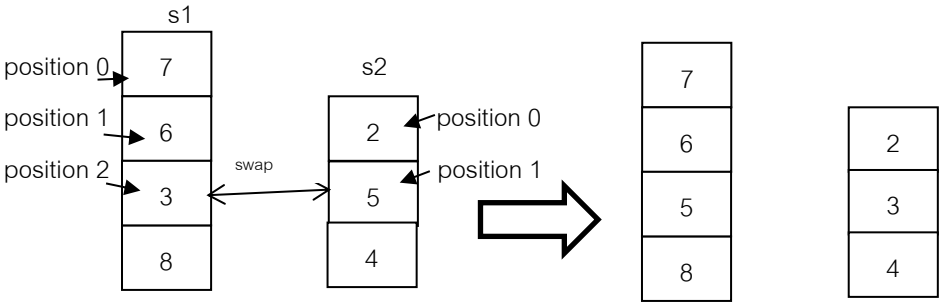
2. (10 marks) Assume we are using stacks of integer, s1 and s2, from **class Stack**, which has the code of all methods defined in the following Java interface

```
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;
}
```

Explain in detail how you can exchange data between position p1 in s1 and position p2 in s2 (Do not write the code. All operations must use methods of Stack).

For example, swapping position 2 of s1 and position 1 of s2 will yield the following:

Name……ID……CR58……



- You are allowed to create primitive type variables.
- You are not allowed to create non-primitive type variables, arrays, lists, or any data structures, except Stack(s). You get no mark if you do not follow this restriction.

3. (7 marks) In implementation of queue using array, you are given class QueueArray below:

```
public class QueueArray {
	protected int[] theArray;
	protected int size; // number of currently stored data.
	protected int front;
}
```

Variable **front** is used to mark the position of the first data (data go round the array).

Write code for method **public void insertLast(int x)** of class QueueArray. This method adds x as the last data in the queue.

4. 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;
	}
```

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

```
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();
```

Need to be filled ! See question 4 a)

```
}

// 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();
```

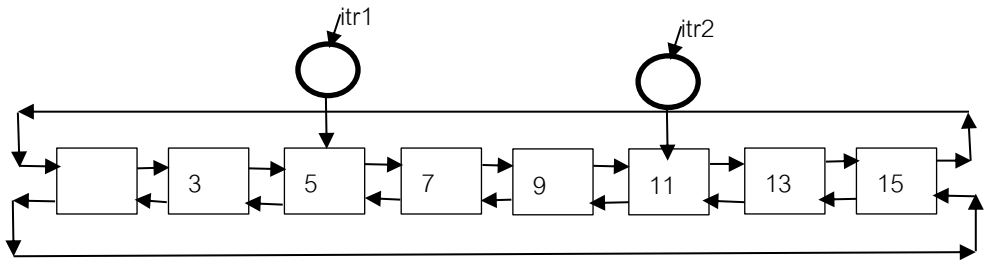
```
	}

	// 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 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;

        }

}
```

Assume your list always store non-duplicated integers:

a. (5 marks) Fill in the code for method **insert** on page 6. Write your answer in the answer book.

b. (6 marks) Assume we have a generic non-empty list that has 2 DListIterators, itr1 and itr2, where itr1 is always to the left of itr2. Both itr1 and itr2 are not at header node. An example list is shown below:



- Explain what you need to do to remove all data from position marked by itr1 upto and including itr2. From the example picture, the list after the removal will contain 3, 13, 15.
    - All removed data must be completely cut off from your result list. That is, the removed nodes must not be able to access node(s) in the result list (and vice versa).

Name……..ID……CR58…….

- Write down the runtime of your explained solution.

c. (6 marks) Write code for method **public void changeToIntersect(CDLinkedList list2)** of class **CDLinkedList**:

- This method changes **this** list to be a list that contains only data that are both in **this** and in **list2**.
    - You are not allowed to create array, list, stack, queue, or tree, or any data structure. You get 0 mark if you do not follow this restriction.

For example, if the original **this** list contains **2,4,6,8** and list2 contains **4,3,6,9**, then your **this** list after the method finishes will contain **4,6**.

Solution

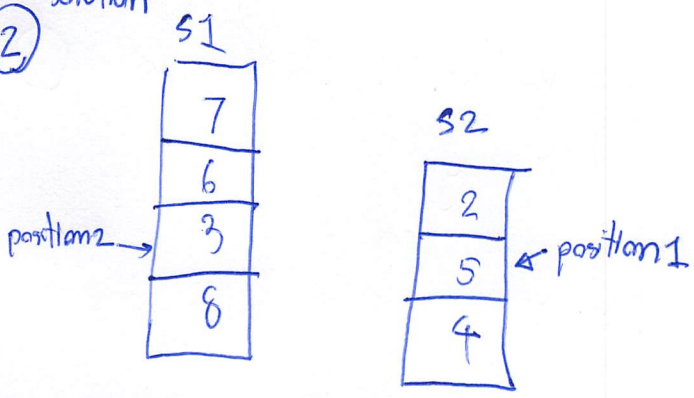
① This problem is solved by

- creating a new binary search tree
- iterate through the original tree and insert values (only the values greater than v) into the new tree.

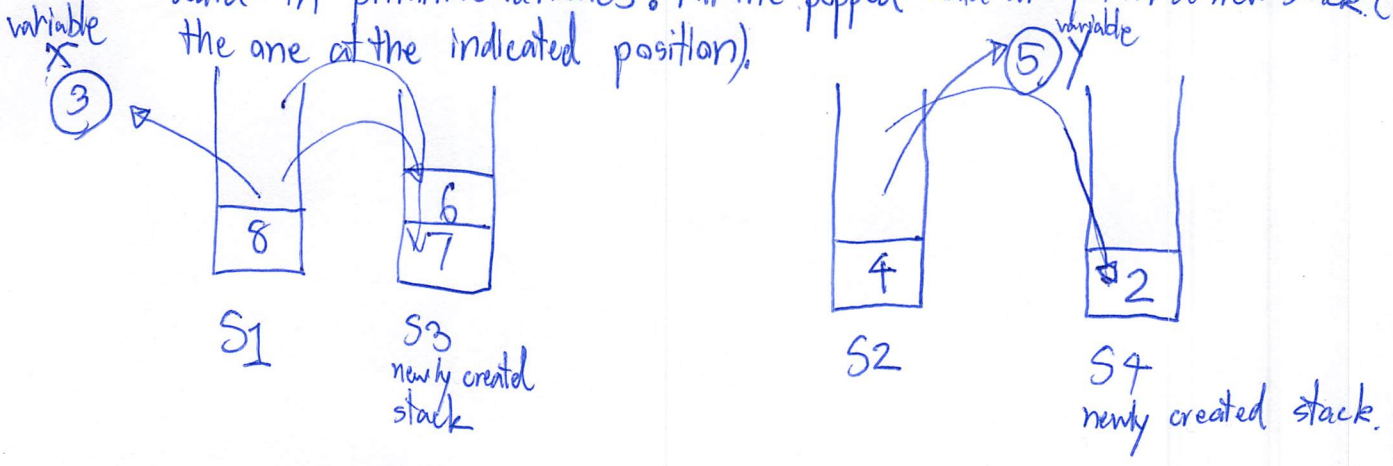
```
public BST greaterThan(int v) {
    if(isEmpty())
        return new BST();          } If the original tree is empty returns an empty tree.
    BST greater = new BST();
    TreeIterator itr = findMin(root);
    int currentData = itr.currentNode.data;   } start from the node that stores the minimum value,
    while(itr.hasNext()){
            if(currentData > v){
                greater.insert(currentData);
            }
            currentData = itr.next();
    }
    if(currentData > v){
            greater.insert(currentData);
    }
    return greater;                } return the new tree.
}
```

go through every node with iterator, including the node with the maximum value.

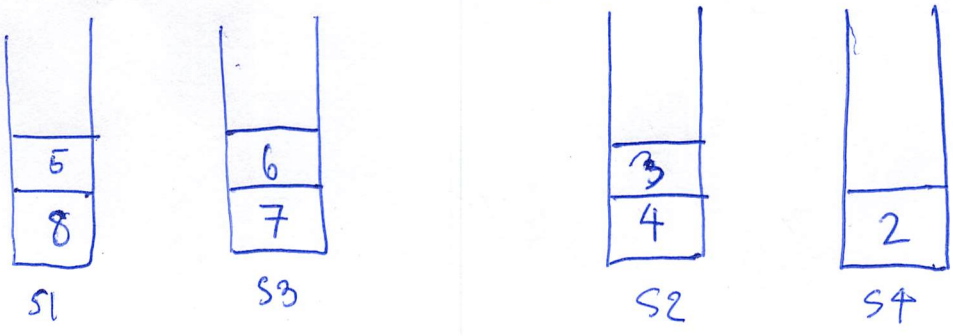
(2) Solution



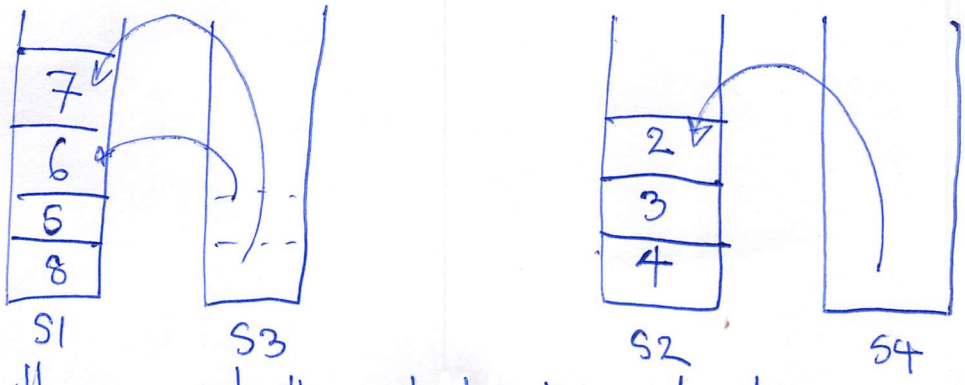
- Pop everything, including the data at the indicated position. Keep the last popped data in primitive variables. All the popped data are put in a new stack. (except the one at the indicated position).



- push variable x into S2, variable y into S1.



- Pop contents of S3 and push them back into S1. Similarly, pop contents from S4 and push them into S2



We will now get the stack with swapped positions as specified in the question.

Solution

③
```
public void insertLast(int x){
    if( size == theArray.length )
        doubleCapacity();
    theArray[(front+size)% theArray.length] = x;
    size++;
}

public void doubleCapacity(){
    int[] temp = new int[theArray.length *2];
    for( int i=0; i<size; i++){
        temp[i] = theArray[(front+i)%theArray.length];
    }
    theArray = temp;
    front = 0;
}
```

} need to expand array if the queue is full.

} Don't forget to make index go round and increase size too.

} make array twice as large

} Copy everything from old array to new array, New array will have its first data at position 0.

} Don't forget to reset the value of front.

524

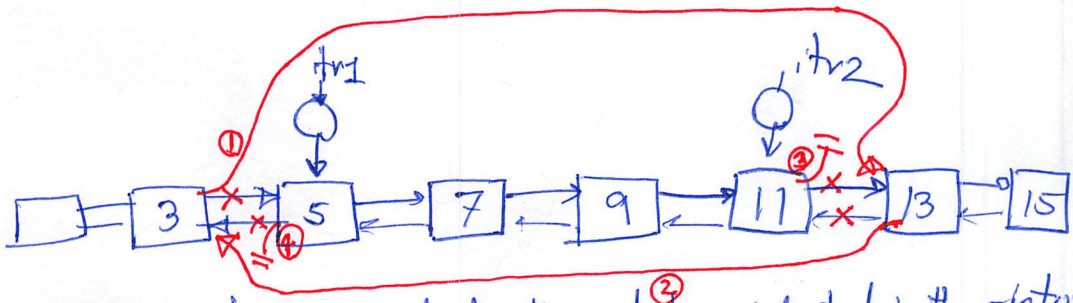
Solution

④

a)

```
DLinkIterator p3 = new DLinkIterator(p2.currentNode.nextNode);
DLinkNode n = new DLinkNode(value, p3.currentNode, p2.currentNode);
p2.currentNode.nextNode = n;
p3.currentNode.previousNode = n;
size++;
```

b)



- Change pointers accordly to the red line indicated in the picture.

Basically, it is ① itr1.currentNode.previousNode.nextNode = itr2.currentNode.nextNode;

② itr2.currentNode.nextNode.previousNode = itr1.currentNode.previousNode;

③ itr2.currentNode.nextNode = null;

④ itr1.currentNode.previousNode = null;

Since there are only 4 steps, and no loop at all, the runtime is Θ(1).

c) The idea:

For each node in this list:
check each node in list2. (have to go through every node until the end, or the same data is found.)

if the same data is found in this and list2
continue to next node of this
else
remove the node in this and get ready to inspect the next node of this

```
public void changeToIntersect(CDLinkedList list2) {
    if (isEmpty() || list2.isEmpty()){
        makeEmpty();
        return;
    }
    DListIterator itr1 = new DListIterator(header);
    itr1.next();
    while ( itr1.currentNode != header){
        DListIterator itr2 = new DListIterator( list2.header.nextNode);
        boolean found=false;
        while ( itr2.currentNode != list2.header) {
            if ( itr2.currentNode.data != itr1.currentNode.data) {
                itr2.next();
            }else{
                found =true;
                break;
            }
        }
        if(found){
            itr1.next();
        }else{
            DListIterator tmp = new DListIterator(itr1.currentNode);
            itr1.next();
            removeAt(tmp);
        }
}
```

if either list is empty, set this to empty list and exit.

start from the first data in this list.

} for each data in this list.

loop in list2 until found data from itr1 or not found data),

if not found, delete data in this list.

Name……id ……..……section…….CR58……

FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2190221 Fundamental Data Structure and Algorithm

Year 2, Second Semester, Final Examination Date 16 May 2018 Time 08.30-11.30 (3 hours)

Important

1. This exam paper has 7 questions. There are 8 pages in total including this page. The total mark is 40.
2. Write your answers in the provided answer book.
3. Write your name and ID on the first page of this exam paper.
4. When the exam finishes, students must stop writing and remain in their seats until all question sheets and answer books are collected and the examiners allow students to leave the exam room.
5. A student must sit at his/her desk for at least 45 minutes.
6. 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.
7. No books, lecture notes or written notes of any kinds are allowed in the exam room.
8. No calculators are allowed.
9. 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.
10. Do not take any part of the question sheet and answer 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.
11. A student who violates rules will punished by the following rule:
    - **Suspected cheaters will get an F in the subject they are suspected to cheat, and will not be able to enroll in the next semester.**
    - **Cheaters will get an F in the subject they are caught cheating, and will not be able to enroll in the next 2 semesters.**

I understand and agree to the given instructions.

Signature(……)

Name……id ……..……section…….CR58……

1. (5 marks) Given a quadratic probing hash table (the table has size 13) for integer. Let *hash(x) = x%TableSize*.

The array for this hash table looks like

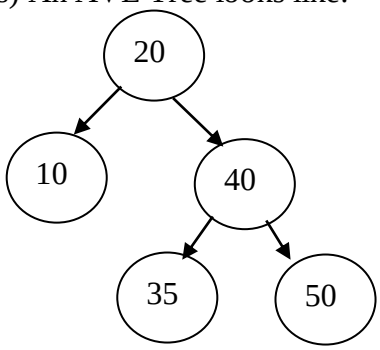
			3				7					

We sequentially do the following actions:

- Add 16
- Add 29
- Delete 7
- Add 20
- Add 12
- Delete 3
- Delete 16

Using DELETED object to mark deleted slot and replacing DELETED object whenever possible, **draw and explain** (step by step) what happens to the array.

2. (5 marks) An AVL Tree looks like:



Draw pictures (if there is a rotation, show the tree before and after each rotation), step by step, what happens when we sequentially add 25, add 60, delete 35, delete 50, and delete 40.

3. (5 marks) A heap of integer that regards small values to be important starts with only one value, 25, inside it. What will the heap look like (in tree form) if we sequentially add 10, add 37, add 6, add 7, and then remove the most important value. Draw the tree before and after each percolation.

Name……id ……..……section…….CR58……

4. (5 marks) We have an array that looks like:

5	3	4	1	2

Draw and explain (step by step) what happens when you do a selection sort (that finds the maximum number to swap each time) on this array. Your picture must show how this array becomes gradually sorted.

5. (5 marks) In this question, we have a heap of integer (small values are more important than large values) constructed using nodes of a binary tree (*class HeapNode*):

```
public class HeapNode{
     int data; //the value that the node stores.
     HeapNode left;
     HeapNode right;
     HeapNode parent; //pointers to other nodes within
                      //the same tree.
}

public class Heap{
     HeapNode root;
}
```

Write code for the following method of class ***Heap***:

public static boolean valueOrdered(HeapNode n)

Node ***n*** is a root of the subtree that we are working on. This method assumes that ***n*** already has a connection structure according to the definition of heap.

This method returns true if the value stored in ***n*** is smaller than other values stored in nodes below it. **AND This fact must be true for every node inside the subtree.**

Name……id ……..……section…….CR58……

6. (10 marks) In this question we are going to write code to do a merge sort on a linked list (It is not going to be fast. Don't worry. We just want to code the algorithm on a linked list). A circular doubly-linked list that stores integer is defined below. (If you need other methods within these given classes, you must write them by yourself.)

```
class DListNode {
	int data;
	DListNode nextNode, previousNode;

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

	DListNode(int theElement, DListNode n, DListNode p) {
		data = theElement;
		nextNode = n;
		previousNode = p;
	}
}//end of class DListNode

public class DListIterator {
      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;
	}

	//move iterator to the next position,
	// then returns the value at that position.
	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;

	}

	// return the value at current position,
	// then move the iterator back one position.
	public int previous() throws Exception{
		if (!hasPrevious())
			throw new NoSuchElementException();
		int data = currentNode.data;
		currentNode = currentNode.previousNode;
		return data;
	}

	public void set(int value) {
```

Name……id ……..……section…….CR58……

```
                currentNode.data = value;
        }
} //end of class DListIterator

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

        public CDLinkedList() {
                size = 0;
                header = new DListNode(HEADERVALUE);

                //necessary, otherwise next/previous node will be null
                makeEmpty();
        }

        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, DListIterator p) throws Exception {
                if (p == null)
                        throw new Exception();
                if (p.currentNode == null)
                        throw new Exception();

                DListIterator p3 = new DListIterator(p.currentNode.nextNode);
                DListNode n = new DListNode(value, p3.currentNode,
p.currentNode);
                p.currentNode.nextNode = n;
                p3.currentNode.previousNode = n;
                size++;
        }
} // end of class CDLinkedList
```

Name……id ……..……section…….CR58……

Write code for the following new methods of ***CDLinkedList***:

i) **(**5 marks**)** ***public static CDLinkedList merge(CDLinkedList l1, CDLinkedList l2).***

This method takes two inputs. Each input is a linked list that stores data sorted from small to large. It returns a new linked list that contains all data from both l1 and l2, and all data in this new list are sorted from small to large. **l1 and l2 MUST remain unchanged**.

ii) (3 marks) ***public static CDLinkedList split(int i, CDLinkedList list).***

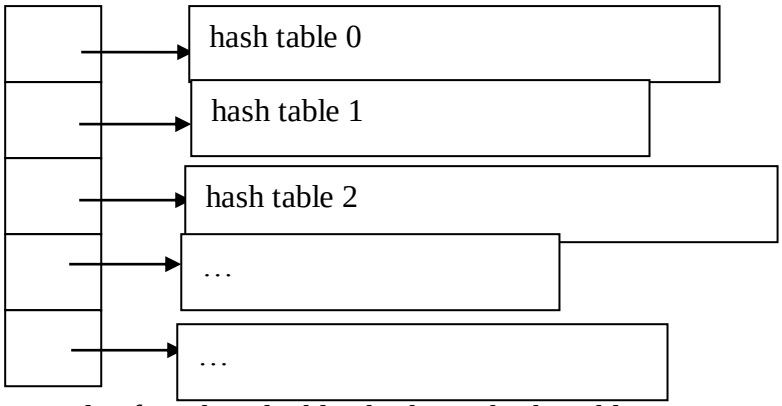
If i is 1, this method returns a new linked list with all its values coming from the left half of ***list***. Left and right half are divided evenly.

If i is 2, this method returns a new linked list with all its values coming from the right half of ***list***. Left and right half are divided evenly.

list must remain unchanged after this method finishes.

iii) (2 marks) ***public static CDLinkedList mergeSort(CDLinkedList l).*** This method takes a linked list and performs a mergesort on the list. It returns the sorted list as a result.

7. (5 marks) A special kind of separate chaining hash table is implemented using an array of open addressing (double hashing) hash tables. A concept table is shown below:



The code for the double hashing hash table is given below (**Class *DoubleHashing* <u>does not</u> have some methods though**). You can call all the given methods and access all the given variables.

```
1: public class Utility{
2:      public static boolean isPrime(int n){
3:       //Check whether n is prime
4:      }
5:
6:      public static int nextPrime(int n){
7:       //return n if n is prime.
8:       // Otherwise return the next prime number after n.
9:      }
```

Name……id ……..……section…….CR58……

```
10:   } //end of class Utility
```

```
1: public class OpenAddressing{
2:      static int DEFAULT_SIZE = 101;
3:      static final Object DELETED = new Object();
4:      static int MAXFACTOR = 0.5;
5:      int currentSize =0;
6:      Object[] array;
7:
8:      public OpenAddressing(){
9:         this(DEFAULT_SIZE);
10:      }
11:
12:      public OpenAddressing(int size){
13:         int nextPrimeSize = Utility.nextPrime(size);
14:         array = new Object[nextPrimeSize];
15:     }
16:   } // end of class OpenAddressing.
```

```
1: class DoubleHashing extends OpenAddressing
2:     static int MAXFACTOR = 0.75;
3:     int occupiedSlots = 0;
4:
5:     public DoubleHashing(){
6:        this(DEFAULT_SIZE);
7:     }
8:
9:     public DoubleHashing(int size){
10:          super(size);
11:     }
12:
13:     public int hash(Object data){
14:       int hashValue = data.hashCode();
15:       int abs = Math.abs(hashValue);
16:       return abs%array.length;
17:     }
18:
19:     public int hash2(Object data){
20:       //Return a value calculated from data.
21:       //It uses a different calculation from hash(Object data).
22:     }
23:   } //end of class DoubleHashing
```

The class for our special separate chaining hash table is defined below. You can call all the given methods and access all the given variables. **The class is not complete, however.**

```
1: public class SepChaining{
2:    static int DEFAULT_SIZE = 101;
3:    static int MAXLOAD = 2;
4:    DoubleHashing[] tables;
5:    int currentSize =0;
6:
7:    public SepChaining(){
8:       this(DEFAULT_SIZE);
9:    }
10:
11:   public SepChaining(int size){
12:       int nextPrimeSize = Utility.nextPrime(size);
```

Name……id ……..……section…….CR58……

```
13:         tables = new DoubleHashing[nextPrimeSize];
14:         for(int i=0; i<tables.length; i++){
15:             tables[i] = new DoubleHasing();
16:         }
17:     }
18:
19:     public int hashSpecial(Object data){
20:         //This is the hash function for this separate chaining
21:         // hash table. It returns the position within tables
22:         // that you need to put data in.
23:     }
24:
25:     } // end of class SepChaining
```

Write code for method ***public void remove(Object data)*** of class ***SepChaining***.

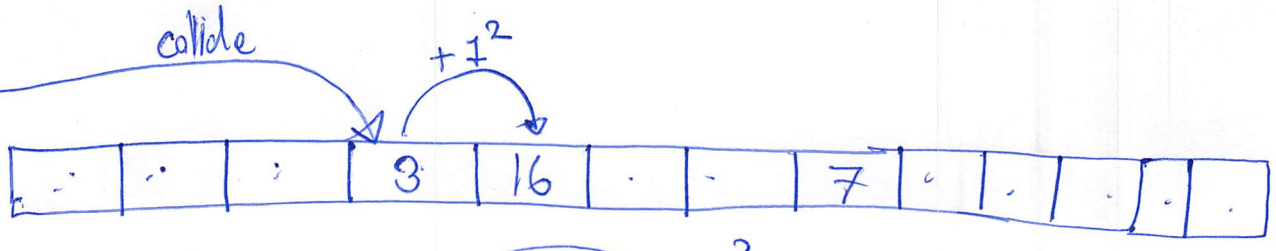
This method does nothing if ***data*** is not in our separate chaining hash table. But if the data is in the hash table, it removes the data using lazy deletion.

*To compare ***this*** object with object ***o***, use method ***boolean equals(Object o)*** which returns true if ***this*** is equal to ***o***, and returns false otherwise.

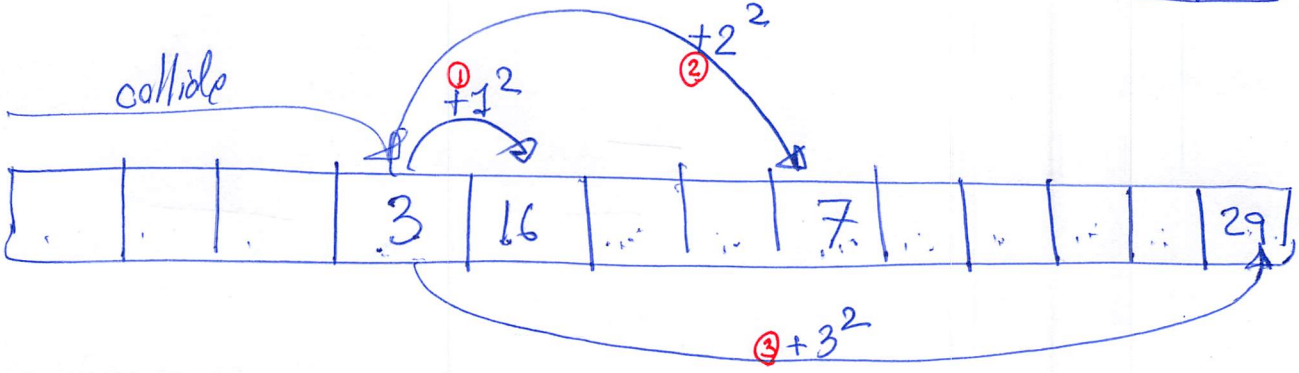
*If any other methods or variables not given in this question are needed, you must write their code by yourself.

Final Exam.
Solution

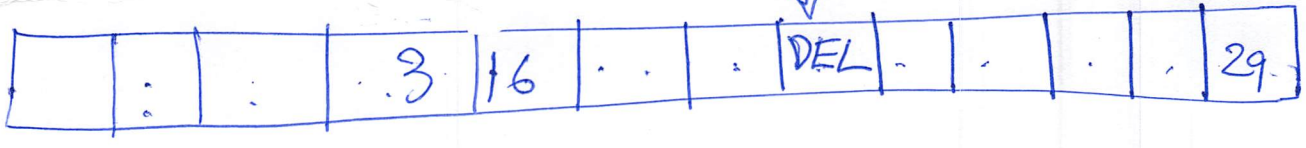
① ~~Add 16~~



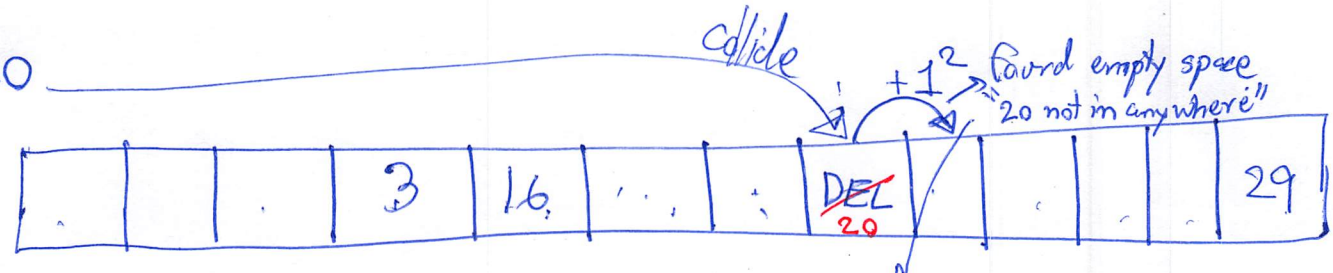
Add 29



Delete 7



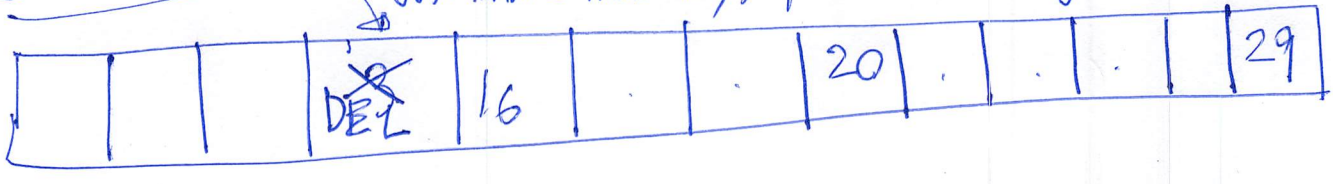
Add 20



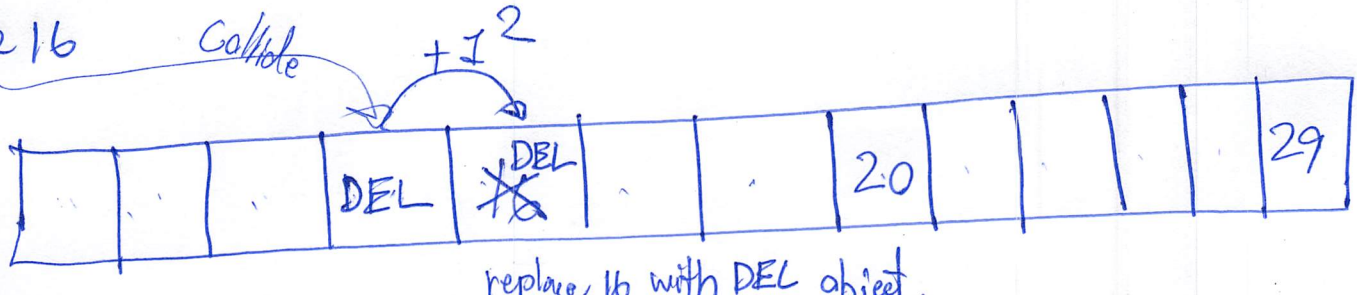
So, replace DEL with 20.

Delete 3

Just find 3 immediately, replace with DEL object.



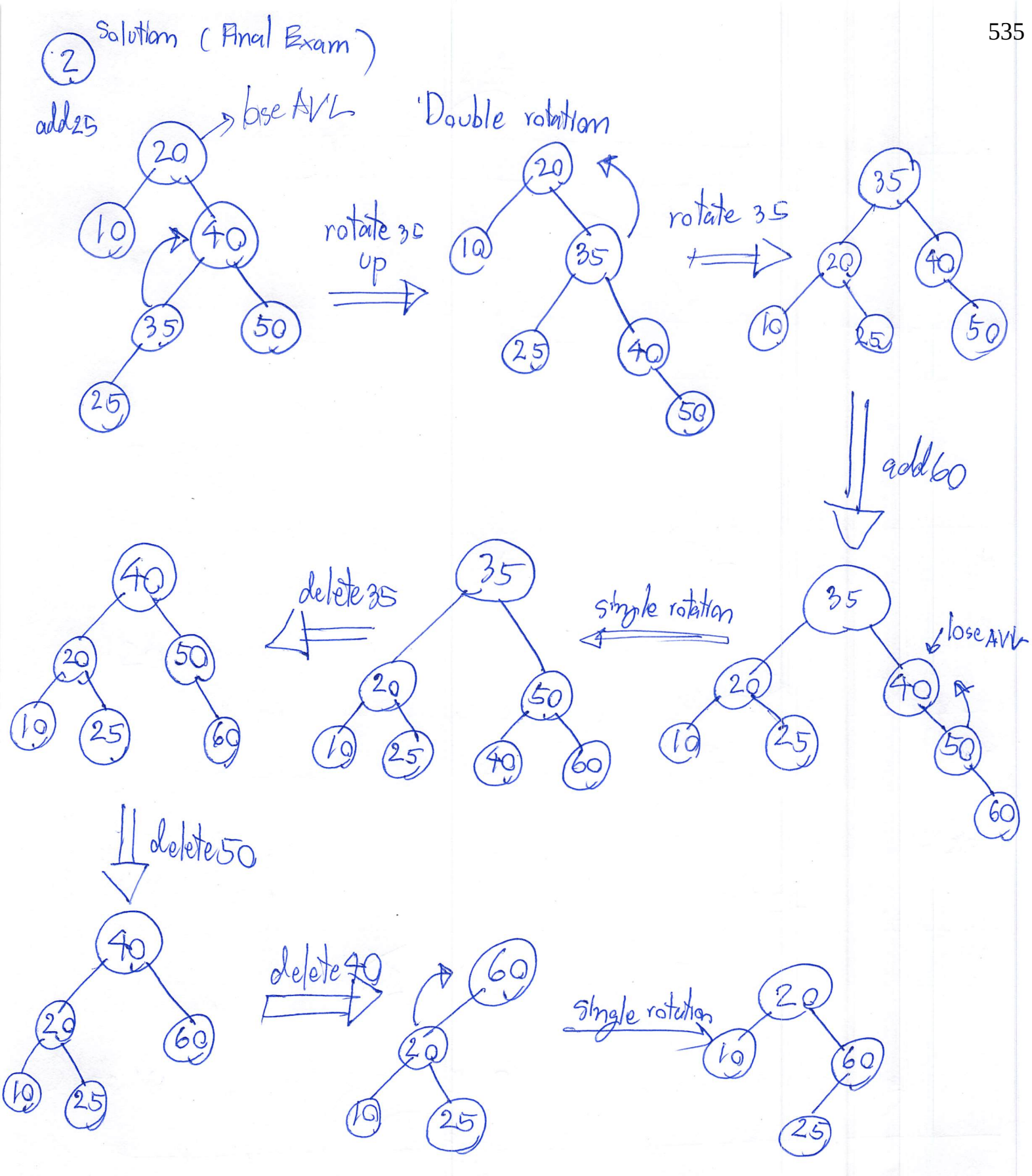
Delete 16



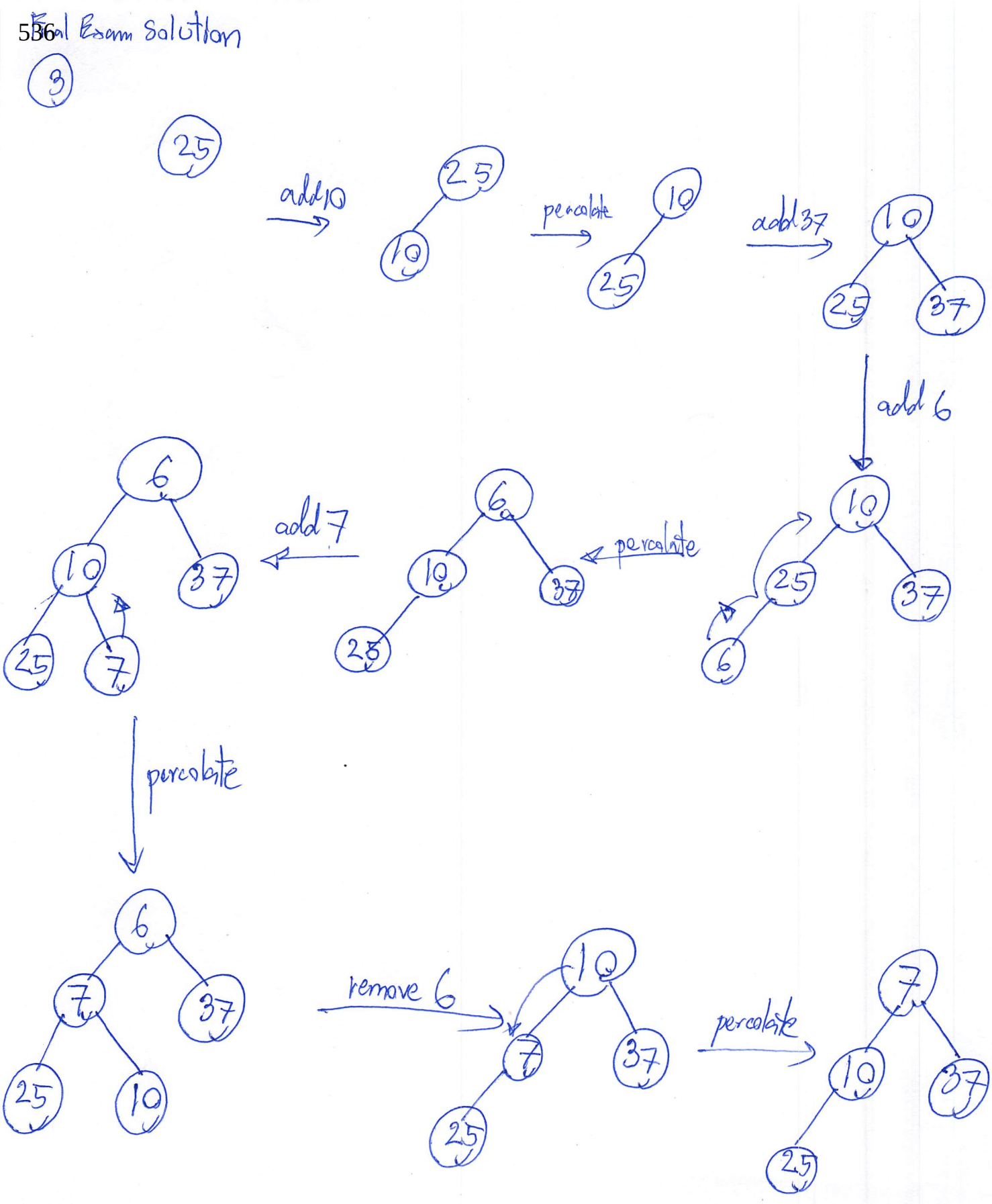
replace 16 with DEL object.

A complete drawing can explain the solution without much writing.

(2) Solution (Final Exam)

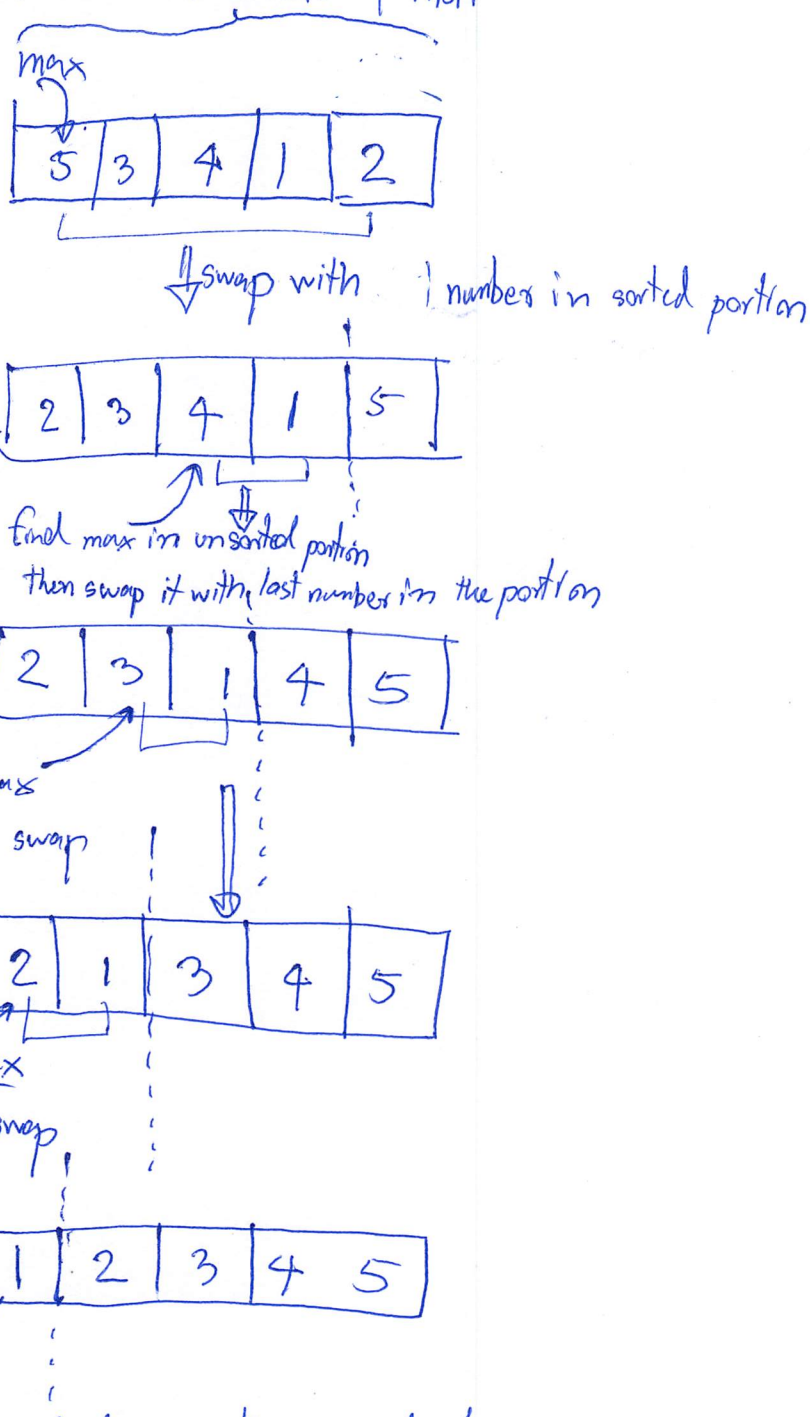


Final Exam Solution



Final Exam Solution. unsorted portion

(4)



only 1 number in the unsorted portion. So the array is considered sorted.

Final Exam Solution

⑤
```
public static boolean valueOrdered(HeapNode n){
    if(n==null)
        return true;

    if(n.left==null && n.right==null)
        return true;
    if(n.left!=null && n.right==null)
        return ((n.data < n.left.data) && valueOrdered(n.left));

    return (n.data < n.left.data) && (n.data < n.right.data) &&
        valueOrdered(n.left) && valueOrdered(n.right);
}
```

Final Exam solution

⑥ i)

```
public static CDLinkedList merge(CDLinkedList l1, CDLinkedList l2) {
    DListIterator itr1 = new DListIterator(l1.header.nextNode);
    DListIterator itr2 = new DListIterator(l2.header.nextNode);
    CDLinkedList l3 = new CDLinkedList();
    DListIterator itr3 = new DListIterator(l3.header);
    while (itr1.currentNode != l1.header && itr2.currentNode != l2.header) {
        if(itr1.currentNode.data <= itr2.currentNode.data) {
            l3.insert(itr1.currentNode.data, itr3);
            itr1.next();
            itr3.next();
        } else {
            l3.insert(itr2.currentNode.data, itr3);
            itr2.next();
            itr3.next();
        }
    }
    while (itr1.currentNode != l1.header) {
        l3.insert(itr1.currentNode.data, itr3);
        itr1.next();
        itr3.next();
    }
    while (itr2.currentNode != l2.header) {
        l3.insert(itr2.currentNode.data, itr3);
        itr2.next();
        itr3.next();
    }
    return l3;
}
```

ii) public static CDLinkedList split(int i, CDLinkedList list) {

```
int half = list.size/2;
DListIterator itr = new DListIterator(list.header);
CDLinkedList leftHalf = new CDLinkedList();
CDLinkedList rightHalf = new CDLinkedList();
for (int j=1; j<=half; j++) {
  if(i==1) {
    leftHalf.insert(itr.next(), leftHalf.header);
  } else if (i==2){
    itr.next();
  }
}

for(int k=half+1; k<=list.size; k++) {
  if(i==1) {
    itr.next();
  } else if(i==2) {
    rightHalf.insert(itr.next(), rightHalf.header);
  }
}
if(i==1) return leftHalf;
if(i==2) return rightHalf;
```

order in the result list does not matter here,

}

Final Exam Solutions

iii) public static CDLinkedList mergeSort(CDLinkedList l) {
if(l.isEmpty()) return new CDLinkedList();
CDLinkedList left = split(1, l);
CDLinkedList right = split(2, l);
CDLinkedList sortedLeft = mergeSort(left);
CDLinkedList sortedRight = mergeSort(right);
CDLinkedList result = merge(sortedLeft, sortedRight);
return result;
}

just need an overall mergesort algorithm.

Final Exam Solution

(7)

```
public void remove (Object data) {
    int indexTables = hashSpecial(data);
    DoubleHashy d = tables[indexTables];
    int indexDoubleHashy = d.hash(data);

    while (!d.array[indexDoubleHashy].equals(data) &&
           d.array[indexDoubleHashy] != null) {
        indexDoubleHashy = indexDoubleHashy + d.hash2(data);
    }
    if (d.array[indexDoubleHashy].equals(data)) {
        d.array[indexDoubleHashy] = DELETED;
        currentSize--;
        return;
    }
}
```

find position in separate chaining first.

loop until find data or find null in doubleHashy hash table

lazy deletion