TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Student ID: .....

EXAMINATIONS — 2012

MID YEAR

COMP103
Introduction to
Data Structures and Algorithms

SOLUTIONS

Time Allowed: 3 Hours

Instructions:

1. Attempt **all** of the questions.
2. *Read each question carefully before attempting it.* (Suggestion: You do not have to answer the questions in the order shown. Do the questions you find easiest first.)
3. This examination will be marked out of **180** marks, so allocate approximately one minute per mark.
4. Write your answers in the boxes in this test paper and hand in all sheets.
5. Non-electronic translation dictionaries are permitted.
6. Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination.
7. Documentation on some relevant Java classes, interfaces, and exceptions can be found at the end of the paper.

Questions	**Marks**
1. Collections	[30]
2. Sorting	[30]
3. Recursion and Linked Lists	[25]
4. Trees	[35]
5. Heaps	[20]
6. Hashing	[30]
7. Final Hard Question	[10]

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Student ID: .....

Question 1. Collections [30 marks]

(a) [3 marks] You are writing a program that needs to store a collection of values with duplicates. Java does not have a class called Bag in its java.util package. Which java.util class would you chose to use instead? Justify your answer.

ArrayList that is treated as Collection type.

(b) [3 marks] What is the best general purpose implementation of a set in java.util? Justify your answer.

HashSet as with hashing all getting and setting operations will be $O(1)$.

(c) [3 marks] State *two* ways in which *merge sort* is better than *quick sort*.

Merge sort is stable. Worst case behaviour of merge sort is $O(nlog)$ while for quick sort it is $O(n^2)$.

(d) [3 marks] State *two* ways in which *quick sort* is better than *merge sort*.

QS is faster on average than MS. QS can be done *in place* without the need for additional array(s).

(Question 1 continued on next page)

(Question 1 continued)

Assume that the class MyMap below is intended to implement a map. It uses an ArrayList of Entry objects each containing a key and value pair.

(e) [8 marks] Initialise the data field as appropriate and implement a put method that puts a mapping from key to value into the map and replaces the existing mapping if key is already in this map.

```
import java. util .*;
public class MyMap<K, V> {
    class Entry<K, V> {
        public K key;
        public V value;
        public Entry(K k, V v) { this.key = k; this.value = v; }
    }
    ArrayList<Entry<K, V>> data =
                                                new ArrayList<Entry<K, V>>();



    public void put(K key, V value) {

        for (Entry<K, V> e : data) {
            if (e.key.equals(key)) {
                e.value = value;
                return;
            }
        }
        Entry<K, V> e = new Entry<K, V>(key, value);
        data.add(e);









    }
```

(Question 1 continued on next page)

(Question 1 continued)

(f) [5 marks] Implement the remove method that removes a mapping for a given key if there is one.

```
    public void remove(K key) {









        for (Entry<K, V> e : data) {
            if (e.key.equals(key)) {
                data.remove(e);
                return;
            }
        }

    }
}
```

(g) [5 marks] State and justify the cost of *your version* of MyMap's put and remove methods.

All two should be around $O(n)$ as they potentially need to check every single entry in the map.

SPARE PAGE FOR EXTRA ANSWERS

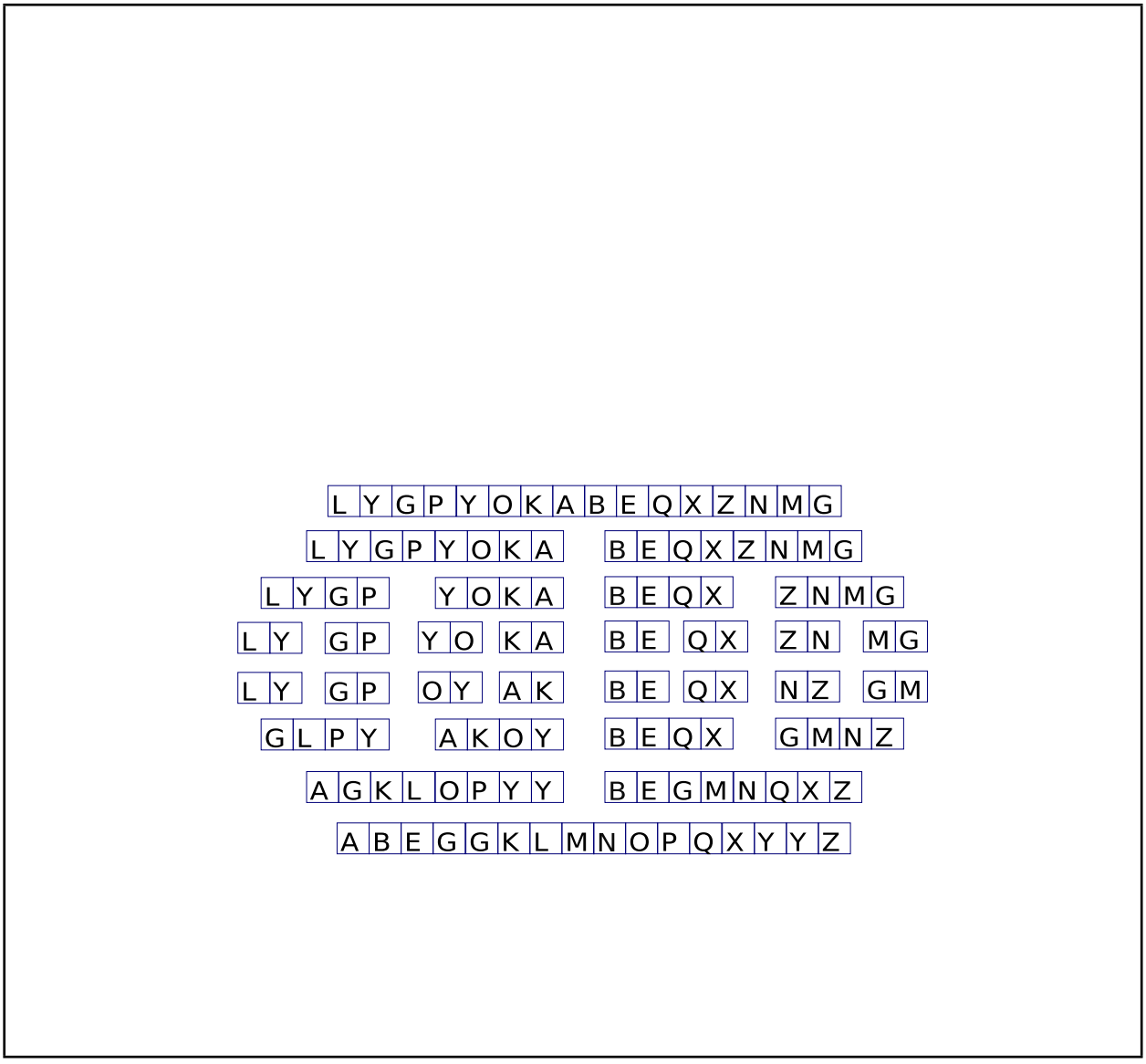
Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 2. Sorting

[30 marks]

(a) [10 marks] Consider the following array of letters. Draw a diagram of merge sort working on this array. Show how it divides and combines the parts resulting in the sorted version.

L	Y	G	P	Y	O	K	A	B	E	Q	X	Z	N	M	G



(Question 2 continued on next page)

(Question 2 continued)

Suppose quick sort is called on the following array. Assume the pivot is chosen to be the element in the first position of the subarray being processed (*not the median of first, middle, and last value as was done in the lectures!*). The first step is shown, you need to show the second and the third steps, circling the pivots involved as shown.

L	Y	G	P	Y	O	K	A	B	E	Q	X	Z	N	M	G

After the first split (note the circle around the pivot):

G	K	A	B	E	G	L	Y	P	Y	O	Q	X	Z	N	M

(b) [5 marks] After the next split of both left and right parts (circle pivot used in left part and circle pivot used in right part):

(c) [5 marks] After the next split of each of the four parts above (circle the four pivots):

You *do not* need to show any further steps.

Answer diagram:

L	Y	G	P	Y	O	K	A	B	E	Q	X	Z	N	M	G
G	K	A	B	E	G	L	Y	P	Y	O	Q	X	Z	N	M
A	B	E	G	G	K	L	P	O	Q	X	N	M	Y	Y	Z
A	B	E	G	G	K	L	O	N	M	P	Q	X	Y	Y	Z

(Question 2 continued on next page)

(Question 2 continued)

(d) [10 marks] Here is our array for one last time. Show how it changes as you sort it using selection sort. For every step clearly identify the sorted and unsorted parts. *STOP when half of the array is sorted.*

L	Y	G	P	Y	O	K	A	B	E	Q	X	Z	N	M	G

No answer given as this question is considered obvious.

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 3. Recursion and Linked Lists [25 marks]

Use recursion to implement a LinkedNode class representing a single node of a linked list. The fields and the constructor are given to you below. You will need to complete three additional recursive methods: size, add, remove.

```
public class LinkedNode<E> {
    public E item;
    public LinkedNode<E> next;

    public LinkedNode(E item, LinkedNode<E> next) {
        this.item = item; this.next = next;
    }
```

(a) [3 marks] Implement a recursive method that returns the size of a linked list when called on the head node of the list.

```
    public int size() {




        if (this.next == null)
            return 1;
        else
            return 1 + this.next.size();
    }
```

(Question 3 continued on next page)

(b) [4 marks] Implement a recursive method that adds an item to the linked list when called on the head node of a linked list. If the item is already present in the list, it should return false and not add the item. If the item is not present, it should add the item and return true.

```
    public boolean add(E item) {



        if (this.next == null) {
            this.next = new LinkedNode<E>(item, null);
            return true;
        } else if (this.item.equals(item)) {
            return false;
        } else {
            return this.next.add(item);
        }


    }
```

(c) [3 marks] Implement a recursive method that removes an item from a linked list when called on the head node of a linked list. The method should return false if and only if the item is not present. *You can assume that the item is NOT in the first node of the list.*

```
    public boolean remove(E item) {




        if (this.next == null) {
            return false;
        } else if (this.next.item.equals(item)) {
            this.next = this.next.next;
            return true;
        } else {
            return this.next.remove(item);
        }
    }
}
```

(Question 3 continued on next page)

(Question 3 continued)

(d) [5 marks] Give a better version of the remove method in the LinkedNode class that also handles the case where the item is in the first node. Assume that the list *contains at least 2 items.*

```
public boolean remove(E item) {
    if (this.item.equals(item)) { // 1st node only will succeed here!
        this.item = this.next.item;
        this.next = this.next.next;
        return true;
    } else if (this.next == null) {
        return false;
    } else if (this.next.item.equals(item)) {
        this.next = this.next.next;
        return true;
    } else {
        return this.next.remove(item);
    }
}
```

(Question 3 continued on next page)

(e) [5 marks] Would you be able to write a version of the remove method in the LinkedNode class that can remove the first item of a list with only one item in it? If yes, then outline the code and explain why it would work, if not, then explain and justify why not.

Not really possible as we cannot set the pointer that points at the current (first) LinkedNode instance to null. HOWEVER, there is an alternative where one can effectively use "tombstones" like in linear probing of hash tables to get this to work, which is also an acceptable correct answer.

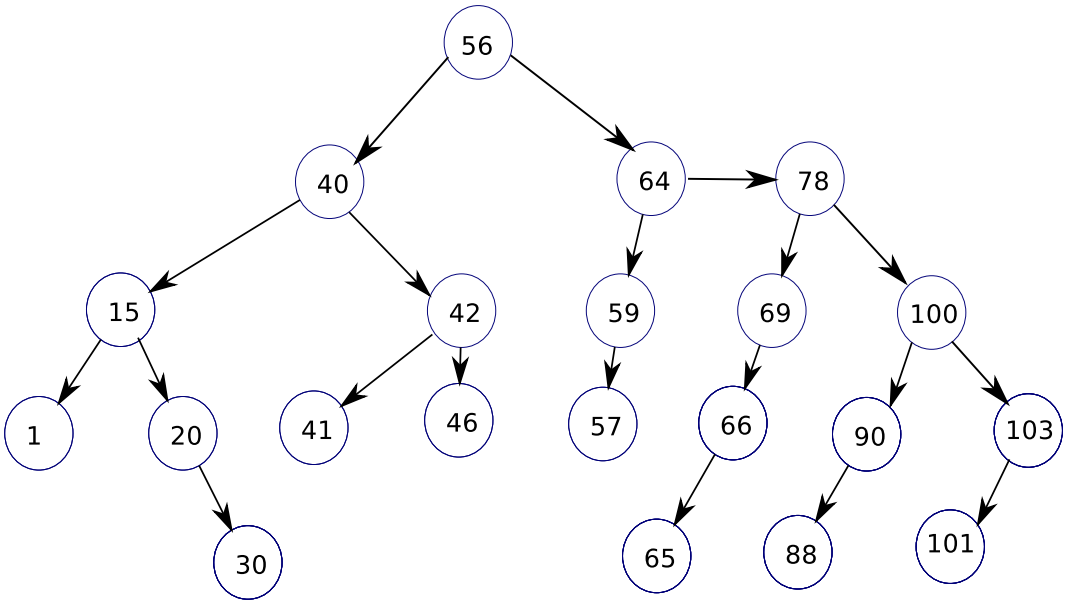
(f) [5 marks] Explain how you would use the LinkedNode class to implement a *queue*. Would you need to adjust the behaviour of the LinkedNode class to make it work for a general queue? Justify your answer.

Ideally show a code example showing how the queue class will have two pointers for front and back of the list (both LinkedNode instances) to make sure that working with queue is always $O(1)$.

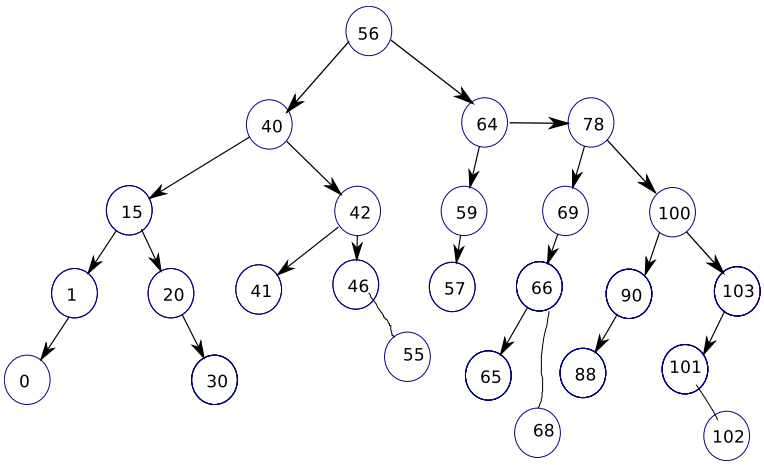
The LinkedNode as is does not allow duplicates while queue does - this will need to be adjusted so that duplicates are allowed.

Question 4. Trees [35 marks]

(a) [10 marks] Consider the following Binary Search Tree that is used to implement a Set. Show what happens as you add the following 5 items (*using the algorithm discussed in lectures*) in the following order: 0, 55, 68, 69, 102.



All but 69 are added in a trivial fashion. 69 is a duplicate and is thus ignored:



(Question 4 continued on next page)

(Question 4 continued)

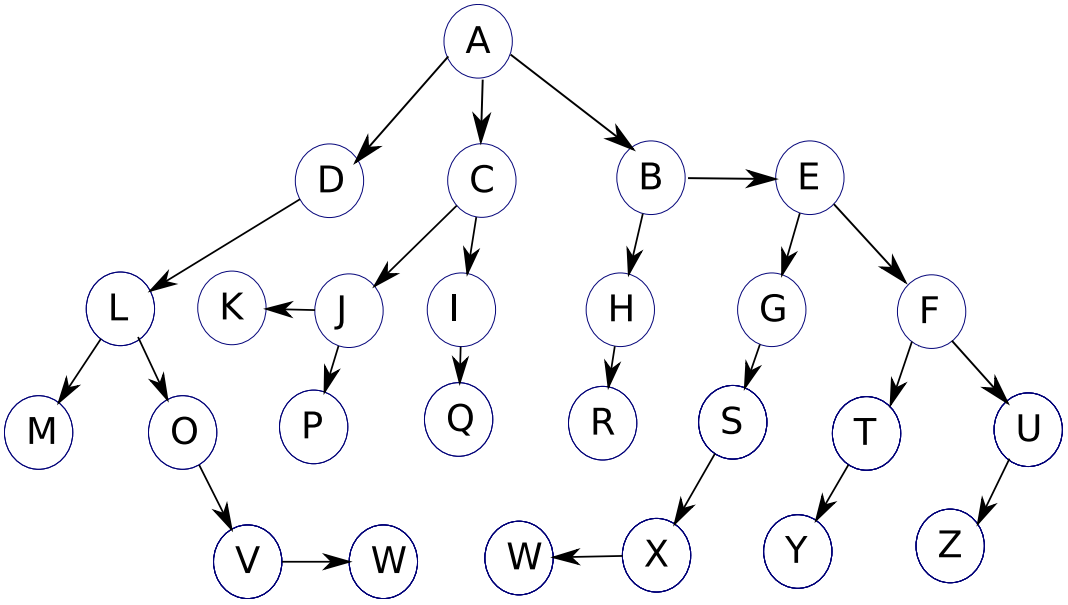
SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

(Question 4 continued on next page)

(Question 4 continued)

Consider the following general tree:



(b) [5 marks] List the nodes in the *breadth first* order of traversal (always processing children of a node left to right).

A D C B L J I H E M O K P Q R G F V S T U W X Y Z W

(c) [5 marks] List the nodes in the *pre-order depth first* order of traversal (always processing children of a node left to right).

A D L M O V W C J K P I Q B H R E G S X W F T Y U Z

(Question 4 continued on next page)

(Question 4 continued)

You are given a class GeneralTreeNode that implements a node of a general tree:

```
import java. util .*;

public class GeneralTreeNode<E> {
    public E item;
    public Set<GeneralTreeNode<E>> children = new HashSet<GeneralTreeNode<E>>();
```

(d) [5 marks] Write a recursive method on a GeneralTreeNode that returns the maximum *depth* of the tree rooted at the current node. A tree of only one node has depth 1.

```
    public int depth() {

        int maxD = 0;
        for (GeneralTreeNode<E> n : children) {
            int d = n.depth();
            if (d > maxD) maxD = d;
        }
        return maxD + 1;

    }
```

(Question 4 continued on next page)

(Question 4 continued)

(e) [10 marks] Write a recursive method levelWidth that returns the width of a given level inside a tree. Assume the root is at level 1.

```
public int levelWidth( int level ) {

    if (level == 1) {
        return 1;
    } else if (level == 2) {
        return children.size();
    } else {
        int ans = 0;
        for (GeneralTreeNode<E> ch : children) {
            ans += ch.levelWidth(level - 1);
        }
        return ans;
    }

}
```

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 5. Heaps [20 marks]

(a) [5 marks] A heap is a *complete partially ordered binary tree stored in an array*. Why is it important for this tree to be complete? Justify your answer.

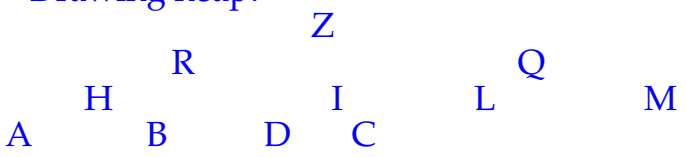
To be able to store it in an array with no gaps and for the heap algorithm to work without losing track of the values stored in the leaves because of the gaps on the way there from the root element.

(Question 5 continued on next page)

(b) [4 marks] Draw the following heap as a tree.

Z	R	Q	H	I	L	M	A	B	D	C					

Drawing heap:



(c) [3 marks] Given the following heap (assume that the letters coming later in the alphabet have to be *higher* in the partially ordered tree), perform the following operation.

Z	R	Q	H	I	L	M	A	B	D	C					

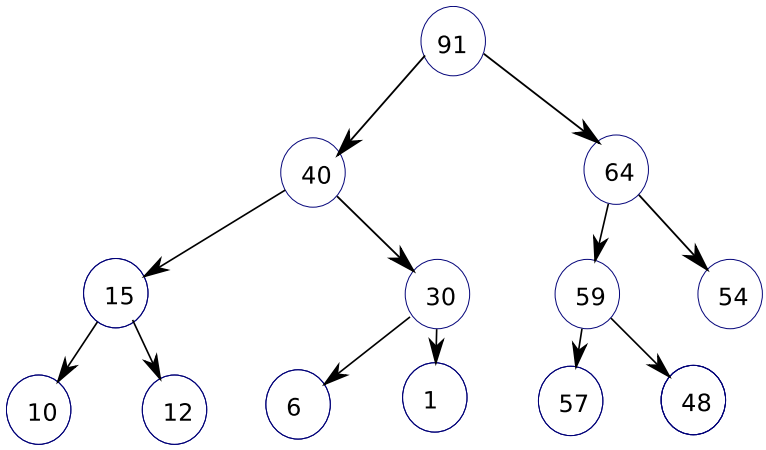
Add P:

Z R Q H I P M A B D C L

(Question 5 continued on next page)

(Question 5 continued)

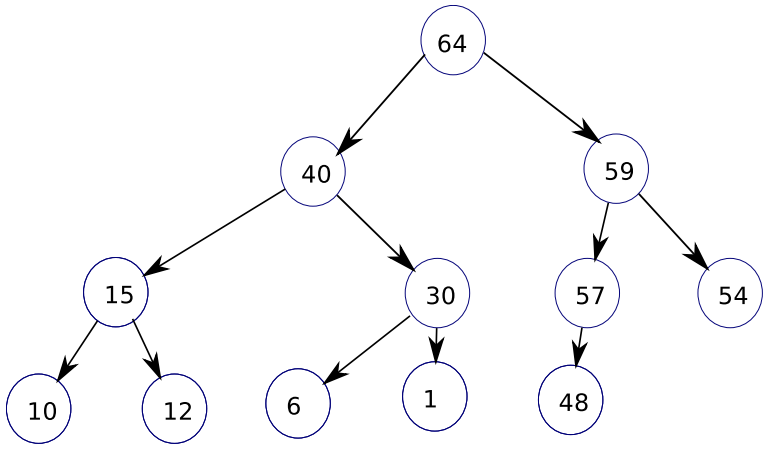
Consider the following heap drawn as a tree:



(d) [3 marks] Show how the heap drawn as a tree above would be represented in the array.

91 40 64 15 30 59 54 10 12 6 1 57 48

(e) [5 marks] Draw the resulting tree after 91 is removed:



SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

Question 6. Hashing [30 marks]

Consider the following class:

```
public class Computer {
    char[] tag = new char[6];
    String manufacturer;
    int HDDSize;
    int RAMSize;
    boolean has3DgraphicsCard;
    String modelNumber;
}
```

(a) [4 marks] Given an example of a *bad* hash function for this class and justify why it is not going to be useful for hashing.

For example adding HDDSize and RAMSize - it does not take all the fields into account and these are least likely to be unique, as well as the resulting number likely to be similar to most computers.

(b) [6 marks] Given an example of a *good* hash function for this class (that does not use a memory address!) and justify why it is going to be useful for hashing.

For example, multiplying each element in tag array by a different prime, then multiplying each character in modelNumber by a large prime and adding them together and using the result. Ideally can use the other fields, but these two are most unique.

(Question 6 continued on next page)

(Question 6 continued)

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

(Question 6 continued on next page)

(Question 6 continued)

(c) [10 marks] Implement the add method for the HashSet class below. Assume that you will never make this HashSet full and use *linear probing* for collision resolution. Feel free to utilise the default hashCode function as defined in java.lang.Object. *You should not assume that any helper functions such as* contains *are present.*

```
public class HashSet<E> {
    public final int INIT_CAPACITY = 7;
    E[] data = (E[]) new Object[INIT_CAPACITY];

    public boolean add(E element) {









                }
    }
}
```

(Question 6 continued on next page)

(Question 6 continued)

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

```
public class HashSet<E> {
    public final int INIT_CAPACITY = 7;
    E[] data = (E[]) new Object[INIT_CAPACITY];

    // Returns false if the item is already present.
    public boolean add(E element) {
        int idx = element.hashCode() % data.length;
        if (data[idx] == null) {
            data[idx] = element;
            return true;
        } else {
            if (data[idx].equals(element))
                return false;

            int p = idx + 1;
            while (p < data.length) {
                if (data[p] == null) {
                    data[p] = element;
                    return true;
                }
                if (data[p].equals(element))
                    return false;
                p++;
            }
            p = 0;
            while (p < idx) {
                if (data[p] == null) {
                    data[p] = element;
                    return true;
                }
                if (data[p].equals(element))
                    return false;
                p++;
            }
            return false; // FULL
        }
    }
}
```

(Question 6 continued on next page)

(Question 6 continued)

(d) [10 marks] Implement an Iterator for your HashSet above.

```
public class HashSet<E> implements Iterable<E> {
    public final int INIT_CAPACITY = 7;
    E[] data = (E[]) new Object[INIT_CAPACITY];

    public java.util.Iterator<E> iterator() { return new HashSetIterator(); }

    class HashSetIterator implements java.util.Iterator<E> {




        public HashSetIterator() {




        }

        public boolean hasNext() {




        }

        public E next() {




        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```

(Question 6 continued on next page)

(Question 6 continued)

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

```
public class HashSet<E> implements Iterable<E> {
    public final int INIT_CAPACITY = 7;
    E[] data = (E[]) new Object[INIT_CAPACITY];

    public java.util.Iterator<E> iterator() { return new HashSetIterator(); }

    class HashSetIterator implements java.util.Iterator<E> {
        private int next;
        public HashSetIterator() {
            while (next < data.length && data[next] == null)
                next++;
        }
        public boolean hasNext() {
            return next < data.length;
        }
        public E next() {
            E ans = data[next];
            next++;
            while (next < data.length && data[next] == null)
                next++;
            return ans;
        }
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }
}
```

Question 7. Final Hard Question [10 marks]

(a) [10 marks] Implement the following split method for your LinkedNode class from Question 3 that operates *recursively* and is initially called on the first node of the linked list.

- For a single node list, the value returned is null.
- For a list with 2 or more nodes that has an *even* number of nodes, the value returned is the head of the second half of the list and the original list is shortened to only contain the first half.
- For a list with 2 or more nodes that has an *odd* number of nodes, the first half list should be one node longer than the second half list.

For example, if you call this split method on list: {1,2,3,4,5} then {4,5} will be returned and the original list will become {1,2,3}. If you call it on {1,2,3,4} then {3,4} will be returned and the original list will become {1,2}.

Note that Java's / operator returns the truncated integer value as in: $5/2 = 2$ and $4/2 = 2$ and $7/2 = 3$.

```
public LinkedNode<E> split() {









}
```

(Question 7 continued on next page)

(Question 7 continued)

SPARE PAGE FOR EXTRA ANSWERS

Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

```
public LinkedNode<E> iterativeSplit() {
    if (this.next == null) {
        return null;
    }
    LinkedNode<E> n = this;
    int i = this.size() / 2;
    while (i-- > 1)
        n = n.next;
    LinkedNode<E> ans = n.next;
    n.next = null;
    return ans;
}

public LinkedNode<E>recursiveSplit() { // TO DO, see assignment solution
}
```

****

Appendix (may be removed)

Brief (and simplified) specifications of some relevant interfaces and classes.

```
public class UI extends java.lang.Object
    public static void clearGraphics();
    public static void setColor(java.awt.Color col);
    public static void drawOval(double x, double y, double width, double height);
    public static void fillOval (double x, double y, double width, double height);

public class Scanner
    public boolean hasNext(); // there is more to read
    public String next(); // return the next token (word)
    public String nextLine(); // return the next line
    public int nextInt(); // return the next integer

public class Random
    public int nextInt(int n); // return a random integer between 0 and n−1
    public double nextDouble(); // return a random double between 0.0 and 1.0

public interface Iterator<E>
    public boolean hasNext();
    public E next();
    public void remove();

public interface Iterable<E>                // Can use in the "for each" loop
    public Iterator<E> iterator();

public interface Comparable<E>              // Can compare this to another E
    public int compareTo(E o);

public interface Comparator<E>              // Can use this to compare two E's
    public int compare(E o1, E o2);
```

```
public interface Collection<E>
    public boolean isEmpty();
    public int size ();
    public boolean add(E element);
    public Iterator <E> iterator();

public interface List<E> extends Collection<E>
    // Implementations: ArrayList
    public E get( int index);
    public void set( int index, E element);
    public void add(E element);
    public void add(int index, E element);
    public void remove(int index);
    public void remove(Object element);

public interface Set extends Collection<E>
    // Implementations: ArraySet, SortedArraySet, HashSet
    public boolean contains(Object element);
    public boolean add(E element);
    public boolean remove(Object element);

public interface Queue<E> extends Collection<E>
    // Implementations: ArrayQueue, LinkedList
    public E peek ();                    // returns null if queue is empty
    public E poll  ();                   // returns null if queue is empty
    public boolean offer  (E element);

public class Stack<E> implements Collection<E>
    public E peek ();                    // returns null if stack is empty
    public E pop ();                     // returns null if stack is empty
    public E push (E element);

public interface Map<K, V>
    // Implementations: HashMap, TreeMap, ArrayMap
    public V get(K key);                 // returns null if no such key
    public void put(K key, V value);
    public void remove(K key);
    public Set<Map.Entry<K, V>> entrySet();
```