Student ID: ....

EXAMINATIONS – 2015

TRIMESTER 2

COMP103
INTRODUCTION TO DATA STRUCTURES AND ALGORITHMS

Time Allowed: TWO HOURS

CLOSED BOOK

Permitted materials: Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in this examination.

Non-electronic foreign language to English dictionaries are permitted.

Other materials are not allowed.

Instructions: Attempt ALL Questions.

Answer in the appropriate boxes if possible — if you write your answer elsewhere, make it clear where your answer can be found.

The exam will be marked out of 120 marks.

Documentation on some relevant Java classes and interfaces can be found at the end of the paper. You may tear that page off if it helps.

There are spare pages for your working and your answers in this exam, but you may ask for additional paper if you need it.

Questions	**Marks**
1. General Questions	[10]
2. Implementing a Collection	[10]
3. Complexity	[10]
4. Recursion, Sorting	[10]
5. Linked Lists and Trees	[30]
6. Binary Search Trees	[20]
7. Priority Queues and Heaps	[20]
8. Hashing	[10]

Question 1. General Questions [10 marks]

(a) [4 marks] By drawing a circle around the correct answer, indicate whether the following statements are TRUE or FALSE:

TRUE / FALSE: List is an abstract class that implements Collection.
False

TRUE / FALSE: Only elements whose type implements Comparable can be sorted.
False

TRUE / FALSE: Selection Sort is a stable sort.
False

TRUE / FALSE: The implements relationship is only allowed between interfaces.
False !

(b) [2 marks] Joe wants to create a set of music files where each element has type MusicFile. Is the below line of code adequate? Justify your answer, making as many observations as you can.

```
Set myMusicFiles = new Set<MusicFile>();
```

(c) [2 marks] Briefly explain why it makes sense for the Java Collections Library to offer multiple implementation variants of "List".

Best choice depends on application context. All variants have trade-offs that need to be evaluated for a particular usage.

(d) [2 marks] Briefly explain how programs can be written such that changing the choice of collection implementation, e.g., TreeSet → HashSet, causes as few changes as possible.

Use interface "Set" for variable declarations. This means only places where collections are created need to be changed.

Question 2. Implementing a Collection

[10 marks]

Complete the methods add and contains for the class ArrayBag. The "Bag" collection type (also known as a "MultiSet") is like a *Set* but may contain any element more than once. Assume that methods ensureCapacity() and int findIndexOf(E item) have already been written and can be used by you. The findIndexOf method returns "-1" if item is not in the bag.

For full marks, give an implementation that is memory efficient in the presence of lots of duplicates. *Hint:* include an array of counts - we have left extra space for this.

You must not use collections from the Java Collections Library in your implementation.

```
public class ArrayBag <E> {
    private static int INITIALCAPACITY = 10;
    private E[] data;
    private int size = 0;




    public ArrayBag() {
        data = (E[]) new Object[INITIALCAPACITY];


    }

    public void add(E item) {
        if (item == null) throw new IllegalArgumentException("null arg for bag");









    }

    public int contains(E item) {
        if (item == null) throw new IllegalArgumentException("null arg for bag");









        return result;
    }
```

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. Complexity [10 marks]

(a) [4 marks] For the following functions, give the complexity class they fall into.

$\frac{1}{2}n^3 + 2n^2 + 99 \quad \in O(____________)$

$\frac{n}{10000} + 10000 \; log \; n \quad \in O(____________)$

$\frac{1}{n} + 15 \quad \in O(____________)$

(b) [3 marks] For the following operations, give the complexity class they fall into.

Adding an element to an ArraySet $\in O(____________)$

Method contains in SortedArraySet $\in O(____________)$

Multiplying an m-by-n matrix with a vector of size n. $\in O(____________)$

(c) [3 marks] Why does InsertionSort not use binary search to find the insertion position for the next element? Wouldn't that improve its complexity class?

No, as the item still needs to be inserted, requiring the shifting of O(n) elements.

Student ID: ....

Question 4. Recursion, Sorting [10 marks]

Consider the following array:

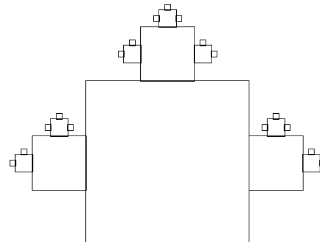
2	5	8	9	7	3	4	6

(a) [2 marks] Which two elements would Selection Sort swap first?

(b) [2 marks] Consider running MergeSort and QuickSort on lots of example problems. Which algorithm would you expect to show the *least variety* in its runtime, and why?

MergeSort: it always does the same number of comparisons. With QuickSort, it depends on what the pivot value turns out to be.

(c) [6 marks] Write a recursive method drawBoxes that uses UI.drawRect(x, y, width, height) to draw the picture below. The initial call to your method will provide 500 and 400 as values for x and y respectively; the main square should have its centre point at (500, 400) and a side length of 300. Make sure that the sides of subsquares are one third the size of their parent square. The recursion should continue so long as the width is greater than 10. Add any additional parameters to your method that you need.



Student ID: ....

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. Linked Lists and Trees [30 marks]

(a) [6 marks] Suppose you are working on a program that uses a *List* of episodes (ie. objects of class Episode), implemented as a list of LinkedNodes. You are to write a method secondToLast for this program. The method is passed the head of the list as an argument, and should return a reference to the second to last Episode the list. If no such item exists, it should return null. Your implementation of this method needs to be iterative.

```
/**Returns the value in the second-to-last node of a list starting at a node */
public Episode secondToLast (LinkedNode<Episode> list){

        if ( list == null)          return null;
        LinkedNode<Episode> rest = list;
        if (rest.next() == null) return null; // rest is the last item.
        while(rest.next.next != null)
                rest = rest.next();
        return rest.get();

}
```

(b) [4 marks] Lectures discussed the use of a Stack for carrying out an iterative, pre-order traversal of a general tree. In the box below, give *pseudocode* for this algorithm for processing the nodes.

```
make a stack,
put root on the stack,
while stack not empty:
    pop stack
    put its children on the stack
    process node

.
```

(c) [3 marks] How would you change the algorithm in (b) so that it results in an iterative **breadth first** traversal of the same tree?

A Queue instead of a Stack leads to breadth-first traversal.

(d) [3 marks] How would you change the algorithm in (b) so that it results in an iterative **in-order** traversal of the same tree?

Use a Stack but (i) initialise by putting the root AND ITS LEFT DESCENDANTS on the stack, and thereafter (ii) after processing a node, if that node has a RIGHT CHILD, put that child AND ALL ITS LEFT DESCENDANTS on the stack. And by the way WHY AM I SHOUTING?

(e) [2 marks] Give two reasons why we might want to provide an *iterative* implementation of a depth-first traversal, as opposed to the more obvious *recursive* implementation.

Allows iterators. Also, having an explicit stack permits *multiple* iterators. Also, an explicit Stack can be more memory efficient than recursive form (activation stack) as does not need to retain local variables.

(f) [3 marks] In a "quad" tree, each node has up to 4 children. If a quad tree is complete, and contains 80 nodes, what is its height? (Note: we say that a tree consisting of just the root has a height of zero).

Height is 3: It has 1 at the root, 4 at level 1, 16 at 2, and 59 at 3. 80 = 1+4+16+59

(g) [8 marks] Now consider the following TreeNode class:

```
public class TreeNode {
    private String word;
    private Set<TreeNode> children = new HashSet<TreeNode>();
     :
     :
}
```

In the box below, complete the find method for the TreeNode class. This should use recursion to search the subtree defined by the argument node for a node whose field word matches the argument text.
The method should return null if no match is found.

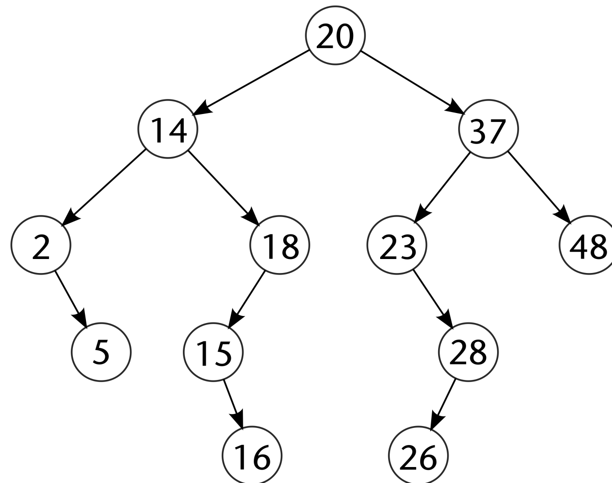
```
private TreeNode find(TreeNode node, String text) {
     if (node.word.equals(text))
          return node;
    for (TreeNode ch: node.children) {
          TreeNode ans = find(ch, text);
          if (ans != null) return ans;
    }
    return null;
}
```

(h) [1 mark] For a full binary tree, roughly what proportion do the leaves make up, out of the total number of nodes?

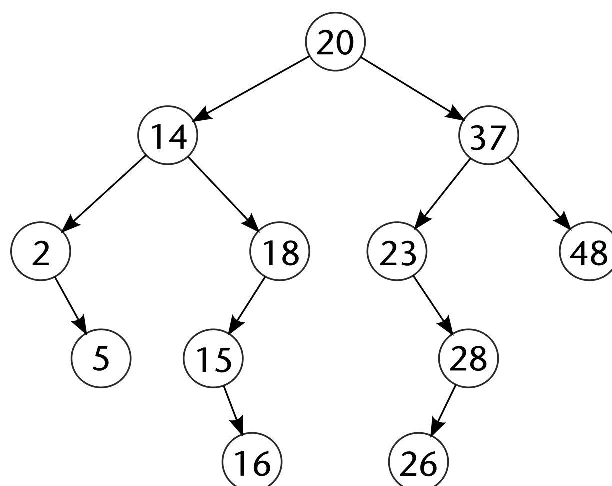
Half of them.

Question 6. Binary search trees [20 marks]

(a) [5 marks] Consider the following Binary Search Tree, that is being used to implement a *Set*. By drawing on the diagram below, show what happens as you add the following 5 items (*using the algorithm described in lectures*) in the following order: 0, 21, 27, 48, 3



(b) [5 marks] Starting from the *same starting* Binary Search Tree (re-drawn below), show what the tree would look like if the values 2, 15, and 37 were removed from the Set.



(c) [8 marks] A BSTSet is a *Set* that uses a tree of BSTNode objects as the data structure.

In the box below, complete the **pseudocode** for the add method in BSTSet.

Assume "item" is a reference to the value that is stored in the BSTNode.

For comparison, here is the pseudocode for the add method of the BSTNode class. In this case it's a recursive method, because it's being called by a BSTNode, but your method will be called by BSTSet instead.

```
public boolean add(E value) {
    if value matches item return false
    if  value < item      // belongs in left subtree
        if there is no left child
            insert as a new left child, and return true
        else call add(value) on the left child, and return the result
    else                  // belongs in right subtree
        if there is no right child
            insert as a new right child, and return true
        else call add(value) on right child, and return the result

}
```

```
public boolean add (E value) {          // Note: this is part of BSTSet, not BSTNode


    if set is empty
        create new root with item value
        increment count
        return true

    else
        ask root node to add a node with item value
        if addition was successful
            increment count
            return true

    return false


}
```

(d) [2 marks] Would adding the following values to a Binary Search Tree, in the given order, result in a *complete* tree?

20, 10, 17, 30, 25, 2

yes. (not a full tree, but a complete one according to the defn given in lectures).

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 7. Priority Queues and Heaps [20 marks]

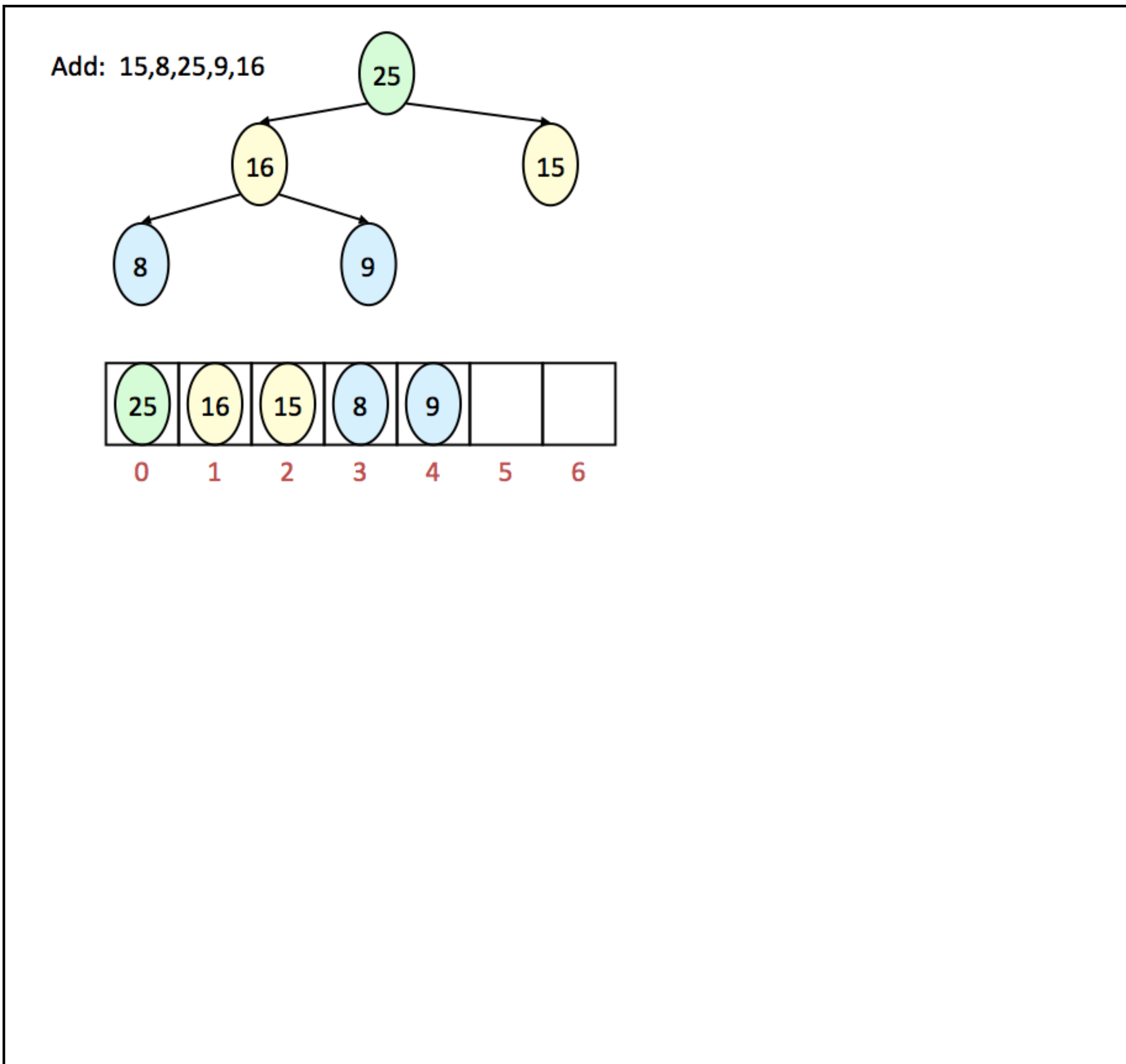
(a) [2 marks] State the property a tree must satisfy in order to be a *partially ordered* tree.

The value in each node must be greater than the values in its children, according to some consistent comparator.

(b) [8 marks] Show the heap that results from adding the following values to a heap that starts off empty: 15, 8, 25, 9, 16 (in that order).

This is a "max" heap, with the largest value at the root (not the smallest).

Draw both the partially-ordered tree representation, and the array representation of the heap.



(c) [2 marks] What is the complexity of turning an unsorted array into heap order?

O(n)

(d) [2 marks] Suppose you built an array implementation of a *Heap* that was based on a **ternary** tree, that is, a tree which has up to 3 children per node, instead of the usual binary tree. What would be the indices of the children of the node at index i in the array?

3i+1, 3i+2, 3i+3

(e) [6 marks] Two efficient implementations of a priority queue are:

- an array of Queues, indexed by the priority, one queue for each priority value
- a HeapQueue, using a partially ordered tree in an array

Give one *advantage* and one *disadvantage* of using the first implementation, compared to the second.

Advantage of using an array of Queues:

Very fast O(1) dequeue and enqueue, cf: O(log n) for HeapQueue

Disadvantage:

Only applicable if we have finite, discrete priorities.

Student ID: ....

Question 8. Hashing

[10 marks]

(a) [3 marks] With probing in Hash Tables, why is it a good idea to rehash to a larger table *well before* the current one is full?

Performance degrades with the number of collisions. A very full table means a very large number of collisions.

(b) [4 marks] Suppose you are working on an application that uses a *Set*. Three possible implementations of *Set* that we have met in this course are ArraySet, BSTSet, and HashSet. Which of these implementations would you choose, and why, in the following cases?

- Your application requires many calls to add(value) and remove(value), but does not have to list out all the members of the current set.

HashSet, which is O(1) for those operations.

- Your application requires many calls to contains, but not many to add. Also, it is often called upon to list out all the members of the current set.

TreeSet, which is O(log n). HashSet would be better were it not for the iterator, which could be very slow.

(c) [3 marks] Why is it important that two items that are ".equal()" have the same hashCode?

If two objects are equal, they should appear only once in a set. But if their hashcodes differ, they will appear twice in the set.

* * * * * * * * * * * * * * *

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.

Appendix (may be removed)

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

interface *Collection<E>*
  public *boolean* isEmpty()
  public *int* size()
  public *boolean* add(*E* item)
  public *boolean* contains(Object item)
  public *boolean* remove(Object element)
  public Iterator<*E*> iterator()

interface *List<E>* **extends** *Collection<E>*
  // Implementations: ArrayList, LinkedList
  public *E* get(*int* index)
  public *E* set(*int* index, *E* element)
  public void add(*int* index, *E* element)
  public *E* remove(*int* index)
  // plus methods inherited from Collection

interface *Set* **extends** *Collection<E>*
  // Implementations: ArraySet, HashSet, TreeSet
  // methods inherited from Collection

interface *Queue<E>* **extends** *Collection<E>*
  // Implementations: ArrayQueue, LinkedList, PriorityQueue
  public *E* peek () *// returns null if queue is empty*
  public *E* poll () *// returns null if queue is empty*
  public *boolean* offer (*E* element) *// returns false if fails to add*
  // plus methods inherited from Collection

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) *// returns element being pushed*
  // plus methods inherited from Collection

interface Map<K, V>
  // Implementations: HashMap, TreeMap, ArrayMap
  public V get(K key) *// returns null if no such key*
  public V put(K key, V value) *// returns old value, or null*
  public V remove(K key) *// returns old value, or null*
  public *boolean* containsKey(K key)
  public *Set*<K> keySet() *// returns a Set of all the keys*

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

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

interface Comparable<E>              // Can compare this to another E
    public int compareTo(E o);       // -ve if this less than o; +ve if greater than o;

interface Comparator<E>              // Can use this to compare two E's
    public int compare(E o1, E o2);  // -ve if o1 less than o2; +ve if greater than o2
```

```
class Collections
    public static void sort( List<E>)
    public static void sort( List<E>, Comparator<E>)
    public static void shuffle ( List<E>, Comparator<E>)

class Arrays
    public static <E> void sort(E[ ] ar, Comparator<E> comp);

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

class String
    public int length()
    public String substring( int beginIndex, int endIndex)
```