



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.

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

TRUE / FALSE: Selection Sort is a stable sort.

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

(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”.

(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.

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 ;
    }
}

```

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 3. Complexity

[10 marks]

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

$$\frac{1}{2}n^3 + 2n^2 + 99 \in O(\underline{\hspace{2cm}})$$

$$\frac{n}{10000} + 10000 \log n \in O(\underline{\hspace{2cm}})$$

$$\frac{1}{n} + 15 \in O(\underline{\hspace{2cm}})$$

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

Adding an element to an `ArraySet` $\in O(\underline{\hspace{2cm}})$

Method `contains` in `SortedArraySet` $\in O(\underline{\hspace{2cm}})$

Multiplying an m-by-n matrix with a vector of size n. $\in O(\underline{\hspace{2cm}})$

(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?

Question 4. Recursion, Sorting

[10 marks]

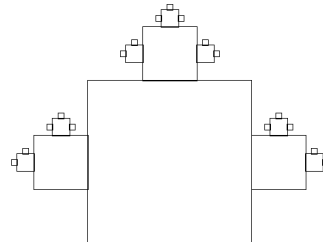
Consider the following array:

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?

(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.

(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?

(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?

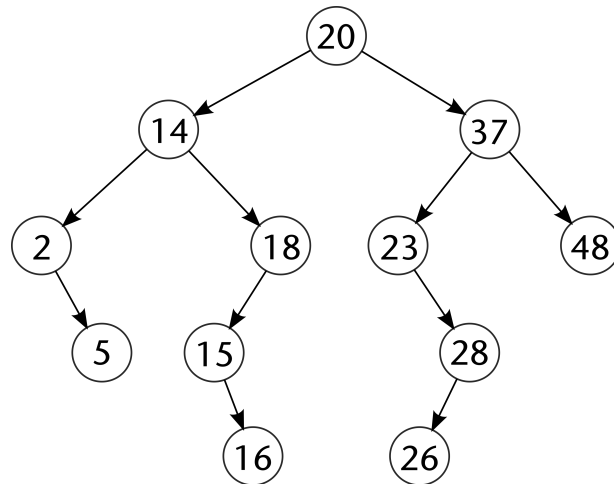
(e) [2 marks] Give two reasons why we might want to provide an *iterative* implementation of a depth-first traversal, as opposed to the more obvious *recursive* implementation.

(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).

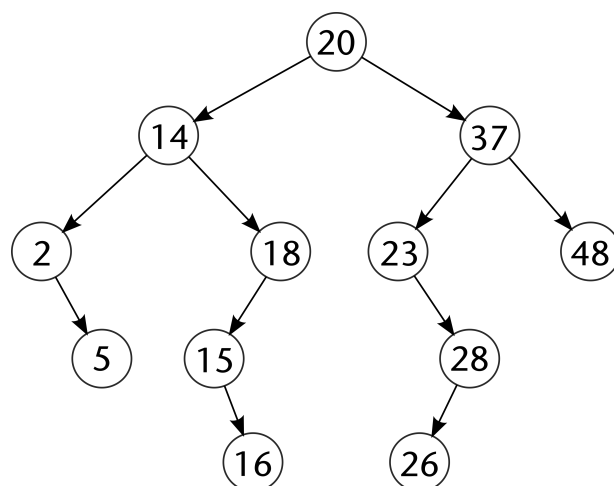
Question 6. Binary search trees

[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
```

```
}
```

(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

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.

(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?

- (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 :

Disadvantage :

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?

(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.

- 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.

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

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.

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)
```
