



EXAMINATIONS — 2011  
**Trimester 1, MID-TERM TEST**

**COMP103**  
**Introduction to**  
**Data Structures and Algorithms**

**Time Allowed:** 45 minutes

- Instructions:**
1. Attempt **all** of the questions.
  2. *Read each question carefully before attempting it.*
  3. This test will be marked out of **45** 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. Documentation on some relevant Java classes, interfaces, and exceptions can be found at the end of the paper.

<b>Questions</b>	<b>Marks</b>
1. Various Questions	[16]
2. Using Collections	[20]
4. Recursion, and Sorting	[9]

**Question 1. Various questions**

[16 marks]

**(a)** [5 marks] Draw lines between things on the left and the collections on the right indicating the *best choice* of Collection for representing that thing.

The students enrolled at Victoria.

QUEUE

A pile of plates.

SET

Names of visitors to a building, in the order in which they visited.

STACK

Customers at a supermarket check-out.

MAP

Friends and their email addresses.

LIST

**(b)** [2 marks] What is the average case “Big- $\mathcal{O}$ ” cost of searching for an item in a Set of  $n$  items implemented using a UNsorted array?

**(c)** [2 marks] What is the average case “Big- $\mathcal{O}$ ” cost of searching for an item in a Set of  $n$  items implemented using a Sorted array?

**(d)** [2 marks] What is the average case “Big- $\mathcal{O}$ ” cost of removing an item from a Set of  $n$  items implemented using a Sorted array?

(e) [3 marks] In the following, `allCreatures` is a List of `Creature` objects. We would like to test each `Creature` object using its `isDead` method (which returns a `Boolean`), and remove those for which this returns the value `true`. What is the problem with the following code for doing this?

```
Iterator iter = allCreatures.iterator ();  
while ( iter .hasNext()) {  
    Creature c = iter .next ();  
    if (c.isDead()) allCreatures.remove(c);  
}
```

(f) [2 marks] In words, describe TWO DIFFERENT ways by which you could to fix this the “`isDead`” creatures do get removed from the `allCreatures` list. You don’t need to provide the code for this question.

Solution 1:

  
  
  
  
  
  
  
  
  
  

Solution 2:

## Question 2. Using several collections

[20 marks]

Suppose you are working on a class `BooksOrganiser`, to be used for organising the records relating to a set of books. The `Book` class is as follows:

```
public class Book {
    private int yearPublished, editionNumber;
    private String author, title ;

    public Book(String author, String title , int yr, int edition) {
        this.yearPublished = yr;
        this.editionNumber = edition;
        this.author = author;
        this.title = title ;
    }

    public Integer getYear() { return yearPublished; }
    public Integer getEdition(){ return editionNumber; }
    public String getAuthor() { return author; }
    public String getTitle () { return title ; }
}
```

(a) [8 marks] Complete the following `Comparator` for the `BooksOrganiser` class. The comparator should compare two `Book` objects on the basis of their year of publication or if these are the same, by edition number. That is: if the books were published in different years, it should return a positive value if the first `Book` was published earlier, and a negative value otherwise. But if two `Books` were published in the *same* year, the comparator should return a positive value if the first has a *higher* edition number.

*Hint:* see the documentation in the appendix.

```
/** Comparator that will order Books based on their year of
    publication . If two books appear in the same year , they should
    be compared on the basis of their edition number. */
private class BookComparator implements Comparator <Book> {

}
}
```

(Question 2 continued on next page)

**(Question 2 continued)****(b)** [12 marks]

A `makeMapOfBooks` method in the `BooksOrganiser` class takes a `Set` of `Book` objects as an argument, and returns a `Map`. The keys of this `Map` should be the set of unique authors in the original `Set`. The value for a given key should be a `List` of all the `Books` by that author. Documentation for `Set`, `Map`, `List`, etc. is given in the appendix.

In addition, each such `List` needs to be *in order*, as determined by `BookComparator`. You may find the `Collections.sort` method useful for this (see appendix).

*Note: you can gain partial marks by achieving the basic task without doing the ordering.*

```
public Map <String, List<Book>> makeMapOfBooks( Set<Book> allBooks ) {
```

```
}
```

Question 3. Recursion and Sorting

[9 marks]

```
public void recFunc(int n) {  
    System.out.print(n + " ");  
    if (n > 14)  
        return;  
    else  
        recFunc(2*n + 1);  
}
```

(a) [4 marks] What will be printed out when recFunc(1) is called?

(b) [2 marks] Name a sorting algorithm with a worst case Big- $\mathcal{O}$  cost of  $\mathcal{O}(n \log n)$  .

(c) [3 marks] In the QuickSort algorithm, it is possible to use a pivot value that is not itself in the List being sorted. If "P" is the pivot value, show the array after the first call to the partition method of QuickSort, if the array starts off as shown below:

H	S	J	M	T	Q	B	T	Z

\*\*\*\*\*

**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.

**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.**

**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*>

```
public Iterator <E> iterator();
```

*// Can use in the "for each" loop*

**public interface** *Comparable* <*E*>

```
public int compareTo(E o);
```

*// Can compare this to another E*

**public interface** *Comparator* <*E*>

```
public int compare(E o1, E o2);
```

*// Can use this to compare two E's*

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

```
public interface Collection<E>
    public boolean isEmpty();
    public int size ();
    public boolean add();
    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: HashSet, SortedSet, TreeSet
    public boolean contains(Object element);
    public boolean add(E element);
    public boolean remove(Object element);
```

```
public interface Queue<E> extends Collection<E>
    // Implementations: 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);
```

```
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, LinkedHashMap
    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();
```