



EXAMINATIONS – 2013
TRIMESTER 1

COMP 103
INTRODUCTION TO
DATA STRUCTURES
AND ALGORITHMS

Time Allowed: THREE HOURS

Instructions: Closed Book.

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 180 marks.

Documentation on some relevant Java classes, interfaces, and exceptions 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.

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

Non-electronic foreign language dictionaries are permitted.

Questions	Marks
1. Collections	[12]
2. Using collections	[28]
3. Implementing a Collection	[17]
4. Binary Search	[12]
5. Recursion and Sorting	[21]
6. Linked Lists	[15]
7. Stacks, Queues	[7]
8. Trees, Tree Traversals	[28]
9. Partially Ordered Trees, and Heaps	[16]
10. Hashing	[17]
11. Various	[7]

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 1. Collections

[12 marks]

(a) [2 marks] A List differs from a Set in two main ways. What are these differences?

(b) [2 marks] In Java, what happens to an object when it is no longer referenced by any other objects?

(c) [2 marks] How many levels are there in a complete binary tree having 32 nodes?

(d) [2 marks] TreeSort is based on what search algorithm?

(e) [2 marks] Why does it not make sense to do an in-order traversal of a Ternary Tree?

(f) [2 marks] If you want to sort a collection on a device with a small amount of memory, which of the fast sorting algorithms we studied (MergeSort, QuickSort, TreeSort and HeapSort) would you *not* employ, and why not?

Question 2. Using collections

[28 marks]

This question is about a program called SongsOrganiser, which manages information about a playlist of Songs. The following code for a Song class just stores information:

```
public class Song {
    private int year;
    private String artist, title;

    // constructor
    public Song(String artist, String title, int yr) {
        this.year = yr;
        this.artist = artist;
        this.title = title;
    }
    // some 'get' methods to provide information.
    public String getArtist() { return artist; }
    public String getTitle() { return title; }
    public Integer getYear() { return year; }
}
```

A real SongsOrganiser program would read in data from a large file. The demo version shown here creates Song objects explicitly, and stores them in a List of Songs:

```
public class SongsOrganiser {
    // constructor
    public SongsOrganiser() {
        List<Song> playlist = new ArrayList<Song> ();

        playlist.add(new Song("Phoenix Foundation", "Hitchcock",2005));
        playlist.add(new Song("Phoenix Foundation", "Hitchcock",2005));
        playlist.add(new Song("Phoenix Foundation", "Buffalo",2010));
        playlist.add(new Song("Dresden Dolls", "Shores of California",2007));
        playlist.add(new Song("Tiny Ruins", "Priest with balloons",2011));
        playlist.add(new Song("SJD", "Lena",2012));
        playlist.add(new Song("Azelia Banks", "212",2011));
        playlist.add(new Song("SJD", "Beautiful haze",2007));
        playlist.add(new Song("SJD", "Beautiful haze",2007));
        playlist.add(new Song("Phoenix Foundation", "Buffalo",2010));

        // re-order the list
        reorderPlaylist(playlist);

        // remove duplicates
        playlist = removeDuplicates(playlist);
    }
    :
}
```

(Question 2 continued)

The method `reorderPlaylist()` in `SongsOrganiser` takes the list and reorders it by using a comparator. The first question below asks you to write the comparator, the second asks for `reorderPlaylist()`, and the third asks for `removeDuplicates()`.

(a) [10 marks] In the box below complete the code for `SongComparator`. This should compare `Songs` based on their year, artist, and title, *in that order*. That is, if two `Songs` differ in their year it should order them on that basis, but if the years are the same, it should compare them by their artist field using the usual alphabetical ordering. And if they still agree, it should compare the titles.

Note: in Java, strings are `Comparable`: a `String` can call the `compareTo(String another)` method. This compares two strings lexicographically (ie. the usual alphabetical ordering).

```

/** Comparator that will order Songs based on their year and title. */
private class SongComparator implements Comparator <Song> {

}

```

(b) [6 marks] In the box below, complete the `reorderPlaylist` method, which should first create a comparator and then use that to sort the items in the list.

Note: you can do this question even if you did not complete the previous one, as it simply uses the `Comparator`.

Hint: use the `Collections` class given in the Appendix.

```

.

```

(Question 2 continued)

(c) [12 marks] The playlist in the preceding code may have duplications in it, which we would like to remove while preserving the order.

Write a method `removeDuplicates` for the `SongsOrganiser` class which takes a List of Song objects, and returns a List that is in the *same order*, but has any duplicates removed.

Note: there are several ways to answer this.

HINT: it might help to write out the necessary steps as "pseudocode" first.

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. Implementing a collection

[17 marks]

Suppose you wanted to implement a Stack using an array. The “top” of the Stack is going to correspond to the last of the items in the array (*i.e.* the one with the highest index).

Here is partial code for such an `ArrayStack`. The constructor and the `pop` method are given. This question asks you to code the four other methods given at the end.

```
public class ArrayStack <E> {
    private static int INITIALCAPACITY=3;
    private int count=0;
    private E[] data;

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

    public E pop(){
        if (count == 0) return null;
        E saved = data[count-1];
        data[count-1] = null;
        count--;
        return saved;
    }

    public boolean isEmpty(){ ... }
    public E peek(){ ... }
    public E push(E value){ ... }
    private void ensureCapacity(){ ... }
}
```

(a) [2 marks] In the box below, write code to implement the `isEmpty()` method.

(b) [3 marks] In the box below, write code to implement the `peek()` method.

(Question 3 continued)

(c) [6 marks] In the box below, write code to implement the `push()` method. Your code should include a call to `ensureCapacity()` at some point, in case the array is full.

(d) [6 marks] Complete the `ensureCapacity()` method below, which doubles the size of the data array being used, if the array becomes full.

Question 5. Recursion and Sorting

[21 marks]

(a) [5 marks] Starting with the following array of letters, complete the steps that the InsertionSort algorithm follows to yield an alphabetically sorted array.

(Note: you may not need all the rows that are given).

M	Y	V	E	C	T	O	R
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
0	1	2	3	4	5	6	7

(b) [4 marks] In words, explain why the InsertionSort algorithm has cost $\mathcal{O}(n^2)$ on lists that begin randomly ordered, but has cost $\mathcal{O}(n)$ on lists that begin in an order that is already nearly sorted.

Cost on randomly ordered Lists is $\mathcal{O}(n^2)$, because...

Cost on nearly-ordered Lists becomes $\mathcal{O}(n)$, because...

(Question 5 continued)

(c) [4 marks] Write a method `thirds()` that uses recursion to keep dividing a number by 3 using integer division, as long as the number remains positive. Your method should print out the results as it goes. For example, calling

`thirds(300);`

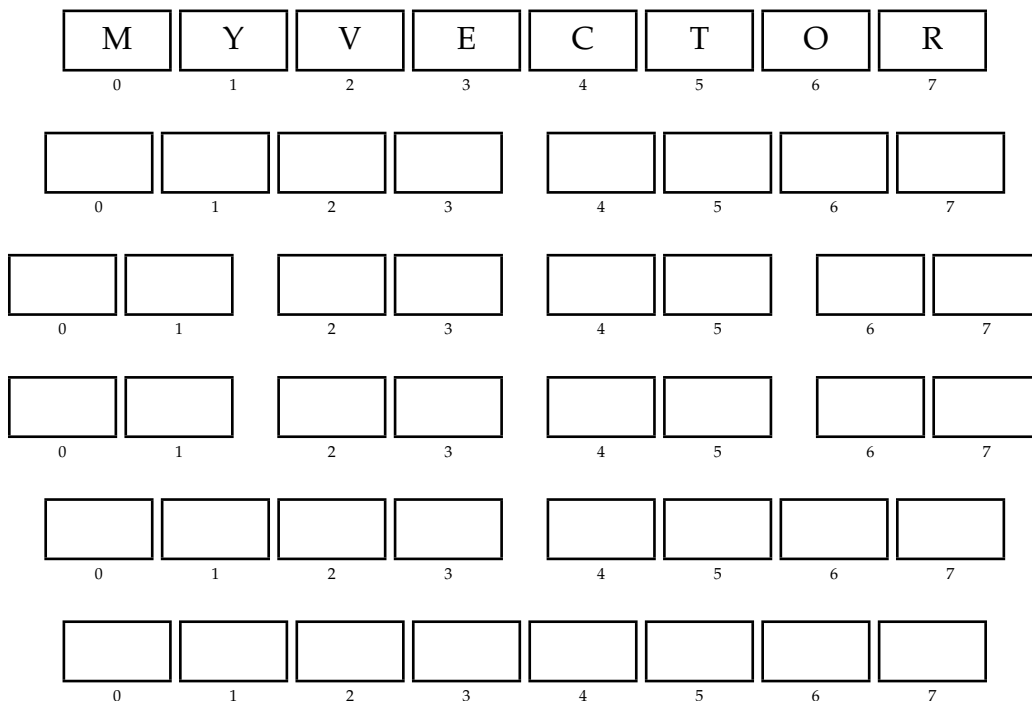
should print this:

300 100 33 11 3 1

```
public void thirds(int n) {  
  
  
  
  
  
  
  
  
  
}
```

(d) [3 marks] What is the “big-*O*” cost of the `thirds(n)` method, in terms of its argument *n*?

(e) [5 marks] Suppose MergeSort is run on the following array of letters. Complete the steps to yield an array that is sorted alphabetically.



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. Linked Lists.

[15 marks]

(a) [15 marks] Suppose you are writing a program to help keep track of a sports team. Each player is represented with an instance of class `Player`, and the list of players on the team is represented with a linked list. Sometimes, one player may be replaced by another player. Write the code that finds a player in the list of players, and replaces that linked list node with a new node that contains the new player.

Your implementation should use the following `LinkedListNode` class:

```
public class LinkedListNode<E>{
    private E value;
    private LinkedListNode<E> next;
    public LinkedListNode(E v, LinkedListNode n){
        value = v;
        next = n;
    }
    public E getValue(){return value;}
    public LinkedListNode getNext(){return next;}
    public void setNext(LinkedListNode n){next = n;}
}
```

You will not need to make use of any methods of the `Player` class except its `equals()` method. In the method header below, the player to replace is passed in the parameter `oldPlayer`, and the new player is in the parameter `newPlayer`.

```
LinkedListNode<Player>
listReplace(LinkedListNode<Player> listhead, Player oldPlayer, Player newPlayer) {

}
}
```

Question 7. Stacks, Queues.

[7 marks]

(a) [5 marks] Consider the following code that uses a stack:

```
Stack<Integer> s = new Stack<Integer>();  
s.push(15);  
s.push(8);  
Integer i1 = s.pop();  
System.out.println(i1);  
s.push(25);  
Integer i2 = s.pop();  
System.out.println(i2);  
Integer i3 = s.pop();  
System.out.println(i3);
```

What will be printed when this code is run?

(b) [2 marks] Which of these (if any) are appropriate uses for a queue:

- (i) Keeping a sorted list of students, and their grades
- (ii) Keeping the next-to-visit nodes for a depth-first traversal of a tree
- (iii) Keeping the next-to-visit nodes for a breadth-first traversal of a tree

Write the number(s) in the answer box below, or write "none".

Question 8. Trees, Tree Traversals

[28 marks]

For this question, we define an empty tree to have depth 0, a tree with only a root to have depth 1, a tree with a root and only one leaf to have depth 2, etc.

(a) [2 marks] What is the depth of a full binary tree that contains three nodes?

(b) [3 marks] Now consider a full binary tree whose depth is *one greater* than in the previous question. How many nodes are in this tree?

(c) [4 marks] How many leaves are there in the bottom layer of a tree with depth 7?

(d) [2 marks] What is the depth of a balanced binary tree with n nodes?

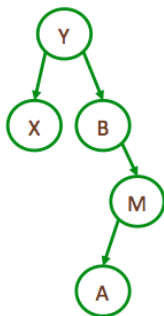
(e) [2 marks] What is the maximum depth of an *unbalanced* binary tree with n nodes?

(f) [1 mark] Consider a full, balanced binary tree with 32 layers (depth = 32). In describing the number of nodes in this tree, which of these would be the most appropriate comparison:

- (i) several thousand
- (ii) $\log_2(32)$
- (iii) the number of squares in half of a chess board
- (iv) the number of people in the world (about 7 billion)

(g) [4 marks] What will the following method print if called on the root of the tree shown below?

```
void treeprint(BinaryTreeNode node) {  
    System.out.println(node.value);  
    if (node.leftChild != null) treeprint(node.leftChild);  
    if (node.rightChild != null) treeprint(node.rightChild);  
}
```



(h) [10 marks] Suppose you have a binary tree that contains integers. Here is a partial description of a node in that tree:

```
public class TreeNode {  
    public int getValue()  
    public void setValue(int value)  
    public TreeNode getLeftChild()  
    public TreeNode getRightChild()  
}
```

Now assume that every leaf has its integer value set, but the values at the other nodes need to be set.

Every node in this tree has either two children, or none. There are no nodes with only one child.

Write a method for the `TreeNode` class that, at each node, computes the difference between the value of its left child and the value of its right child (that is, left - right). It should then set the value of the node to this difference.

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 9. Partially Ordered Trees, and Heaps

[16 marks]

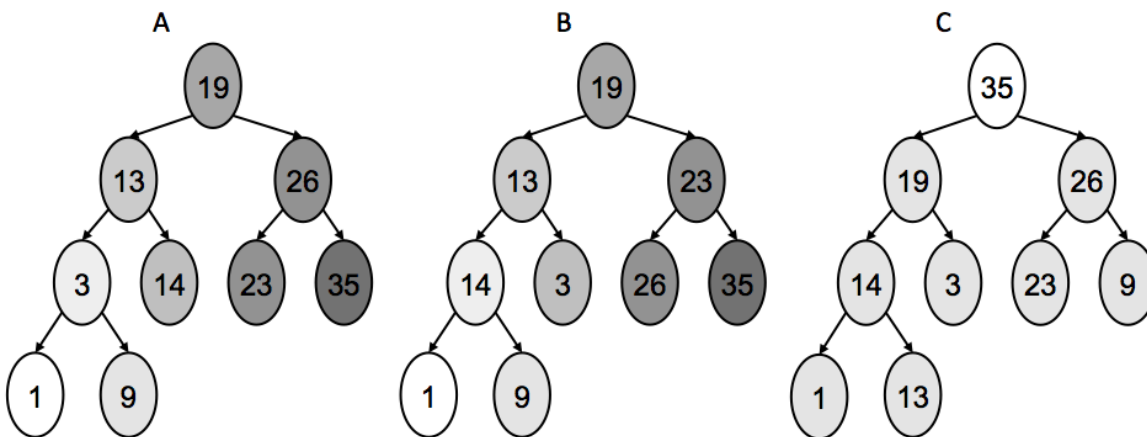
(a) [2 marks] Under best-case assumptions, which is the fastest data structure for implementing contains() (i.e. set membership: "Is a given item in the set?") :

- a binary search tree,
- a heap, or
- a hash table.

Justify your answer.

Justification

(b) [4 marks] Of the trees labeled A,B,C below, one is a binary search tree (BST), one is a partially ordered tree (POT), and one is neither. For each of A,B,C, write what type of tree it is.



A

B

C

(c) [10 marks] Show the resulting heap when the following values are added to an empty heap, in this order: 15, 8, 25, 9, 16 .

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.

draw the unordered tree	and the array:
-------------------------	----------------

Question 10. Hashing

[17 marks]

(a) [4 marks] Briefly describe what “Collision” means in hashing:

(b) [2 marks] Hash tables can be used to implement maps, sets, bags, and other data structures. Consider a hash table of size 100, where each entry in the array is a linked list storing items whose hash code is the index of that array cell (i.e., this hash table implements a set).

After entering 10 such items in the table, what is the complexity (‘big \mathcal{O} ’ cost) of checking whether a given item is already in the table? Assume that the hash function is well suited for the data that is entered in the table.

(c) [2 marks] Now assume that millions of different items have been entered in the table, but the table itself is still of size 100. What is the complexity of looking up whether a given item is in the table? Briefly explain your answer.

Justification:

(d) [2 marks] Now consider using a hash table to implement a bag rather than a set. If you start with an empty bag, and add the same item many times, what is the complexity of looking up whether some item (possibly different) is in the table? Briefly explain your answer.

Justification:

(e) [3 marks] Suppose we want to make a hash table containing the citizens of New Zealand. We are looking for a good hash key. One possibility is to use the town or city where each person lives, mapped to an integer, as the hash key. That is, we make a list of towns in New Zealand, sorted by size in decreasing order, and use the index of the town as the hash key. People living in Auckland (population 1.4 million) would have key 0, Wellington (population 395,000) would have key 1, etc.

Is this a good hash key? Briefly explain your answer.

Justification:

(f) [4 marks] You are given a `Person` class that includes information on the person's date of birth. That information is stored as a set of three integers, one for the year, one for the month, and one for the day of birth. Here is a partial description of the `Person` class:

```
public class Person implements Comparable<Person> {
    public Person(String name, int day, int month, int year)
    public String getName()
    public int getYearOfBirth()
    public int getMonthOfBirth()
    public int getDayOfBirth()
    public int hashCode()
    ...
}
```

The `hashCode` method is defined as follows:

```
int hashCode() {
    return getYearOfBirth();
}
```

Now we'll add the following people to a hash table:

<i>Name</i>	<i>Date of birth</i>
Hamish Skywalker	1/2/95
Jon Smith	28/07/93
Amy Dilbert	5/12/93
Bill Smith	11/11/90
Bertha Duckworth	28/9/72

How many people will collide in the hash table? Can you give their names?

Question 11. Various

[7 marks]

(a) [2 marks] Describe what is meant by a “Stable” sort.

(b) [5 marks] As background for this question, recall that SelectionSort has $\mathcal{O}(n^2)$ complexity. An intuitive explanation of this is that there are n items, and each item requires $\mathcal{O}(n)$ complexity to select from among the remaining unsorted items. Thus, n items each with complexity $\mathcal{O}(n)$ gives $n \times \mathcal{O}(n) = \mathcal{O}(n^2)$.

Consider one of the algorithms you studied that has $\mathcal{O}(n \log_2 n)$ complexity (HeapSort might be an easy example, but you can choose another $\mathcal{O}(n \log_2 n)$ algorithm).

For the algorithm you choose, give a brief explanation (similar to the explanation for SelectionSort given above) of where the n comes from, and where the $\log_2 n$ comes from.

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
```

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

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