



EXAMINATIONS – 2013

TRIMESTER 2

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. General questions	[11]
2. Using collections	[25]
3. Implementing a Collection	[28]
4. Recursion and Sorting	[16]
5. Linked Structures	[22]
6. Trees	[26]
7. Binary Search Trees	[20]
8. Priority Queues and Heaps	[20]
9. Various other questions	[12]

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. General questions

[11 marks]

(a) [3 marks] A Queue differs from a Set in three main ways. What are these differences?

(b) [2 marks] What is a natural data structure to use, in order to reverse the order of items in a collection?

(c) [2 marks] What is wrong with using the length attribute of a String as its hash code?

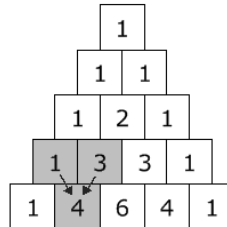
(d) [2 marks] For a SortedArraySet, the complexity of contains() is $\mathcal{O}(\log n)$. Why?

(e) [2 marks] For a SortedArraySet, is the complexity of the remove operation the same as for contains? Explain why or why not.

Question 2. Using collections

[25 marks]

The figure below shows the first few rows of a pattern known as *Pascal's triangle*, which has many interesting mathematical properties. Each row is formed by summing 2 numbers from the row above, except at the edges, where there's a 1.



(a) [10 marks]

Suppose we represent one (horizontal) line of Pascal's triangle in Java using a *List*.

Complete the method `generateNextList` that takes one line (as a *List* of integers) as its argument, and returns the next line (also as a *List* of integers).

For example, if called with the list 1, 3, 3, 1, it should return the list 1, 4, 6, 4, 1.

```
public List <Integer> generateNextList(List<Integer> current) {
```

```
}
```

(b) [10 marks] Complete the method below, that constructs a *Pascal's Triangle* with a certain number of lines. It should take the number of lines as an argument, and should return the *Pascal's Triangle* as a *List of Lists*.

Your method should make use of the `generateNextList()` method described in (a).

```

public List <List<Integer>> generateTriangleNums(int nlines){

}

```

(c) [5 marks] Complete the method below, which prints a *Pascal's triangle*. For example, when called as `displayTriangleNums(6)` it should generate the output shown below. You should make use of the `generateTriangleNums()` method described in (b).

```

1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1

```

```

public void displayTriangleNums(int numlines){

}

```

Question 3. Implementing a Collection

[28 marks]

The space agency *NASA* wants to send a probe into deep space, and the probe's onboard code uses Java arrays. Unfortunately cosmic rays occasionally corrupt the contents of these arrays, changing their values at random. Since the mission has very high cost, *NASA* opts to use an `ArrayList` implementation that is very robust: it will keep **three arrays** for a `List` instead of the usual **one**.

Here is partial code for their `RobustArrayList` implementation of the `List` interface.

The questions will ask you to complete the code for the methods `isEmpty`, `add`, `get` and `ensureOneCapacity`.

```
import java.util .*;

public class RobustArrayList <E> implements List <E> {
    private static int INITIALCAPACITY=2;
    private int count=0;
    private E[] data1, data2, data3;

    public RobustArrayList() {
        data1 = (E[]) new Object[INITIALCAPACITY];
        data2 = (E[]) new Object[INITIALCAPACITY];
        data3 = (E[]) new Object[INITIALCAPACITY];
    }

    :
    :
    :
    // Other public methods (isEmpty, size, set, get, add, remove, etc) would go here
    :
    :
    :

    private void ensureCapacity(){
        data1 = ensureOneCapacity(data1); // see below
        data2 = ensureOneCapacity(data2);
        data3 = ensureOneCapacity(data3);
    }

    // The "ensureOneCapacity" method would go here also.
    :
}
```


(c) [8 marks] Complete the method `add()` for `RobustArrayList`, which adds a new item at the specified position. Note that this is the “add at index” form of the operation, not “add at end”.

```
public void add(int index, E item){
```

```
}
```


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.

Question 4. Recursion and Sorting

[16 marks]

(a) [4 marks] A method `recHalf` is supposed to print larger and larger chunks of a *String* passed as an argument. For example, `recHalf("ABCDEFGH")` should print the following:

A
AB
ABCD
ABCDEFGH

In the box below, give a *recursive* solution for this method.

Hint: Java *String* objects have a method `substring(int beginIndex, int endIndex)` that returns a new string that is a subsequence of the original. They also have a method `length()`.

```
public void recHalf(String str) {
```

```
}
```

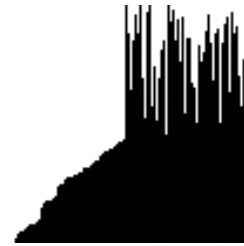
(b) [2 marks] `QuickSort` is a much faster algorithm than `InsertionSort` in most circumstances. However, there is one case in which `InsertionSort` will work faster: what is that case?

(c) [5 marks]

The figure on the right shows SelectionSort part of the way through the sorting operation. The figure shows a snapshot of the whole array being sorted, with the height of the dark bars representing the values of the respective array cells.

Suppose there are 32 items in the whole array.

Indicating your argument by words or a diagram, estimate how many more item-to-item comparisons SelectionSort will need to carry out from this point onwards, for it to finish the sorting operation completely. Give an actual number of steps, not the “big-O” complexity.

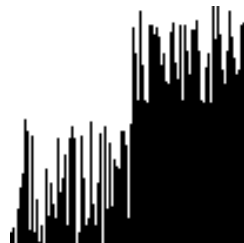


(d) [5 marks]

The figure on the right shows QuickSort part of the way through the sorting operation. Again, suppose there are 32 items in the array being sorted. Indicating your argument by words or a diagram, estimate how many more item-to-item comparisons QuickSort will need to carry out from this point onwards, for it to finish the sorting operation completely.

For simplicity, you may assume that the pivot is almost always about half way through the relevant segment being sorted.

Note: The snapshot shown here was not taken at the same stage as the one for SelectionSort, so the number of steps to go is not necessarily going to be lower than your answer for (c).



Question 5. Linked Structures

[22 marks]

Consider the following IntegerNode class:

```
public class IntegerNode {  
  
    // The integer value stored by the node.  
    private int value;  
  
    // A reference to the next IntegerNode in the linked list .  
    private IntegerNode next;  
  
    ...  
}
```

As usual, an IntegerNode can be regarded as the head of a linked list that is terminated by a null value in the next field of the last node in the list.

Your colleague Sam shows you the code he has drafted for a method to be added to the IntegerNode class that is supposed to add up the numbers in a linked list of IntegerNodes, at odd positions (i.e., 1, 3, 5, ...), with the first node being at position 1.

```
public int skippy() {  
    int result = this.value;  
  
    while (this != null) {  
        result += this.value;  
        this = this.next.next;  
    }  
  
    return result ;  
}
```

(a) [6 marks] There are several problems with his code. Very briefly point out all problems you can identify.

(b) [5 marks] Write a method `public IntegerNode lastButOne()` to be added to `IntegerNode` that returns the reference to the node that precedes the last node in the linked list. You can assume that the list will always contain at least two elements.

```
public IntegerNode lastButOne() {
```

```
}
```

(c) [5 marks] Write a method `int calculate(int x)` to be added to `IntegerNode` that treats the numbers in a linked list of `IntegerNode` as coefficients in a polynomial and returns the value of $n_1 + n_2 * x + n_3 * x^2 + n_4 * x^3 + \dots$, with n_1 being the value for the first node, etc.

```
public int calculate(int x) {
```

```
}
```

(d) [6 marks] You asked your colleague Sam to write a `LinkedList` class that has an `iterator()` method and implements the `Iterable` interface. Sam comes back with a `LinkedList` design that supports a cursor that can be moved forwards and backwards between nodes and argues that this supports an even better form of iteration. Briefly discuss whether Sam has a point. Do you still want Sam to implement your initial design? Provide arguments to justify your answer.

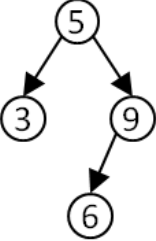
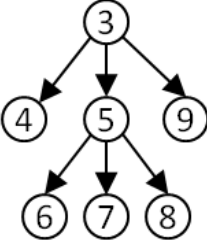
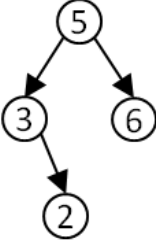

SPARE PAGE FOR EXTRA ANSWERS

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

Question 6. Trees

[26 marks]

(a) [8 marks] For each of the following four structures state which term – binary tree, binary search tree, partially ordered binary tree, general tree, or not a tree – best describes it and briefly justify your answer.

(b) [4 marks] Your colleague Sam has written the following method `contains` for class `GeneralTreeNode` to check whether the subtree formed by the receiver contains the argument node:

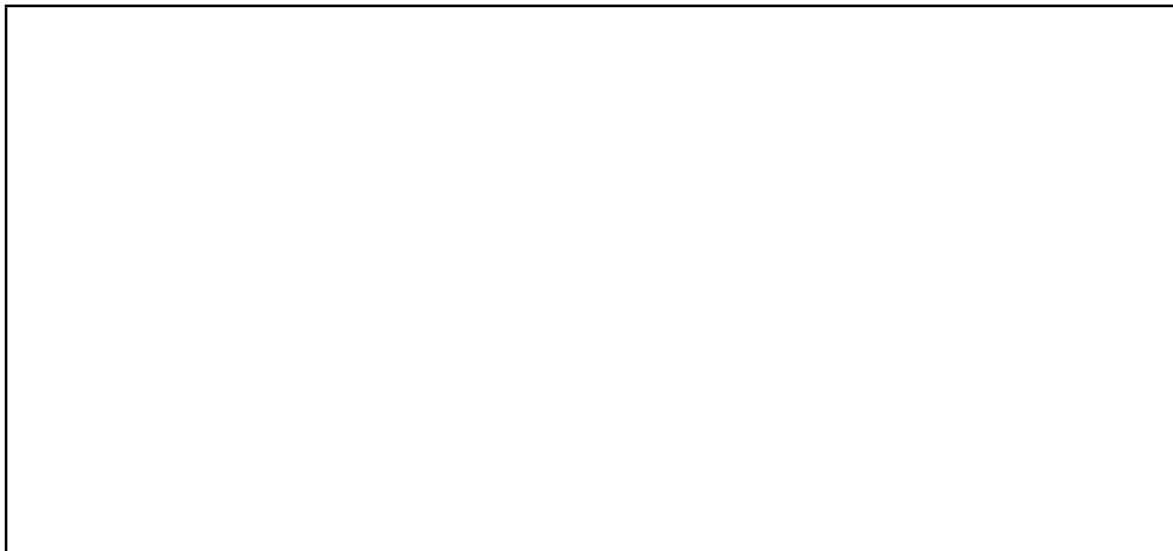
```
public class GeneralTreeNode {  
    private String name;  
  
    private LinkedList<GeneralTreeNode> children = new LinkedList<GeneralTreeNode>();  
  
    ...  
  
    public boolean contains(GeneralTreeNode node) {  
        if (this == node)  
            return true;  
  
        for(GeneralTreeNode child : children) {  
            child.contains(node);  
        }  
    }  
}
```

Briefly point out any bugs you identify and suggest the correction required.

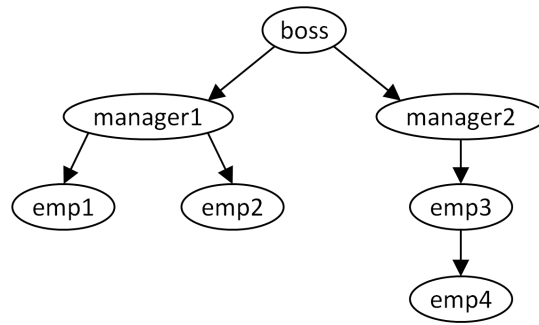
(c) [3 marks] Briefly explain why it is easier to implement a method that removes a node for class `GeneralTreeNode` compared to class `BinarySearchTreeNode`.

(d) [5 marks] Your colleague Jimmy has a terrible coding style. Try to understand what the following method for class `GeneralTreeNode` does, briefly describe its purpose, and state how it normally needs be invoked.

```
public void print(int td, int cd) {  
    if (cd == td) {  
        System.out.println(name);  
        return;  
    }  
  
    for (GeneralTreeNode child : children)  
        child.print(td, cd+1);  
}
```



Consider the following general tree composed of `GeneralTreeNode` nodes:



(e) [6 marks] Give an implementation of method `toString()` for class `GeneralTreeNode` so that it returns the string

boss (manager1 (emp1 emp2) manager2 (emp3 (emp4)))

when invoked on the root node of the above tree.

Class `GeneralTree` node is shown as part of Sam's code just before part (b).

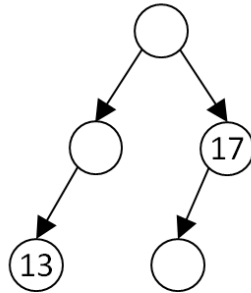
```
public String toString () {
```

```
}
```

Question 7. Binary Search Trees

[20 marks]

Consider the following (*incomplete*) binary search tree that stores integer values:



(a) [1 mark] Assuming the top node is referenced by `root`, what is the value of `root.left().left().getValue()`?

(b) [3 marks] Fill in the missing integer numbers into the blank nodes above.

(c) [2 marks] Write down the sequence of vales produced by a post-order traversal, assuming values have been filled in as necessary.

(d) [2 marks] Briefly describe a method of returning the largest value stored in a binary search tree without ever comparing two values with each other.

(e) [2 marks] Your colleague Ben has implemented a `remove` method for a binary search tree implementation. For some reason he did not follow the COMP 103 notes and hence does not replace a node that has two children with the leftmost child of its right subtree. Instead he replaces it with the rightmost child of its left subtree.

Briefly state your response to Ben's design.

(f) [10 marks] Consider the following implementation for a binary tree containing integers:

```
public class BinaryIntegerTree
{
    private int value;

    private BinaryIntegerTree left ;
    private BinaryIntegerTree right ;
    ...
}
```

In the box below, complete a method “isBinarySearchTree” for this class. The method tests whether a tree satisfies the conditions for being a binary search tree.

You may assume that the range of values contained only spans 1 – 99.

Question 8. Priority Queues and Heaps

[20 marks]

(a) [2 marks] Your colleague Pete has started to write an implementation of a generic PriorityQueue class. He has required the element type to implement interface “Comparable”.

Briefly discuss whether his requirement is reasonable.

(b) [2 marks] Pete now suggests to extend class PriorityQueue with a constructor that accepts a *comparator* object.

Do you think Pete’s suggestion is a good one? Justify your answer.

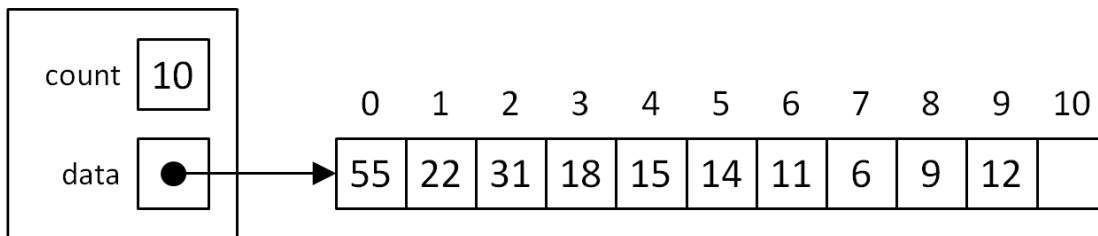
(c) [3 marks] Pete is excited about using a heap for his priority queue implementation. He thinks he can use one of the four standard traversal strategies to traverse the underlying partially ordered tree so that the elements are yielded in the order of descending priority.

Do you think Pete’s plan is going to work? Justify your answer.

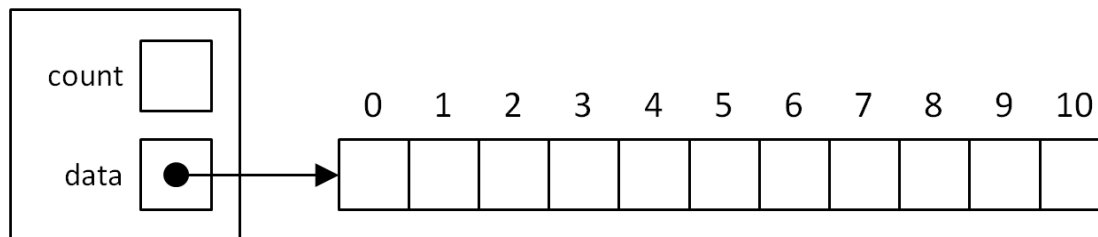
(d) [3 marks] Pete wants to add a constructor to class `PriorityQueue` that takes a collection of elements as an argument and populates the queue with them. Pete thinks he cannot beat $O(n \log n)$ as the asymptotic complexity for this constructor.

Do you think Pete is right? Justify your response.

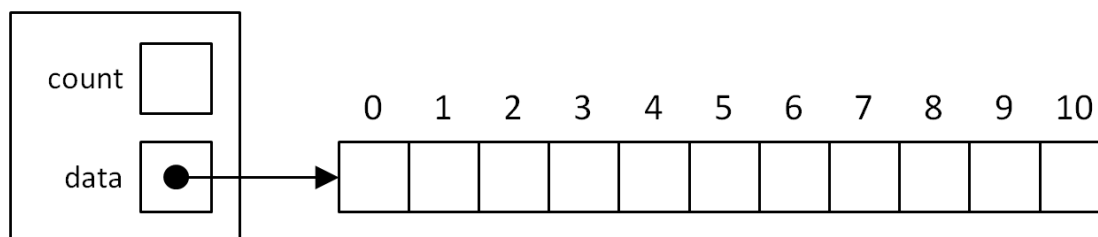
Pete is not sure whether his implementation is correct and asks you for help. He shows you the following state of his priority queue which uses an internal array to store items in a heap. In his implementation elements with larger values have higher priority.



(e) [5 marks] Provide Pete with the values after one `poll()` operation.



(f) [5 marks] Again using Pete's original state, provide him with the values after one `offer(33)` operation.



Question 9. Various other questions

[12 marks]

(a) [2 marks] It is possible to build and use linked lists by using no more than class `LinkedListNode`. Why does it make sense to also provide a class `LinkedList`?

(b) [2 marks] Your colleague Sally suggests replacing your company's current `LinkedList`-based implementation of a `Set` data structure with a `BinarySearchTree`-based implementation.

What speed-up in terms of asymptotic complexity should Sally expect for the operation "remove" in the average case?

(c) [2 marks] What speed-up in terms of asymptotic complexity should Sally expect for the operation "remove" in the worst-case and when would this worst-case occur?

(d) [2 marks] Suggest an even better solution to Sally.

(e) [2 marks] What approach would you use to efficiently print the hundred largest numbers out of an unsorted collection of one billion elements?

(f) [2 marks] How would you determine whether two binary search trees contain the same values?

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)
```
