TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA
UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2016

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 or as instructed in the questions — 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. Stacks and Queues	[10]
2. Complexity	[10]
3. Linked Lists	[10]
4. Binary Search Trees	[15]
5. Heaps	[15]
6. Hashing	[15]
7. General Questions	[20]
8. Iterators	[10]
9. Recursion	[15]

1. Stacks and Queues **(10 marks)**

(a) **(5 marks)** Postfix order (also known as "Reverse Polish Notation") rewrites the expressions to have the operations *following* the arguments. For example, $6+4$ becomes $6\ 4\ +$ and $\sqrt{9}$ becomes $9\ \sqrt{}$. Rewrite the following arithmetic expression in postfix order:

$\sqrt{((5 \times 2)/(4-6+3))}$

(b) **(5 marks)** What does the following code print?

```
Queue<String> q = new ArrayDeque<String>();
q.offer("Z"); q.offer("Y"); q.offer("X");
q.poll(); q.offer("A"); q.offer("B"); q.poll();
UI.println(q);
q.poll(); q.offer("C"); q.offer("D");
UI.println(q);
q.offer("E"); q.offer("F"); q.poll();
UI.println(q);
```

2. Complexity **(10 marks)**

(a) **(3 marks)** For the following functions, give the complexity class they fall into.

$\frac{1}{2}n^4 + 200n^3 + 99 \quad \in O(\underline{\hspace{4cm}})$

$n + n + n \quad \in O(\underline{\hspace{4cm}})$

$\frac{2}{n} + 1 \quad \in O(\underline{\hspace{4cm}})$

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

Adding an element to the front of a `LinkedList` of length *n* $\in O(\underline{\hspace{4cm}})$

Method `contains` in `ArrayList` of length *n* $\in O(\underline{\hspace{4cm}})$

Iterating through all *n* elements in a set of *m* bags $\in O(\underline{\hspace{4cm}})$

(c) **(4 marks)**

List the following sorting algorithms from lowest to highest in terms of their worst case complexity: Bubble Sort, Quick Sort, Merge Sort, Heap Sort, Insertion Sort, Selection Sort. In cases where they have the same complexity — list them in alphabetical order.

3. Linked Lists **(10 marks)**

(a) **(5 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 `middleEpisode` for this program. The method is passed the head of the list as an argument, and should return a reference to the middle `Episode` in the list if the size is odd, and *either* of the middle nodes if the size is even. If no such item exists because the list is empty, it should return null. Your implementation of this method needs to be iterative.

```
/* Returns the value in the middle node of a list
   starting at a given node */
public Episode middleEpisode (LinkedNode<Episode> list){

}
```

(b) **(5 marks)** Give *pseudocode* for the algorithm for processing the nodes in a general tree in an iterative, breadth-first traversal.

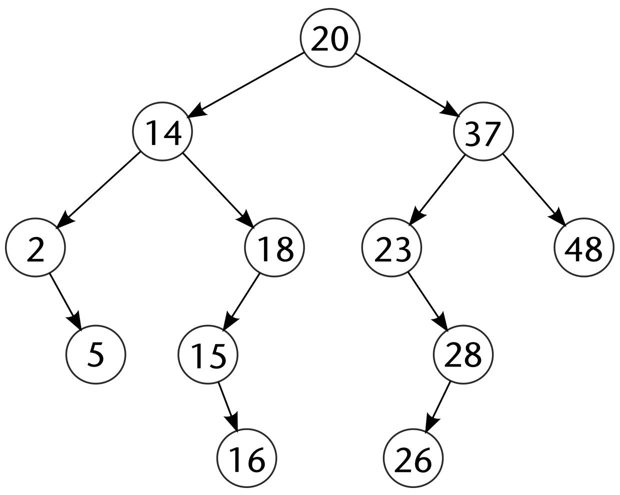
.

SPARE PAGE FOR EXTRA ANSWERS

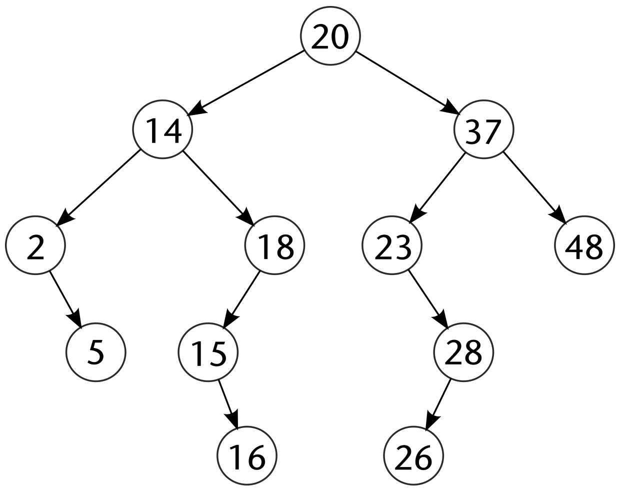
Cross out rough working that you do not want marked.
Specify the question number for work that you do want marked.

4. Binary Search Trees **(15 marks)**

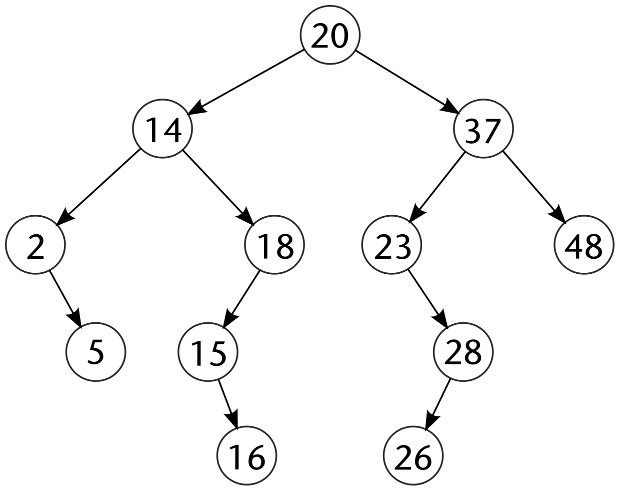
(a) **(5 marks)** Consider the following Binary Search Tree, that is being used to implement a *Set*. By drawing on the diagram below, show what happens as you add the following items (*using the algorithm described in lectures*) in the following order: 50, 36, 21, 17



(b) **(5 marks)** The binary search tree below is the same as the one before. Add nodes of your choice to change it into a *complete* binary search tree which represents a *Set* of items. If it helps to make your answer clearer, redraw the entire tree in the space below.



(c) **(3 marks)** Again using the same binary search tree as before (re-drawn below) that represents a *Set*, show what the tree would look like if the values 2 and 37 were removed from the Set.



(d) **(2 marks)** Now remove the value 20 from the resulting tree above (after deleting 2 and 37) and redraw the final tree below:

5. Heaps **(15 marks)**

(a) **(10 marks)** Show the heap that results from adding the following values to a heap that starts off empty: 15, 0, 25, 26, 27, 1, 2 (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.

draw the partially ordered tree

and the array:

(b) **(5 marks)** Show that turning an unsorted array into a heap is $O(n)$. You can follow the same argument as was presented in the 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.

6. Hashing **(15 marks)**

(a) **(5 marks)** Explain the difference between *open addressing* and *closed addressing* in hashing:

(b) **(10 marks)** Describe two different *collision resolution techniques* for *open addressing*, and state advantages and disadvantages of each:

1st Technique:

2nd Technique:

7. General Questions **(20 marks)**

(a) **(5 marks)** Give an example of a *unit test* and describe what they are used for in software development.

(b) **(5 marks)** Explain what the purpose is of *Comparable* interface in the Java Collections, and give a realistic example of where you might use it in a software project.

(c) **(5 marks)** Consider the Java `Map` implementation. For some maps, the number of items in the *set* of values and the *set* of keys is different. Explain why it is the case and give an example of a map to illustrate your answer.

(d) **(5 marks)** Why is it important to override *both* the `equals` method and the `hashCode` method in a class? What can go wrong if you override one but not the other?

8. Iterators **(10 marks)**

(a) **(10 marks)** In the box below write code for the three missing methods of an iterator for `ArrayList`:

```
private class ArrayListIterator<E> implements Iterator<E> {
  private ArrayList<E> list;
  private int nextIndex;
  private boolean removeable = false;

  private ArrayListIterator(ArrayList<E> list) {







  }

  private boolean hasNext() {







  }

  private E next() {















  }
}
```

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.

9. Recursion **(15 marks)**

Consider the following triple tree data structure (from the test):

```
class TripleTreeNode {
        public TripleTreeNode left, middle, right;
        public String value;
        public TripleTreeNode(String value) {
                this.value = value;
        }
}
```

(a) **(7 marks)** Implement *recursively* a `printAll()` method that will go inside `TripleTreeNode` class that prints all the nodes in the tree by doing a *depth first* traversal. Each node should be indented by the number of spaces that is equal to the depth of the node being printed.

(b) **(8 marks)** **(Hard)** Now implement *recursively* the `drawAll()` method that will also go inside `TripleTreeNode` class that *draws* all the nodes in the tree, *while ensuring that all the nodes of each subtree do not draw over each other*.

Assume that you just need to draw circles (using UI) with a fixed size of 10 pixels diameter. Use `UI.drawOval(x, y, width, height)` where `x,y` is the top left corner and the width and height are both 10. You don't need to print any values in these circles - you just need to ensure that the children will have enough space to draw a subtree without interfering with the other nodes.

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

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.

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: ArrayDeque, LinkedList
  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

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
```

```
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)
```