TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI

VICTORIA

UNIVERSITY OF WELLINGTON

EXAMINATIONS – 2017

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 is provided separately.

There are spare pages for your working and your answers in this exam paper, but you may ask for additional paper if you need it.

Questions	**Marks**
1. General Questions	[12]
2. Linked Lists	[14]
3. Complexity	[14]
4. Binary Search Trees	[15]
5. General Trees	[15]
6. Heaps	[15]
7. Sorting	[20]
8. Hashing	[15]

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.

1. General Questions **(12 marks)**

(a) **(4 marks)** By drawing a circle around the correct answer, indicate whether the following statements are TRUE or FALSE:

TRUE / FALSE: ArrayList is a class that *extends* Collection.

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

TRUE / FALSE: ArraySet *implements* Set.

TRUE / FALSE: ArrayList directly implements List.

(b) **(4 marks)** Ben wants to create a list of photos where each element has type Photo. Identify two issues with Ben's attempt below.

```
ArrayList<Photo> myPhotos = new List<Photo>();
```

(c) **(2 marks)** Describe a difference between an interface and a class.

(d) **(2 marks)** Describe an advantage of using linked structures – e.g., a linked list – compared to using arrays.

2. Linked Lists **(14 marks)**

Your task is to complete the implementation of class PlayList below. A PlayList object has a *cursor* that points to the LinkedNode object that references the currently selected Song object. This cursor can be set to specific positions and moved forward and backward. As you can see by reading the available code, PlayList uses a linked list approach to store the songs available in a play list. Do not use any other collections from the Java collection library to complete the implementation.

```
public class PlayList {                // manages a list of songs
    LinkedNode<Song> head;             // the first song in the play list
    LinkedNode<Song> cursor;           // current point for insertion , removal, ...

    public PlayList() {                // creates an empty play list
        cursor = head = null;          //  cursor = null, if the play list is empty
    }
    ...
}
```

Assume the following methods for LinkedNode to be available:

```
class LinkedNode<E> { ...
    LinkedNode<E> getNext() {return next;}
    void setNext(LinkedNode<E> next) {this.next = next;}
}
```

(a) **(2 marks)** Implement the following moveCursorToStart method for PlayList that sets the cursor to the first song in the list.

```
public void moveCursorToStart() {



}
```

(b) **(3 marks)** Implement the following moveCursorToEnd method for PlayList that sets the cursor to the last song in the list.

```
public void moveCursorToEnd() {



}
```

(c) **(3 marks)** Implement the following moveCursorRight method for PlayList that moves the cursor to the next song in the list.

```
public void moveCursorRight() {

}
```

(d) **(6 marks)** Implement the following moveCursorLeft method for PlayList that moves the cursor to the previous song in the list.

```
public void moveCursorLeft() {

}
```

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.

3. Complexity **(14 marks)**

(a) **(5 marks)** For the following functions, give the complexity class they fall into.

$5n^5 + 4n + 10 \quad \in O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$

$\frac{1000}{n} + 1000\, n \quad \in O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$

$\frac{2}{logn} + 2n - \frac{1}{2}n \quad \in O(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)$

(b) **(2 marks)** What is the average case complexity class of the following code, in terms of number of assignments?

```
for ( int i=0; i<n; i++) {
    a[i] = a[i] * 2;
}
```

O(____________)

(c) **(4 marks)** What is the average case complexity class of the following code, in terms of number of assignments?

```
for ( int i=0; i<n; i++)
    for ( int j=0; j<i; j++)
        a[i][j] = a[i][j] + 1;
```

O(____________)

(d) **(3 marks)** What is the average case complexity class of the following code, in terms of number of assignments?

```
int i=1;

while (i<n) {
    a[i-1] = a[ i ];
    i = i * 2;
}
```

O(____________)

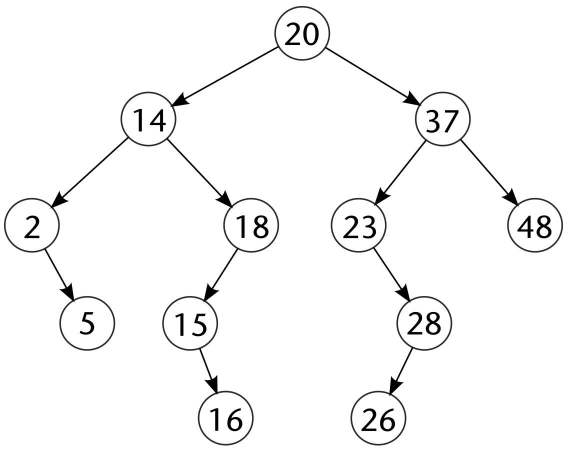
4. Binary Search Trees **(15 marks)**

   (a) **(4 marks)** Suppose you start with an empty Binary Search Tree, and insert the following values in the order shown:

   4 7 5 1 3 8 6 2

   Draw the resulting Binary Search Tree.

(b) **(4 marks)** Show the result of deleting **20** from the following Binary Search Tree.



(c) **(4 marks)** Why is it a bad idea to insert values into a Binary Search Tree in ascending order?

(d) **(3 marks)** Explain briefly how you would find the smallest value in a non-empty Binary Search Tree.

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.

5. General Trees **(15 marks)**

Suppose you need to maintain a collection of information about the members of a large family, starting from a single ancestor. For each family member, you need to store the person's name (assumed to be unique string), a set of attributes and a list of that person's children, thus forming a tree. The attributes are just strings that represent facts about the person, such as "New Zealander", "Farmer" or "Nelson".

(a) **(3 marks)** Write Java declarations for the fields required in the class used to represent nodes in this tree.

```
public class Person {

}
```

(b) **(6 marks)** Complete the following method, in the `Person` class, to print the names of all people in the family, starting with the `Person` the method is called on.

```
public void printNames() {

}
```

(c) **(6 marks)** Complete the following method, in the `Person` class, to count the number of people in the family having a given attribute, starting with the `Person` the method is called on.

```
public int countWithAttribute(String att) {






}
```

6. Heaps **(15 marks)**

(a) A *heap* is defined to be a complete partially ordered tree stored in an array.

i. **(2 marks)** What is the defining property of a *complete* tree?

ii. **(2 marks)** What is the defining property of a *partially ordered* tree?

iii. **(1 mark)** If the root of the heap is stored at index 0, what are the indexes of the left and right children of the node at index k (if they exist)?

iv. **(1 mark)** How do you determine whether a node in a heap is a leaf?

(b) **(6 marks)** Suppose the following values are added, in the order shown, to a heap that starts off empty:

1 3 5 6 4 2 7

Show the partially-ordered tree represented by the heap after each value is added.

Note that you should construct a "max" heap, so the largest value (7) ends at the root.

Draw the array containing the heap after all values have been added:

(c) **(3 marks)**

Given the following array representing a “max” heap:

20	14	15	10	9	13	7	8	6

Draw the partially-ordered tree representation of the heap after removing its largest element.

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.

7. Sorting **(20 marks)**

(a) Selection sort and insertion sort both work by dividing the list (represented as an array) into a sorted part and an unsorted part, and increasing the size of the sorted part by one in each execution of the outer loop.

For each of these algorithms, explain briefly how the algorithm chooses the next item to be added to the sorted part, and how it is added.

i. **(3 marks)** Selection sort

ii. **(3 marks)** Insertion Sort

(b) Merge sort and Quicksort both work by splitting the list to be sorted into two parts, sorting each of them recursively, and then combining the results to produce the required sorted list.

Briefly describe each of these steps, and give the cost of each step in terms of the sizes of the sub-lists being split or combined.

i. **(5 marks)** Merge sort:

Split:

Cost:

Combine:

Cost:

ii. **(5 marks)** Quicksort:

Split:

Cost:

Combine:

Cost:

(c) **(4 marks)** Suppose you have a large set of integers to sort (more than a million), and you know that all of the integers are between one and a thousand.

Describe an efficient way of sorting these numbers, and give its cost.

8. Hashing **(15 marks)**

(a) **(5 marks)** Suppose you have a hash table which stores strings, using *closed hashing*, and that the following words have hash codes as shown below:

Word:	**the**	**quick**	**lazy**	**dog**	**jumps**	**over**	**brown**	**fox**
Hash code:	5	6	8	2	6	7	3	7

Show the table that would result after inserting the above words into the table, in the order shown, starting with an empty table, and using *linear probing* to resolve collisions.

You should show the position of each word in the table by drawing a line from the relevant cell in the table to the word that is stored there. For example, you should start by drawing a line from cell 5 to "the".

0	1	2	3	4	5	6	7	8	9	10	11	12

the **quick** **lazy** **dog** **jumps** **over** **brown** **fox**

(b) **(5 marks)** Explain how *open hashing* avoids the problem of handling collisions associated with closed hashing, as in part **(a)**.

(c) **(5 marks)** Consider a hash function for words that just takes the positions in the alphabet (counting from 0, as shown in the table below) of the first and last letters and adds them together. For example the hash code for "the" is $19 + 4 = 23$.

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12

n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

Explain why this hash function is not very good, and how you would design a better hash function.

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

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.

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.

Documentation

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)
```