TE WHARE WĀNANGA O TE ŪPOKO O TE IKA A MĀUI
VICTORIA
UNIVERSITY OF WELLINGTON

Student ID: .....

EXAMINATIONS — 2012

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 (i.e. allocate 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.

Questions	**Marks**
1. Collections	[6]
2. Implementing a Stack	[19]
3. Sorting	[10]
4. Recursion	[10]

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 [6 marks]

(a) [6 marks] In the boxes below give a short (e.g. one sentence) description for each of the following collections identifying their *key feature(s)*.

Bag

Set

Map

Stack

Queue

List

Question 2. Implementing a Stack [19 marks]

(a) [8 marks] Complete the following class that implements a Stack using an array. Note that pop() should through an EmptyStackException if the stack is empty.

```
public class Stack<E> {
        static int INITIAL_CAPACITY = 2;
        E[] data;


        public Stack() {




        }

        public void push(E item) {




        }

        public E pop() {




        }









}
```

(Question 2 continued on next page)

(Question 2 continued)

(b) [6 marks] Write an iterator for your Stack class above that is defined as a private inner class and goes through the items in the stack *from the top to the bottom* - in other words, the *first* item should be the one that would have come off *first* if a pop method were to be called. For simplicity, assume that you don't need to implement the remove method and that you don't need to worry about anyone modifying the Stack while you are iterating through it.

```
import java. util . Iterator ;
public class Stack<E> implements Iterable<E> {
        // CODE FROM PREVIOUS SUBQUESTION IMPLEMENTING STACK
        public Iterator <E> iterator() {
                return new StackIterator<E>(this);
        }
        private class StackIterator<E> implements Iterator<E> {
                Stack<E> stack;
                int position ;

                public StackIterator(Stack<E> stack) {









                  }

                  public boolean hasNext() {









                  }

                  public E next() {









                  }
                  public void remove() { throw new UnsupportedOperationException(); }
        } }
```

(Question 2 continued on next page)

(Question 2 continued)

(c) [3 marks] What will the following code print when using the Stack you have just implemented?

```
public static void main(String[] a) {
    Stack<String> ss = new Stack<String>();
    ss.push("A"); ss.push("B"); ss.push("C");
    System.out.println(ss.pop() + ss.pop() + ss.pop());
    for (String s : ss)
        System.out.println(s);
     System.out.println("------");
}
```

(d) [2 marks] What is the best order of items when iterating through a Stack? Justify your answer.

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.

Question 3. Sorting [10 marks]

(a) [3 marks] Consider the *insertion sort* as discussed in the lectures. Give an example of an array of 10 elements that would produce the *best* cost behaviour for insertion sort and explain briefly why.

(b) [3 marks] Consider the *insertion sort* as discussed in the lectures. Give an example of an array of 10 elements that would produce the *worst* cost behaviour for insertion sort and explain briefly why.

(c) [2 marks] What does it mean for a sorting algorithm to be *stable*?

(d) [2 marks] Imagine you are given two sorting algorithms: one that has worst cost $O(nlog(n))$ and one that has worst cost $O(n^2)$. Which of these two algorithms is going to sort 1 trillion integers faster? Justify your answer.

Question 4. Recursion [10 marks]

(a) [4 marks] Fix the recursive code below that tries to calculate a factorial. A reminder: $n! = 1 * 2 * ... * (n-1) * n$.

```
public int factorial (int n) {
        if (n > 1) {
                return n * factorial (n);
        } else {
                return 1;
        }
}
```

(b) [4 marks] Write a recursive function that returns the nth Fibonacci number. A reminder that Fibonacci sequence is as follows: $1, 1, 2, 3, 5, 8, \ldots, f_{n-2}, f_{n-1}, f_n, \ldots$ where $f_n = f_{n-1} + f_{n-2}$.

```
public int fib (int n) {




}
```

(c) [2 marks] Explain why your recursive version of fib is particularly inefficient. 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.

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>                // Can use in the "for each" loop
    public Iterator<E> iterator();

public interface Comparable<E>              // Can compare this to another E
    public int compareTo(E o);

public interface Comparator<E>              // Can use this to compare two E's
    public int compare(E o1, E o2);
```

```
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: ArraySet, SortedArraySet, HashSet
    public boolean contains(Object element);
    public boolean add(E element);
    public boolean remove(Object element);

public interface Queue<E> extends Collection<E>
    // Implementations: ArrayQueue, LinkedList
    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, ArrayMap
    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();
```