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

Student ID: .....

EXAMINATIONS — 2011

Trimester 2, MID-TERM TEST

COMP103
Introduction to
Data Structures and Algorithms

SOLUTIONS

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, so allocate approximately 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. Various Questions	[10]
2. Using and Implementing Collections	[25]
3. Recursion, and Sorting	[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.

Student ID: .....

Question 1. Various questions [10 marks]

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

- A Stack is an interface, that implements Collection: TRUE/FALSE — False
- A Priority Queue can add elements anywhere: TRUE/FALSE — True
- A for-each loop can be used on all Iterable types: TRUE/FALSE — True
- Selection Sort is a stable sort: TRUE/FALSE — False

(b) [2 marks] If it takes 10 steps to search through an array of 1024 items using binary search, how many steps will it take to search through an array of 4096 items (*ie.* four times as many)?

12 steps

(c) [4 marks] Write down the "Big-$\mathcal{O}$" costs of the following, for an array with n items:

- Insertion Sort on an almost sorted array:

  Cost: $\mathcal{O}(n)$

- Selection Sort on an almost sorted array:

  Cost: $\mathcal{O}(n^2)$

- Average case of searching for a single item in a SortedArrayBag:

  Cost: $\mathcal{O}(\log n)$

- Worst case of Merge Sort:

  Cost: $\mathcal{O}(n \log n)$

Question 2. Using and Implementing Collections [25 marks]

(a) [5 marks] Below is a declaration of a Map (named toDo) which maps tasks (of type Task) to the name of their owners (of type String):

```
Map <Task, String> toDo = new HashMap <Task, String> ();
```

In the box below, write the code to iterate over this map and print the tasks along with the name of their owners.

[Note: You can assume that the class Task is already implemented and provides a toString() method]

```
for(Map.Entry<Task, String> td : toDo.entrySet()) {
        UI. println ("Task = " + td.getKey() + " is owned by " + td.getValue());
}

OR iterate over keys.
```

.

(b) [6 marks]

The remove method of a SortedArrayBag (similar to Assignment#4) uses the findIndex method to locate the item to be removed - as shown in the 3rd line, below.

Complete the remove method so that it removes the item and leaves the array in a correct state. It should return true if the collection changes, and false if it did not change.

```
public boolean remove (Object item) {

    if (item==null) return false;
    int index = findIndex(item);


    if (index == count || !item.equals(data[index]))
    return false;

    count--;
    for ( int i=index; i<count; i++)
        data[ i ]= data[ i +1];
    data[count]=null;
    return true;




}
```

(c) [2 marks] Explain the difference between a Bag and a Set.

Set does NOT allow duplicates, but Bag does.

(d) [2 marks] Name the collection that is used to implement "undo" operations in many applications. What property of this collection makes this possible?

collection:

property: FILO (first in last out) property of stack

Consider the following CompareFaces class that stores and sorts a List of Faces, and then answer the questions on the following page.

```
public class CompareFaces{
    private List <Face> crowd; //private field

    /** constructor that instantiates a new ArrayList of Face and sorts it */
    public CompareFaces(){
        crowd = new ArrayList <Face>();
        Collections.sort(crowd);
    }

    public static void main(String[] args){
        CompareFaces cf = new CompareFaces();
    }

    :
    :

}
```

(e) [4 marks] For the Collections.sort(crowd) to work (in the constructor of the above CompareFaces class), the Face class (below) must be Comparable. Complete the compareTo method (in the Face class below) which compares Faces based on their size.

```
public class Face implements Comparable<Face>{

    private int wd;
    private int ht;
    private String name;
    private int size;

    // constructor
    public Face(String name, int w, int h, int size){
        this.name = name;
        wd = w;
        ht = h;
        this.size = size;
    }

    public int getSize(){  return size;  }

    public int getPosition(){  return wd;  }
    :
    :
    public int compareTo(Face other){
        // YOUR CODE HERE

        return (this.getSize()-other.getSize());


    }

}//end of Face class
```

(f) [6 marks] Write a LeftToRightComparator comparator that will compare Face objects based on their positions on the screen.

```
private class LeftToRightComparator implements ...                    {

private class LeftToRightComparator implements Comparator<Face>{
        public int compare(Face f1, Face f2){
            return (f1.getPosition()-f2.getPosition());
        }
}




}
```

Question 3. Recursion and Sorting [10 marks]

```
public void seriesRec(int n) {
    System.out.print(n + "  ");
    if (n > 15)
        return;
    else
        seriesRec(3*n -1);
}
```

(a) [4 marks] What will be printed out when seriesRec(1) is called?

1 2 5 14 41

(b) [2 marks] Given the initial array shown (1st row), identify the sorting algorithm which results in the array shown (2nd row) after **3 passes through its outer loop** (*assume the usual A-to-Z sort is required*).

Start:	H	F	L	P	Q	E	Z
After 3 passes:	F	H	L	P	Q	E	Z

Sorting algorithm: Insertion Sort

(c) [2 marks] As with the previous question: identify the sorting algorithm which results in the following array after **3 passes through its outer loop** (*assume the usual A-to-Z sort is required*).

Start:	M	J	B	R	P	A	D
After 3 passes:	A	B	D	R	P	M	J

Sorting algorithm: Selection Sort

(d) [2 marks]

List one advantage of MergeSort over QuickSort and one advantage of QuickSort over MergeSort.

- Advantage of MergeSort: eg. worst case still $\mathcal{O}(n^2)$, is stable,...
- Advantage of QuickSort: eg. is in-place,...

****

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();
```