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

Student ID: .....

EXAMINATIONS — 2005

END-YEAR

COMP 103
Introduction to
Data Structures and Algorithms

Time Allowed: 3 Hours

Instructions: There are a total of 180 marks on this exam.
Attempt **all** questions.
Write your answers in the boxes in this test paper and hand in all sheets.
Calculators may be used.
Non-electronic foreign language translation dictionaries may be used.
There is documentation on relevant Java classes and interfaces at the end of the exam paper.

Questions		**Marks**
1.	Basic Questions	[30]
2.	Programming with Priority Queues	[23]
3.	Linked Lists	[20]
4.	Analysis of Algorithms	[23]
5.	Implementing Collections with Arrays	[9]
6.	Implementing Sets with Binary Search Trees	[24]
7.	Implementing Priority Queues	[14]
8.	Trees	[20]
9.	Hash Tables	[17]

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: .....

Question 1. Basic Questions [30 marks]

(a) [2 marks] In which collection type can you add or access a value via an associated key?

A Map

(b) [2 marks] What is the average case Big-O cost of searching for an item in a Set of n items implemented using an unsorted array?

$O(n)$

(c) [2 marks] Name a sorting algorithm with an average case Big-O cost of $O(n^2)$.

Insertion sort, or Selection sort, or Bubble sort

(d) [2 marks] What is the average case Big-O cost of adding an item to a set of n items implemented using a Hash Table with linear probing, if the Hash Table is less than half full.

O(1)

Consider the diagram below of a variable containing a linked list. Assume that each node of the list has the fields item and next.

(e) [2 marks] What is the value of names.next.item?

"Jane"

(f) [2 marks] What is the value of names.next.next.next.item?

"Julie"

(Question 1 continued on next page)

(Question 1 continued)

(g) [2 marks] Suppose we push the item "A" onto an empty stack, and then push "B", and then "C". If we now pop an item from the stack, which items will still be on the stack?

"A" and "B"

Consider the following general tree:

(h) [2 marks] What value is in the parent of the node with the value "R"?

"F"

(i) [2 marks] How many leaf nodes does the tree have?

5

(j) [2 marks] Why is the tree NOT a binary tree?

Because node E has three children

(Question 1 continued on next page)

(Question 1 continued)

(k) [3 marks] Draw a Binary Search Tree containing the values 56, 15, 31, 47, 82, and 23.

(l) [2 marks] If we are adding a value to a Hash Table using buckets, and the first bucket we look at already contains a value, where do we put the value?

In the bucket

(m) [3 marks] If we are adding a value to a Hash Table using linear probing ("open addressing"), and the first cell we look at already contains a value, where do we put the value?

In the next cell to the right that doesn't have a value

(n) [2 marks] Draw a directed graph containing four nodes and five edges.

Question 2. Programming with Priority Queues [23 marks]

This question concerns a ManagePackages program that helps a courier company keep track of the packages it has to deliver. The company processes packages in order of their priority, so the ManagePackages program uses a priority queue.

The program has three actions:

New Package Asks the user for the details of a new package, adds it to the priority queue, and prints out its details.

Next Package Removes the package at the front of the priority queue and prints out the details of the package, or a useful message if there are no packages to deliver.

Report Queue Prints the number of packages in the priority queue, and all their details.

The ManagePackages class on the facing page contains an incomplete method for each of these actions.

The ManagePackages class uses the Package class described below, and the standard Queue interface described in the appendix.

```
public class Package implements Comparable<Package>{
    private String category;          //One of "urgent", "same day", or "next day"
    private int distanceToTravel;     //number of kilometers to delivery address
        ⋮

    /** Constructor asks the user for the details of a new package
        and fills in the field values */
    public Package(String prompt){
            ⋮
    }
    /** Returns a String with all the details of the package */
    public String getDetails(){
            ⋮
    }

        ⋮
}
```

(Question 2 continued on next page)

(a) [12 marks] Complete the three methods in the ManagePackages class below.

```
public class ManagePackages {
    // Priority queue of all packages
    private Queue <Package> packages = new HeapQueue<Package>();
        ⋮
    public ManagePackages(){
        ⋮
    }
    /** Get new package from user, add it to the queue,
        print details to System.out */
    public void enqueueNewPackage(){
        Package p = new Package("Enter details for new package")
        packages.offer(p);
        System.out.println("Enqueued: "+p.getDetails());

    }
    /** Take package off front of queue and print out details of the package.
        If queue is empty, print "No Packages" */
    public Package NextPackageToProcess(){
        Package p = packages.poll();
        if (p == null)
            System.out.printlntextArea.append("No packages");
        else
            System.out.printlntextArea.append(p.getDetails());

    }
    /** Print how many packages are waiting, and all their details */
    public void reportQueue(){
        System.out.println(packages.size() +" packages yet to process:");
        for (Package p : packages)
            System.out.println(p.getDetails());

    }
}
```

(Question 2 continued on next page)

(Question 2 continued)

The priority of packages is based on two factors:

- the category of service, and
- the distance the package has to go (in kilometers).

There are three categories of service: "urgent", "same day", and "next day". All the "urgent" packages are higher priority than the "same day" packages, which are higher priority than the "next day" packages. Within each category, the packages that have the greatest distance to go have the highest priority.

(b) [5 marks] Two efficient implementations of a priority queue are:

- an array of Queue, indexed by the priority, and
- a HeapQueue, using a partially ordered tree in an array.

Given the above definition of priority, why would the first implementation be a bad choice for the ManagePackages program?

Because there are a very large number of possible priorities

(c) [6 marks] The HeapQueue requires that the values inserted into the queue are Comparable, and that higher priority items are considered to be larger than lower priority items. Therefore, the Package class must have a compareTo(*Package* other) method that returns a positive number if this object has a higher priority than other, a negative number if this object has a lower priority than other, and 0 if they have the same priority. The priority depends on the values of the category and distanceToTravel fields.

Complete the following compareTo method for the Package class.

```
public int compareTo(Package other){
    if (category.equals(other.category))
        return distanceToTravel – other.distanceToTravel;
    else
        return category.compareTo(other.category);




}
```

Question 3. Linked Lists [20 marks]

The first part of the LinkedNode class covered in lectures is given below, along with an example of a linked list. You are to complete three methods for this class.

```
public class LinkedNode<E> {
    // Fields
    private E value;
    private LinkedNode <E> next;

    // Constructor
    public LinkedNode(E item, LinkedNode<E> nextNode){
        value = item;
        next = nextNode;
    }
    ⋮
```

(a) [5 marks] Complete the following size method from the LinkedNode class so that it returns the number of nodes in a linked list.

```
public int size(){
    int ans = 0;
    for (LinkedNode<E> rest = this; rest != null; rest = rest.next)
        ans++;
    return ans;
}
OR
public int size(){
    if (next == null) return 1;
    else return 1 + next.size();
}
```

(Question 3 continued on next page)

(Question 3 continued)

(b) [5 marks] The append method joins another linked list on the the end of a linked list. For example, given these two lists

vowels.append(tall) should result in

Complete the following append method.

```
/** Appends another list to the end of this list */
public void append(LinkedNode <String> other){
    if (next == null)
        next = other;
    else
        next.append(other);
  or
    LinkedNode<E> rest = this;
    while(rest.next != null) rest = rest.next;
    rest.next = other;



}
```

(Question 3 continued on next page)

(Question 3 continued)

(c) [5 marks] Assuming that your append method works as specified, and tall is the list shown above, what would the following code print out?

```
tall.append(tall);
for (LinkedNode<String> nd = tall; nd != null; nd = nd.next()){
    System.out.println(nd.get());
```

It would print out `b d f h k l t b d f h k l t b d` ...endlessly

(d) [5 marks] (Harder) Complete the following reverse method so that it returns a list that is the reverse of the original list. The method may construct a new list or it may reverse all the pointers in the original list.

```
public LinkedNode<E> reverse(){
    if (next == null) return this;
    LinkedNode<E> ans = null;
    LinkedNode<E> rest = this;
    while(rest != null){
        ans = new LinkedNode<E> (rest.value, ans);
        rest = rest.next;
    }
    return ans;



}
```

Question 4. Analysis of Algorithms [23 marks]

(a) [6 marks] Explain why we typically express the cost of algorithms for collections in Big-O notation instead of in milliseconds or exact numbers of operations.

The cost of running a program that implements an algorithm will depend on many different factors: the size of the collection, the way the program was written, the speed of the computer. Measuring in milliseconds would vary with all these factors, and would not tell us much about the algorithm. Measuring in number of operations would be better, but would have more detail than is necessary, since different computers would take a different time for the same program. Expressing the cost of an algorithm in Big-O notation abstracts from these factors, but still enables useful comparison between algorithms.

(b) [6 marks] For each of the following implementations of a Set, give the Big-O average case cost of the contains and add methods. State any assumptions you make.

Implementation	Cost of contains	Cost of add	
ArraySet (unsorted)	$O(n)$	$O(n)$	
SortedArraySet	$O(\log(n))$	$O(n)$	
LinkedListSet (sorted)	$O(n)$	$O(n)$	
BSTSet (Binary Search Tree)	$O(\log(n))$	$O(\log(n))$	assuming it is balanced
HashSet (with linear probing)	$O(1)$	$O(1)$	assuming it is not close to full
HashSet (with k buckets)	$O(n/k)$	$O(n/k)$	

(Question 4 continued on next page)

(Question 4 continued)

(c) [5 marks] Under what condition will a Binary Search Tree have bad performance? Explain why.

Condition: When the BST is unbalanced, so that the maximum depth of the tree is O(n).

Explanation: When adding a node that belongs near the end of the long branch, the cost will be O(n). This bad performance can happen if the items added are already in sorted order.

(d) [6 marks] Under what conditions will a Hash Table using linear probing have bad performance? Explain why.

Condition: Whenever the hashtable is close to full.

Explanation: Because there will be many collisions, so that adding an item will require looking at many cells.
(Extra marks: Also, if the hash function is bad so that many items hash to the same value.)

Question 5. Implementing Collections with Arrays [9 marks]

(a) [5 marks] The sorted and unsorted array implementations of List and Set increase the capacity of the collection whenever the array becomes full.

Assume that you are implementing a collection using an array, and the collection has two fields:

```
private E[ ] data;
private int count = 0;
```

Complete the following ensureCapacity method so that it "doubles and copies" the data array when it is full, but does nothing if the array is not full.

```
private void ensureCapacity () {
    if (count < data.length) return;
    E [ ] newArray = (E[ ])(new Object[data.length*2]);
    for (int i = 0; i < count; i++)
        newArray[i] = data[i];
    data = newArray;



}
```

(b) [4 marks] If a collection contains n items, the cost of copying the items to a new array is $O(n)$. Explain why the average cost of adding an item to the end of a list is constant rather than $O(n)$.

When the cost of doubling an array of size n is amortised (averaged) over the next n additions, the cost of adding an item is still just O(1).

Question 6. Implementing Sets with Binary Search Trees [24 marks]

(a) [2 marks] State the property that a Binary Tree must satisfy to be a Binary Search Tree.

The value in each node must be greater than (or equal to) every value in the left subtree of the node, and less than (or equal to) every value in the right subtree of the node.

(b) [4 marks] Consider the following diagram of a Set of numbers implemented as a Binary Search Tree. Show, on the diagram, what the tree would look like if the values 2, 5, 20, and 56 were added to the set.

(Question 6 continued on next page)

(Question 6 continued)

(c) [6 marks] Show, on the diagram, what the tree would look like if the values 32, 7, 57, and 19 were removed from the Set.

(d) [12 marks] Assuming the declarations below, complete the code on the facing page for adding a value to a Set implemented using a Binary Search Tree.
You may assume that the value is not already in the set, and that the values are Comparable (ie, you can use the compareTo method on values).

```
public class BSTSet<E> {
    // data fields
    private BSTNode<E> root;

        ⋮

    //BSTNode class
    private class BSTNode<E> {
        private E value;
        private BSTNode<E> left;
        private BSTNode<E> right;

        public BSTNode(E v){
            value = v;
        }

        ⋮
    }
}
```

```
// in the BSTSet class ...
   /* Adds an item to the set. */
   /* Assumes that item is not present */
   public void add (E item) {
      if (item == null) return;
      if (root == null)
         root = new BSTNode<E> (item);

      else
         root.add(item)
   }
```

```
// in the BSTNode class ...
   /* Adds an item to the binary search tree. */
   /* Assumes that item is not present */
   public boolean add(E item){
      if (item.compareTo(value) < 0)}
         if (left == null)
            left = new BSTNode<E>(item);
         else
            left.add(item);
      }
      else {
         if (right == null)
            right = new BSTNode<E>(val);
         else
            right.add(val);
      }


   }
```

Question 7. Implementing Priority Queues [14 marks]

The HeapQueue class is an implementation of priority queues using a heap — a complete, partially ordered binary tree in an array.

(a) [2 marks] State the property a binary tree must satisfy in order to be a partially ordered binary tree.

The value in each node must be less than (or equal to) the values in its children.

(b) [2 marks] If a complete partially ordered binary tree is stored in an array, and the index of the root is 0, what are the indexes of the children of the node at index *i*?

(2 * i + 1) and (2 * i + 2)

Consider the following HeapQueue of letter-integer pairs, where the integer represents the priority (larger numbers are higher priority).

(c) [4 marks] Show the state of the HeapQueue if the pair P-21 is added to the HeapQueue:

(Question 7 continued on next page)

(Question 7 continued)

(d) [6 marks] Complete the following pushup(*int* i) method that moves the value at index i up the partially ordered tree into its correct position.
Assume that the HeapQueue contains the following fields:

```
private E[ ] data;
private int count;
private Comparator<E> comp;
```

where comp is a comparator that considers values with higher priority to be larger than values with lower priority.

```
private void pushup(int i){
    if (i == 0) return;
    int parent = (i–1)/2;
    if (comp.compare(data[parent], data[i]) < 0)}
        E temp = data[i];
        data[i] = data[parent];
        data[parent] = temp;
        pushup(parent);
    }



}
```

Question 8. Trees

[20 marks]

(a) [4 marks] Write out the order in which nodes would be visited by a depth-first, left-to-right, post-order traversal of the following general tree.

T, L, S, Z, C, X, R, V, K, E, D

(b) [2 marks] What are the minimum and maximum depths of a binary tree with 15 nodes?
Assume the depth is equal to the number of levels in a tree, and that the root node is at level one.

Minimum Depth: 4

Maximum Depth: 15

(c) [3 marks] What is the worst-case Big-O (asymptotic) cost of finding a particular value in a binary tree of depth d?
Assume the depth is the number of levels in a tree, and that the root node is at level one.

Cost: $O(2^d)$

(Question 8 continued on next page)

(Question 8 continued)

(d) [5 marks] The following traversal method below does a depth first traversal of a binary tree using a stack.

```
public static void traversal(TreeNode start){
    Stack<TreeNode> toVisit = new Stack<TreeNode>();
    toVisit.push(start);
    while (! toVisit.isEmpty()){
        TreeNode current = toVisit.pop();
        System.out.println(current.getValue());
        if (current.getLeft() != null)
            toVisit.push(current.getLeft());
        if (current.getRight() != null)
            toVisit.push(current.getRight());
    }
}
```

Suppose traversal is called on the following tree:

Draw the contents of the stack immediately after node "Z" has been added to the stack.

Stack: (from top of stack [S, N, Z]

Note, [K] would be the answer if it were a left-to-right traversal

(Question 8 continued on next page)

(Question 8 continued)

(e) [6 marks] Complete the following getWidth() method so that it returns the number of leaf nodes in a GeneralTree.

```
public class GeneralTree<E> {
    private Set<GeneralTree<E>> children;
    private E value;

        ⋮

    public int getWidth(){
        int ans = 0;
        if (children.isEmpty())
            ans = 1;
        else
            for (GeneralTree<E> ch : children)
                ans = ans + ch.getWidth();
        return ans;



    }
}
```

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 9. Hash Tables [17 marks]

(a) [3 marks] Insert the values 'A', 'X', and 'N' into the Hash Table below, using linear probing to resolve collisions. Assume that 'A' hashes to the index 3, 'X' hashes to 6, and 'N' hashes to 8.

(b) [3 marks] Linear probing and quadratic probing are both techniques to resolve hash collisions. Why is quadratic probing better than linear probing?

It prevents runs from joining together to make long runs

(c) [3 marks] Typically we increase the capacity of a Hash Table with probing when it is only 70% full. Why don't we wait until the Hash Table is 100% full?

Because the performance, especially for adding new items, becomes very slow when the Hash Table is nearly full, because there are so many collisions.

(d) [5 marks] Draw the contents of the array after the following 10 values are inserted into a Hash Table that uses an overflow area to handle collisions.
Note: do *not* increase the size of the array if the table becomes too full.

value:	'D'	'A'	'J'	'K'	'E'	'S'	'L'	'M'	'Z'	'Y'
hashed index:	1	6	3	6	0	5	1	6	4	1

(Question 9 continued on next page)

(Question 9 continued)

(e) [3 marks] What is wrong with the following hash function?

```
public int hashFunction(String key) {
    int hash = 1;
    char[ ] characters = key.toCharArray();
    for (int i = 0; i < characters.length; i++)
        hash = hash + (int) (characters[i] * 256 * Math.random());
    return (hash % data.length); // data is the hash table array
}
```

The use of Math.random means that each time a value is hashed, it will produce a different value. This is useless because it means that the hash table will not be able to find an item again after it has inserted it

****

Appendices

Possibly useful formulas:

- $1 + 2 + 3 + 4 + \cdots + k \;=\; \frac{k(k+1)}{2}$
- $1 + 2 + 4 + 8 + \cdots + 2^k \;=\; 2^{k+1} - 1$
- $a + (a+b) + (a+2b) + \cdots + (a+kb) \;=\; \frac{(2a+kb)(k+1)}{2}$
- $a + as + as^2 + as^3 + \cdots + as^k \;=\; \frac{as^{k+1}-a}{s-1}$

Table of base 2 logarithms:

n	1	2	4	8	16	32	64	128	256	512	1024	1,048,576
$\log(n)$	0	1	2	3	4	5	6	7	8	9	10	20

Brief (and simplified) specifications of relevant interfaces and classes.

```
public interface Iterator<E> {
    public boolean hasNext();
    public E next();
    public void remove();
}
public interface Comparable <E>{
    public int compareTo(E o);
}
public interface Comparator <E>{
    public int compare(E o1, E o2);
}
public class Math{
    public static double random(); //return a random number between 0.0 and 1.0
}
```

```
public interface Collection <E>{
    public boolean isEmpty ();
    public int size ();
    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 (E element);
    public void add (E element);
    public void remove (Object 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 ();
}
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 class Arrays {
    public static <E> void sort(E[ ] ar, Comparator<E> comp);
}
public class Collections {
    public static <E> void sort(List<E> list, Comparator<E> comp);
}
```

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.