



EXAMINATIONS — 2007

SUMMER TRIMESTER

COMP103 Introduction to Data Structures and Algorithms

WITH SOLUTIONS!!!

Time Allowed: 3 Hours

- Instructions:**
1. Attempt **all** of the questions.
 2. *Read each question carefully before attempting it.* (Suggestion: You do not have to answer the questions in the order shown. Do the questions you find easiest first.)
 3. This examination will be marked out of **180** marks, so allocate approximately 1 minute per mark.
 4. Write your answers in the boxes in this test paper and hand in all sheets.
 5. Non-electronic Foreign Language — English translation dictionaries are permitted.
 6. Calculators are allowed.
 7. There is documentation on the relevant Java classes and interfaces at the end of the exam paper.

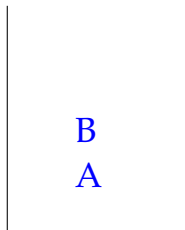
WARNING: THIS COPY CONTAINS SOLUTIONS!!!

Questions	Marks
1. Basic Questions	[20]
2. Implementing a Collection	[13]
3. Performance and Costs	[17]
4. Sorting Linked Lists	[21]
5. Trees	[37]
6. Priority Queues and Heaps	[19]
7. Hashing	[28]
8. Graphs	[25]

Question 1. Basic Questions

[20 marks]

(a) [2 marks] Suppose we add items A , B and C to an empty stack, in that order, and then remove one of them (using `pop`). Draw the contents of the stack, after the execution of `pop`.



Stack bottom.

(b) [2 marks] How many comparisons (on average) does it take to remove an element from a set of n elements implemented using an unsorted array?

$$(n + 1)/2$$

(c) [2 marks] What is the *average-case* asymptotic (“big- O ”) cost of insertion sort?

$$O(n^2)$$

(d) [2 marks] Name a sorting algorithm with *worst-case* asymptotic cost of $O(n \log(n))$

Mergesort or Heapsort

(e) [2 marks] What do we mean when we say that a sorting algorithm is *stable*?

The order of equal values is preserved.

(Question 1 continued on next page)

(Question 1 continued)

(f) [2 marks] What is the *worst-case* number of comparisons required to find a node in a binary search tree if the tree contains n items?

n

(g) [2 marks] What type of data-structure (in addition to the tree) should you use to implement a depth-first traversal of a tree?

A Stack

(h) [2 marks] What is the *average-case* asymptotic cost of HeapSort?

$O(n \log(n))$

(i) [2 marks] What is the defining property of a perfect hash function?

A perfect hash function doesn't produce collisions.

(j) [2 marks] What does it mean to say a graph is *weighted*?

Each edge has a corresponding numerical value.

Question 2. Implementing a Collection

[13 marks]

For this question, you are to complete some of the methods of an `ArrayStack` class, which is an implementation of a stack using an array.

The header and fields of the class are given below:

```
public class ArrayStack<E> extends AbstractStack<E> {  
    private static int INITIALCAPACITY = 8;  
    private int count = 0;  
    private E[ ] data;
```

(a) [3 marks] Complete the following constructor, which should initialise the `data` field.

```
public ArrayStack(){  
    data = (E[ ]) new Object[INITIALCAPACITY];  
  
}
```

(b) [2 marks] Complete the `isEmpty` method, which returns `true` if and only if the stack is empty.

```
public boolean isEmpty(){  
    return count == 0;  
  
}
```

(Question 2 continued on next page)

(Question 2 continued)

(c) [3 marks] Complete the following push method, which adds a value onto the end of the stack. Assume that ArrayStack has an ensureCapacity() which “doubles and copies” the data array if the array is full:

```
public void ensureCapacity() {...}
```

```
public void push(E item){
    ensureCapacity();
    data[count++] = item;
}
```

(d) [5 marks] Complete the following pop method which removes and returns the value at the top of the stack if the stack is not empty, and otherwise throws an EmptyStackException.

```
public E pop(){
    if (isEmpty())
        throw new EmptyStackException();

    return data[--count];

    /*OR: */

    if (isEmpty())
        throw new EmptyStackException();

    count--;
    E ans = data[count]; // This version over-writes references for the
    data[count] = null; // sake of GC.
    return ans;
}
```

Question 3. Performance and Costs

[17 marks]

(a) [6 marks] In the table below, state the asymptotic cost measure for each queue implementation and for each case: worst case of offer, average case of offer, and worst case of poll. Assume that n is the number of elements in the queue.

	Worst case offer	Average case offer	Worst case poll
Using an array with indexes for the head and tail of the queue	$O(n)$	$O(1)$	$O(1)$
Using a linked list, with a reference only to the first node (head of the queue)	$O(n)$	$O(n)$	$O(1)$
Using a linked list, with references to both the first and last nodes	$O(1)$	$O(1)$	$O(1)$

The rest of this question concerns two implementations of a method called findLeastM. The findLeastM method takes a `Set<Integer> set` and an `int m`, and returns a `List<Integer>` containing the m least values in `set`. You do not need to understand precisely how the code works in order to answer the questions below.

Version 1 (below) works by sorting `set`, using mergesort, then takes the first m values (or `set.size()` values, if m is greater than `set.size()`) from that sorted list and putting them into `result`.

```
public List<Integer> findLeastM(Set<Integer> set, int m) {  
    List<Integer> sorted = mergeSort(set);  
  
    List<Integer> result = new ArrayList<Integer>();  
    for (int i = 0; i < Math.min(m, set.size()); i++)  
        result.add(sorted.get(i));  
  
    return result;  
}
```

(b) [4 marks] If `set.size()` is n , what is the asymptotic cost of Version 1 of findLeastM, in terms of m and n ?

$O(n \log(n))$

(Question 3 continued on next page)

(Question 3 continued)

Version 2 (below) works by iterating through `set`, and checking to see if each element should be put into `result`.

```

public List<Integer> findLeastM(Set<Integer> set, int m) {

    List<Integer> result = new ArrayList<Integer>();
    if (m == 0) return result;

    for (Integer num: set) {
        if (result.size() < m) // result not full yet
            result.add(num);
        else { // find the index of the largest item in result
            int maxIndex = 0;
            for (int index = 1; index < m; index++) {
                if (result.get(maxIndex) < result.get(index))
                    maxIndex = index;
            }
            // if num is less than the largest item in result,
            // then replace the largest item with num
            if (num < result.get(maxIndex))
                result.set(maxIndex, num);
        }
    }
    return result;
}

```

(c) [4 marks] If `set.size()` is n , what is the asymptotic (“big-O”) cost of Version 2 of `findLeastM`, in terms of m and n ?

$O(mn)$

(d) [3 marks] In what situations would Version 2 be better than Version 1?

Full marks for the answer: “ $m < \log(n)$ ”, or “ m much smaller than n ”
but the exact answer depends on the constant factors.

Question 4. Sorting Linked Lists

[21 marks]

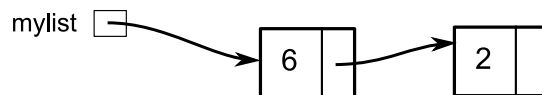
This question is about operations on linked-lists, represented as `LinkedListNode<E>`. The `LinkedListNode<E>` class is defined as follows.

```
class LinkedListNode<E>{
    public E value;
    public LinkedListNode<E> next;

    public LinkedListNode(E v, LinkedListNode<E> n) {
        value = v;
        next = n;
    }
}
```

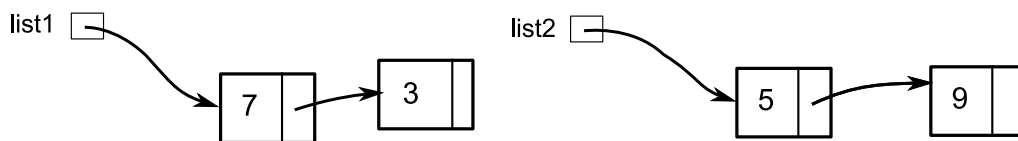
(a) [2 marks] The diagram below depicts a variable `mylist` that holds a reference to a linked list. Draw a diagram of the list after the following code is executed.

```
mylist = new LinkedListNode<Integer>(7, mylist);
```



mylist

(b) [3 marks] The diagram below depicts two variables, `list1` and `list2`, that hold references to two lists.



Draw a diagram showing the resulting lists if the following code is executed.

```
LinkedListNode<Integer> x = list1;
x = x.next;
x.next = list2 ;
```

list1 list2

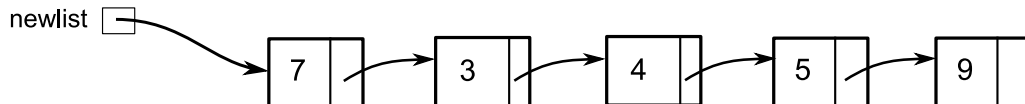
(Question 4 continued on next page)

(Question 4 continued)

(c) [7 marks] Complete the following `append` method which has three parameters: a linked list, an individual value, and a second linked list. The method should return a linked list that contains the values in the first list, then the individual value, then the values in the second linked list.

Your method is allowed to modify the contents of the lists in its parameters.

For example, if `list1` and `list2` are the lists shown in question 4(b), executing
`newlist = append(list1, 4, list2)`
 should result in the list depicted below.



```

public ListNode<E> append(ListNode<E> first, E value, ListNode<E> second) {
    if ( first == null)
        return new ListNode<E>(value, second);

    ListNode<E> temp = first;
    while (temp.next != null)
        temp = temp.next;

    temp.next = new ListNode<E>(value, second);

    return first ;
*OR*
    ListNode<E> ans = new ListNode<E>(first.value, null);
    ListNode<E> tail = ans;
    for (ListNode<E> temp = first.next; temp != null; temp=temp.next){
        tail .next = new ListNode<E>(temp.value, null);
        tail = tail .next;
    }
    tail .next = new ListNode<E>(value, null);
    tail = tail .next;
    for (ListNode<E> temp = second; temp != null; temp=temp.next){
        tail .next = new ListNode<E>(temp.value, null);
        tail = tail .next;
    }
    return ans;
}
  
```

(Question 4 continued on next page)

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 4 continued)

(d) [9 marks] The version of Quicksort presented in the lectures used arrays, and the partition algorithm was quite tricky. It is also possible to implement Quicksort using linked lists, and the partition is much simpler.

To sort a linked list, this version of Quicksort will take the first value out of the list as the pivot, then step through the rest of the list, constructing two new lists which respectively contain all the values less than the pivot and all the values greater than or equal to the pivot. It then recursively sorts the two new lists, and then appends them together, with the pivot in the middle.

Complete the following version of quicksort for linked nodes.

```

public ListNode<E> quicksort(ListNode<E> list, Comparator<E> comp) {
    if ( list == null) return null;

    // extract the pivot
    E pivot = list .value;
    list = list .next;

    // initialise two lists :
    ListNode<E> low = null; // will contain all the values < pivot
    ListNode<E> high = null; // will contain all the values >= pivot

    // step through list , constructing low and high.

    while ( list != null) {
        if (comp.compare(list.value,pivot) < 0)
            low = new ListNode<E>(list.value, low);
        else
            high = new ListNode<E>(list.value, high);

        list = list .next;
    }

    // sort the two lists
    low = quicksort(low, comp);
    high = quicksort(high, comp);
    // append the lists and the pivot (using method from part (c))
    return append(low, pivot, high);
}

```

Question 5. Trees

[37 marks]

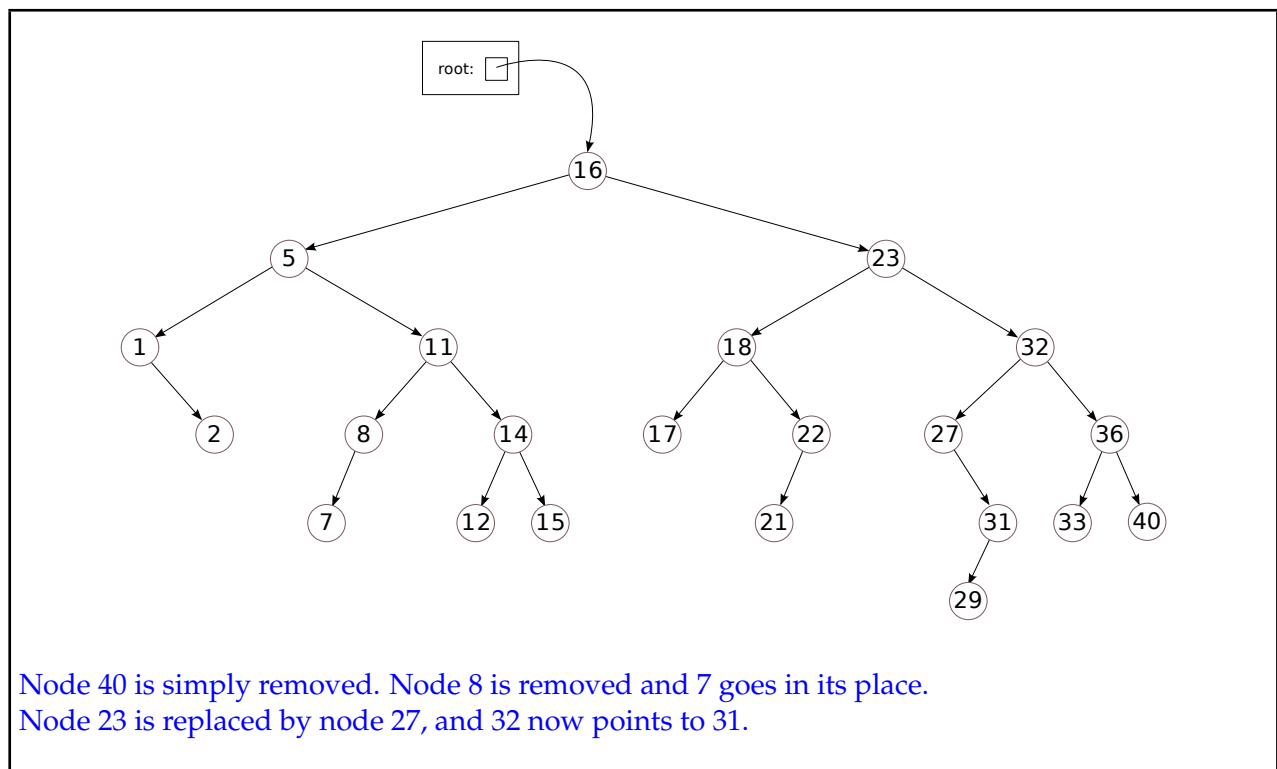
(a) [2 marks] What is the maximum number of values that can be stored in a binary tree with 5 levels?

31

(b) [3 marks] In a binary search tree, what is the relationship between the value in a node and the values in its descendants?

The node's value is greater than all the values in its left subtree and less than (or equal to) all the values in its right subtree.

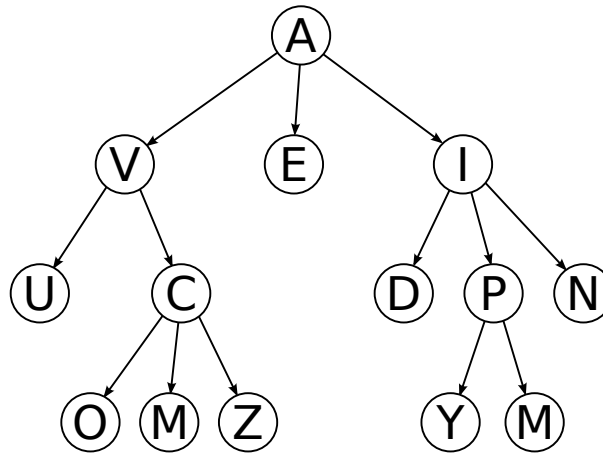
(c) [5 marks] Consider the following diagram of a Set of numbers implemented as a Binary Search Tree. Show, on the diagram, what the tree would look like after the values 40, 8 and 23 are removed from the tree using the algorithm described in lectures.



(Question 5 continued on next page)

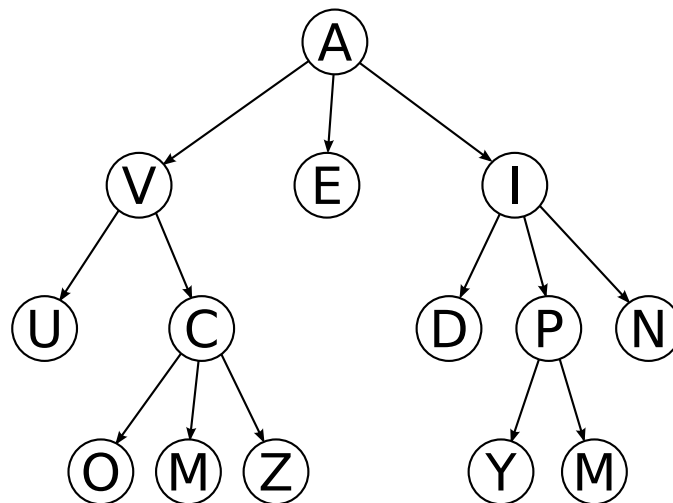
(Question 5 continued)

(d) [3 marks] The following diagram shows a General Tree which has been used to implement a Set. List the nodes in the order in which they will be visited in a depth-first, pre-order, left-to-right, search for the value 'D' (only list nodes which are visited).



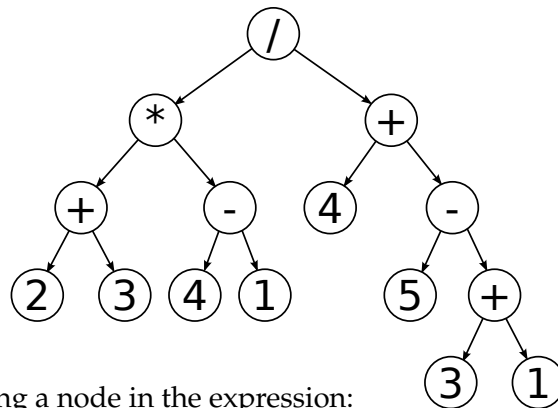
Order visited: A, V, U, C, O, M, Z, E, I, D.

(e) [4 marks] The following diagram shows a General Tree. If a queue is used to implement a breadth-first left-to-right traversal show *the contents of the queue* at the point when the node with value 'D' is visited.



Queue: (D,) P, N, O, M, Z

(f) [8 marks] Trees are often used to do parsing. The following diagram is an example parse tree of an arithmetic expression.



Here is the class representing a node in the expression:

```
private class Node {
    public String value;
    public Node left;
    public Node right;
}
```

Complete the code below which takes a parse tree of an arithmetic expression and prints out the expression in normal mathematical form using round brackets for each sub-expression. For example, given the tree above, the method should print:

$((2 + 3) * (4 - 1)) / (4 + (5 - (3 + 1)))$

Assume all the operators are binary operators (eg, +, *, -, and /).

```
public void print(Node node) {
    if (node.left != null) {
        System.out.print("(");
        print(node.left);
        System.out.print(" " + node.value + " ");
        print(node.right);
        System.out.print(")");
    }
    else {
        System.out.print(node.value);
    }
}

}
```

(g) [12 marks] Suppose we implement a partially ordered binary tree of integers where smaller integers have higher priority (nearer the root of the tree).

```
public class PartiallyOrderedTree {
    private POTNode root;
    private class POTNode {
        private int value;
        public POTNode left;
        public POTNode right;
    }
    /* Swaps the values of two nodes */
    private void swap(POTNode a, POTNode b) {
        int v = a.value; a.value = b.value; b.value = v;
    }
}
```

Complete the following implementation of the *pushDown(POTNode node)* method for *PartiallyOrderedTree*. The method should recursively swap nodes in the tree to restore the partially ordered property. Your method may assume that the subtrees are partially ordered, but the value in the root node is out of place. You should use the *swap* method to exchange the values in two nodes.

```
public void pushDown(POTNode node) {
    if (node.left == null && node.right == null) return;

    else if (node.left == null) {
        if (node.right.value < node.value) {
            swap(node.right, node);
            pushDown(node.right);
        }
    }
    else if (node.right == null) {
        if (node.left.value < node.value) {
            swap(node.left, node);
            pushDown(node.left);
        }
    }
    else if (node.right.priority < node.left.priority) {
        if (node.right.value < node.value) {
            swap(node.right, node);
            pushDown(node.right);
        }
    }
    else {
        else if (node.left.value < node.value) {
            swap(node.left, node);
            pushDown(node.left);
        }
    }
}
```

Question 6. Priority Queues and Heaps

[19 marks]

(a) [8 marks] Assignment 9 asked you to implement a priority queue interface using several different data structures and then compare the performance of your implementations with their theoretical (asymptotic) performance.

Complete the following table, giving the average case asymptotic cost for enqueue and dequeue with each type of priority queue.

You may assume that it is not necessary to preserve FIFO order for values with the same priority.

	Unsorted Array	Sorted Array	Array of Queues	Heap
Enqueue	$O(1)$	$O(n)$	$O(1)$	$O(\log(n))$
Dequeue	$O(n)$	$O(1)$	$O(p)$	$O(\log(n))$

(b) [2 marks] Explain when you should choose to use a Heap rather than an Array of Queues.

Array of Queues is good when there are a small, finite number of priorities. When there are many different priorities then a heap is both more memory efficient and faster.

(c) [3 marks] A Heap is a complete partially-ordered tree which is stored in an array. State two advantages of using an array rather than linked nodes.

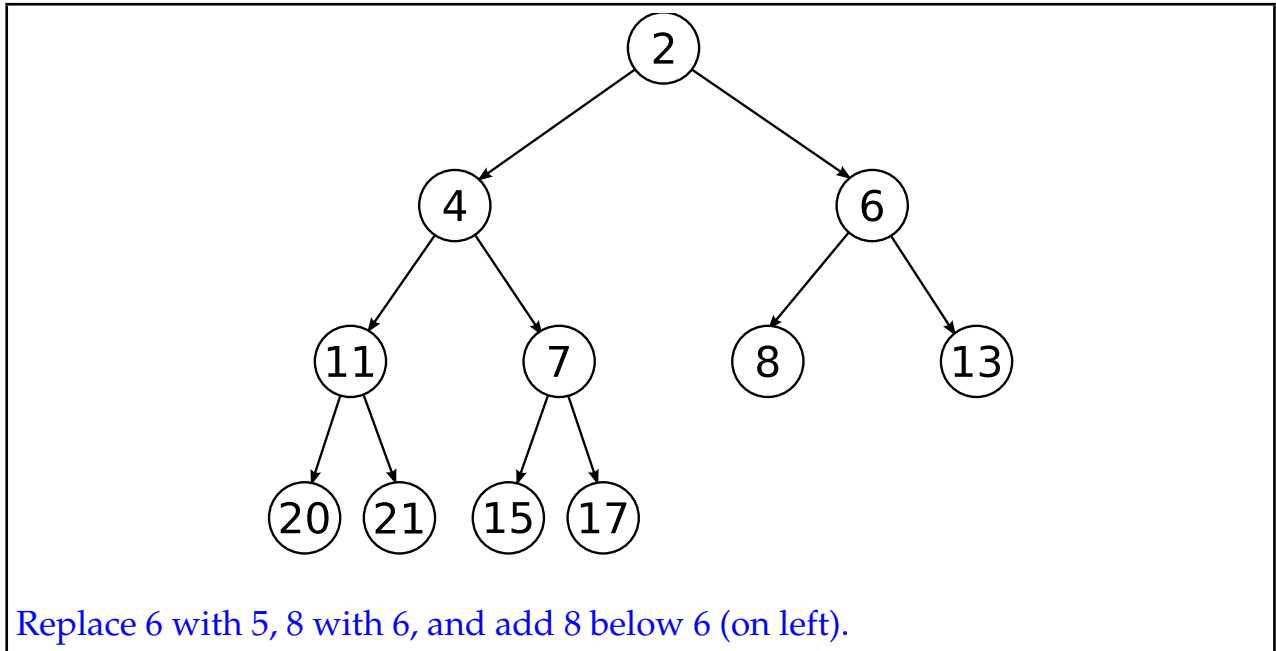
Any two of

- less memory required
- less pointer complexity
- faster (less memory access time)

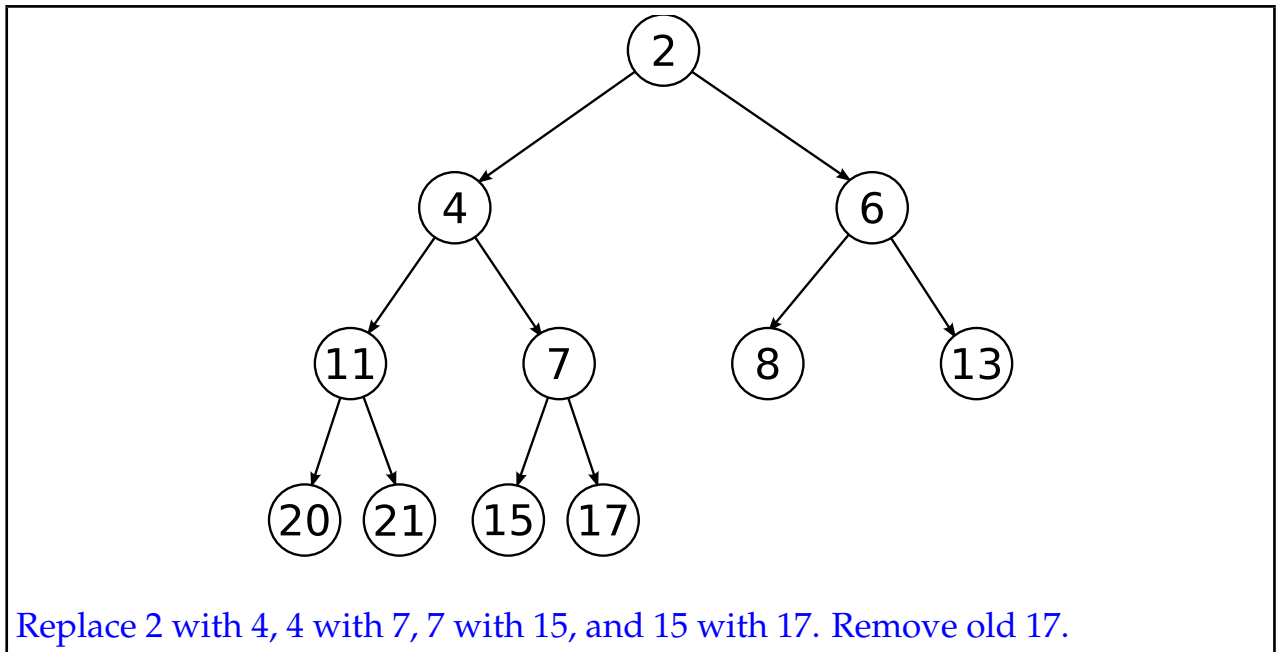
(Question 6 continued on next page)

(Question 6 continued)

(d) [3 marks] Draw the changes to the following complete partially-ordered tree when the value 5 is added.



(e) [3 marks] Draw the changes to the following complete partially-ordered tree when the value 2 is removed.



Question 7. Hashing

[28 marks]

(a) [4 marks] The following array represents a Hash Set which has been partially filled. Using linear probing, insert the following values into the Hash Set:

- S, hash value 10
- E, hash value 4
- F, hash value 5
- P, hash value 9

You do not need to increase the size of the array.

V	A	P		N	E	R	F		G	S
0	1	2	3	4	5	6	7	8	9	10

(b) [2 marks] Explain why removing elements from a Hash Set with open addressing is more challenging than adding elements.

Element may be part of a chain, and removing the element would make other values inaccessible.

(c) [2 marks] Describe an efficient solution to the problem identified in (b).

Replace the removed element with a placeholder. The placeholder cannot be returned but indicates that probing should continue. When a new value is added it can replace the placeholder.

(d) [3 marks] Demonstrate your solution to (b) by removing the value 'V' (hash value 5) and 'F' (hash value 9) from the following Hash Set. Show clearly the state of the hash set after the values have been removed.

K	F		H	Q	V	W			S	E
0	1	2	3	4	5	6	7	8	9	10

K	pl		H	Q	pl	W			S	E
0	1	2	3	4	5	6	7	8	9	10

(e) [6 marks] Implement the `hashCode()` method for the following two classes. You may use Java's `hashCode()` method for computing hash codes for fields.

```
public class Employee {
    private String firstName;
    private String lastName;
    private Date dateOfBirth;
    ...
    public boolean equals(Employee e) {
        return e.firstName.equals(firstName)
            && e.lastName.equals(lastName)
            && e.dateOfBirth.equals(dateOfBirth);
    }
}
```

```
private int hashCode() {
    int hash = firstName.hashCode();
    hash *= 1021;
    hash += lastName.hashCode();
    hash *= 1021;
    hash += date.hashCode();
    return hash;
}
```

```
public class StockItem {
    private String description;
    private String manufacturer;
    private int stock;
    private double price;
    private double markup;
    private int id;
    ...
    public boolean equals(StockItem i) {
        return i.id == this.id;
    }
}
```

```
private int hashCode() {
    return id;
}
```

(Question 7 continued on next page)

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 7 continued)

(f) [3 marks] When a Hash Map which uses open addressing approaches the maximum capacity of its array the performance of the Hash Map degrades. To prevent this, we try to increase the size of the array before performance hit becomes severe. Suggest two metrics for deciding when to increase the size of the Hash Map's array.

- When the array is 70-80% full (of nodes).
- When the array is 70-80% full (of nodes and placeholders).
- When the length of the longest probe chain reaches X
- When a certain ratio of placeholders to nodes is reached.
- Any other reasonable answer.

(g) [8 marks] Implement a method which grows the Hash Map by increasing the size of the array and moving existing values into the new array correctly.

Hint: your method may use the *put()* method!

```
public HashMap<K, V> {
    private class Entry {
        K key;
        V value;
    }
    private Entry[] entries;
    private int size;

    public boolean put(K key, V value) { ... }
    ...

```

```
private void grow() {
    Entry[] old = entries;
    int capacity = entries.size()*2 + 1;
    entries = new Entry[capacity];
    size = 0;
    for (Entry e: old) {
        if (e != null && e.key != null) {
            put (e.key, e.value);
        }
    }
}

```

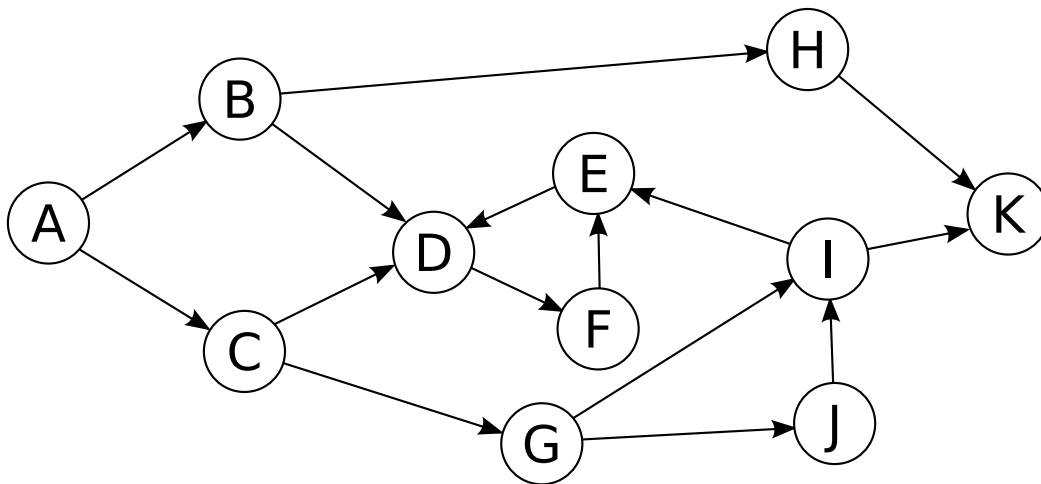
Question 8. Graphs

[25 marks]

(a) [2 marks] What is a path in a graph?

A Path is a sequence of edges (without duplicates is acceptable).

(b) [2 marks] How many paths are there from node A to node K in the following graph?



3

(c) [8 marks] The following code contains a method called `printNodes()` which is supposed to print all of the nodes in an undirected, connected graph. However, the code contains three errors which will prevent it from working correctly. Find each error and correct it.

```
interface GraphNode {
    public Set<GraphNode> neighbours();
}
```

```
public void printNodes(GraphNode node) {

    Stack<GraphNode> toVisit = new LinkedList<GraphNode>();
    Set<GraphNode> seen = new HashSet<GraphNode>();

    toVisit .push(node);

    while (! toVisit .isEmpty() ) {

        GraphNode next = toVisit.pop();

        System.out.println(next);

        for (GraphNode neighbour: next.neighbours()) {

            if (!seen.contains(neighbour)) {
                seen.add(neighbour);
                toVisit .push(neighbour);
            }
        }
    }
}
```

(d) [3 marks] Why would the method above not work correctly for directed graphs, even after the code has been fixed?

Because in an undirected graph it is possible to have nodes which are not reachable from other nodes, even if the graph is connected.
The method would only print nodes which are reachable from the first node.

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 8 continued)

(e) [10 marks] Write an algorithm which, given a set of nodes representing a directed graph, detects whether or not the graph contains cycles.

```
interface GraphNode {
    public Set<GraphNode> neighbours();
}
```

```
public static boolean hasCycle(Set<GraphNode> graph) {
    for (GraphNode node: graph) {
        Set<GraphNode> visited = new HashSet<GraphNode>();
        Queue<GraphNode> toVisit = new LinkedList<GraphNode>();
        for (GraphNode neighbour: node.neighbours()) {
            toVisit . offer (neighbour);
        }
        while (! toVisit .isEmpty()) {
            GraphNode next = toVisit.poll ();
            if (next.equals(node)) {
                return true;
            }
            else if (! visited .contains(next)) {
                visited .add(next);
                for (GraphNode neighbour: next.neighbours()) {
                    toVisit . offer (neighbour);
                }
            }
        }
    }
    return false;
}
```

```
}
```

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.

(You may detach this page)

Appendices

Possibly useful formulas:

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

Table of base 2 logarithms:

n	1	2	4	8	16	32	64	128	256	512	1024	1,048,576
$\log_2(n)$	0	1	2	3	4	5	6	7	8	9	10	20

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

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>

```
public Iterator <E> iterator();
```

// Can use in the "for each" loop

public interface Comparable <E>

```
public int compareTo(E o);
```

// Can compare this to another E

public interface Comparator <E>

```
public int compare(E o1, E o2);
```

// Can use this to compare two E's

DrawingCanvas class:

```
public void drawLine(int x, int y, int u, int v) // Draws line from (x, y) to (u, v)
public void drawOval(int x, int y, int wd, int ht) // Draws outline of oval
public void drawString(String str, int x, int y) // Prints str at (x, y)
```

```
public interface Collection<E>
    public boolean isEmpty();
    public int size ();
    public boolean add();           // returns false if failed to add item
    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 ();           // throws exception if stack is empty
    public E pop ();           // throws exception 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<K> keySet();
    public Collection<V> values();
    public Set<Map.Entry<K, V>> entrySet();
```

```
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);
```