UNIVERSITY OF SOUTHAMPTON COMP1009W1

SEMESTER 2 EXAMINATION 2011/2012

DATA STRUCTURES AND ALGORITHMS

Duration: 120 mins

You must enter your Student ID and your ISS login ID (as a cross-check) on this page. You must not write your name anywhere on the paper.

Student ID:	
ISS ID:	

Question	Marks
1	
2	
3	
4	
Total	

Answer THREE *questions out of* FOUR.

This examination is worth 85%. The tutorials were worth 15%.

University approved calculators MAY be used.

Each answer must be completely contained within the box under the corresponding question. No credit will be given for answers presented elsewhere.

You are advised to write using a soft pencil so that you may readily correct mistakes with an eraser.

You may use a blue book for scratch—it will be discarded without being looked at.

Number of pages 13

Question 1 The following interface provides a very simple undirected graph interface. The nodes are labelled as integers from 0 to $n-1$.

```
import java.util.Iterator;

public interface Graph {
        public int noNodes();
        public void addEdge(int vertex1, int vertex2);
        public Iterator<Integer> neighbours(int vertex);
}
```

The `addEdge(int vertex1, int vertex2)` allows you to add undirected edges. The `neighbours(int vertex)` returns an iterator of the neighbouring vertices to `vertex`.

(a) Write an implementation of *Graph*. You should use common data structure to store the edges. You should use the iterators of the data structures you are using. Approximate code is sufficient. *(18 marks)*

18

```
public class GraphImpl implements Graph {
        /* Class data */
        List<Set<Integer> > neighbour;


        public GraphImpl(int noNodes) {
                neighbour = new ArrayList<Set<Integer> >(noNodes);
                for (int i = 0; i<noNodes; ++i) {
                        neighbour.add(new TreeSet<Integer>());
                }
        }


        public void addEdge(int vertex1, int vertex2) {
                if (!neighbour.get(vertex1).contains(vertex2))
                        neighbour.get(vertex1).add(vertex2);
                if (!neighbour.get(vertex2).contains(vertex1))
                        neighbour.get(vertex2).add(vertex1);
        }

        public Iterator<Integer> neighbours(int vertex) {
                return neighbour.get(vertex).iterator();
        }

        public int noVertices() {
                return neighbour.size();
        }

}
```

(b) Write code that, given a graph which implements the Graph interface, determines whether the graph is connected. Your code should use a set and a stack. *(15 marks)*

$\overline{15}$

```
public static boolean isConnected(Graph g) {
   Stack<Integer> toCheck = new Stack<Integer>();
   Set<Integer> vertexSet = new TreeSet<Integer>();

   toCheck.push(0);
   vertexSet.add(0);
   while (!toCheck.isEmpty()) {
      int vertex = toCheck.pop();
      Iterator<Integer> neighbours = g.neighbours(vertex);
      while (neighbours.hasNext()) {
         int neigh = neighbours.next();
         if (!vertexSet.contains(neigh)) {
            toCheck.push(neigh);
            vertexSet.add(neigh);
         }

      }
   }
   return vertexSet.size()==g.noVertices();
}
```

End of question 1

Q1: (a) $\frac{}{18}$ (b) $\frac{}{15}$ Total $\frac{}{33}$

Question 2

(a) It takes 0.1 seconds to find an element on average in a (balanced) binary search tree consisting of 100 000 elements. How long will it take on average to search a balanced binary tree with 1 000 000 elements? Explain your answer. *(5 marks)*

Searching a binary tree takes on average $\Theta(\log(n))$**. Thus**
$T(100\,000) \approx c\,\log(100\,000) = 0.1$ **so that** $c \approx 0.1/\log(100\,000)$**.**
$T(100\,000) \approx c\,\log(100\,000) \approx 0.1\log(1\,000\,000)/\log(100\,000) = 0.12$ **seconds.**

$\overline{5}$

(b) Using a 10-way tree, rather than a binary tree, it takes 0.01 seconds to find an element on average in a tree consisting of 100 000 elements. How long will it take on average to search a balanced 10-way tree with 1000 000 elements? Explain your answer. *(3 marks)*

It would take 0.012 seconds. Although the base of the logarithms are different the ratio of the logarithms are the same.

$\overline{3}$

(c) It takes 0.1 seconds on average to sort an array of 100 000 elements using *quick sort*. How long would it take on average to sort an array of 1 000 000 elements? Explain your answer. *(5 marks)*

$\overline{5}$

Quick sort is on average $\Theta(n\,\log(n))$. Thus

$$T(100\,000) = 0.1 \approx c100\,000\,\log(100\,000)$$
$$c \approx \frac{0.1}{100\,000\,\log(100\,000)}$$
$$T(1000\,000) \approx c1\,000\,000\,\log(1\,000\,000) = \frac{0.1 \times 1\,000\,000\,\log(1\,000\,000)}{100\,000\,\log(100\,000} = 1.2 \text{ seconds}$$

(d) It takes 10 seconds on average to sort an array of 100 000 elements using *selection sort*. How long would it take on average to sort an array of 1 000 000 elements? Explain your answer. *(5 marks)*

$\overline{5}$

Quick sort is on average $\Theta(n^2)$. If there are 10 times as many elements then it will take 100 times as long so it will take 1 000 seconds.

(e) A dating agency has developed an algorithm that provides a score between each pair of male and female clients. It has 20 male clients and 20 female clients. It decides to look at all possible ways of pairing up each man with a woman. Their program takes 100 seconds on their 20 male and female clients. They sign up another male and female client. How long would you expect their code to take now? Explain your answer. *(5 marks)*

$\overline{5}$

This is a matching problem (actually a linear assignment problem). There are $n!$ matches since if we lined up the male and females and then consider every permutation of the males we would end up with $n!$ distinct matches. Thus, it would take 21 times longer or 2 100 seconds.

(f) The following code computes the n^{th} Fibonacci number *(10 marks)*

$\overline{}$
10

```
public static int fibonacci(n) {
  if (n<3)
     return 1;
  return fibonacci(n-1) + fibonacci(n-2);
}
```

(i) Write down a recursion relation for the number of calls of `fibonacci` and the base cases.

Recursion Relation: $T(n) = 1 + T(n-1) + T(n-2)$

Base Cases: $T(1) = T(2) = 1$

(ii) Use your recursion relation to fill in the table of the number of calls.

$T(1)$	$T(2)$	$T(3)$	$T(4)$	$T(5)$	$T(6)$	$T(7)$	$T(8)$	$T(9)$	$T(10)$
1	**1**	**3**	**5**	**9**	**15**	**25**	**41**	**67**	**109**

(iii) If it takes 1 second to run *fibonacci(9)* how long does it take to run *fibonacci(10)*
It takes approximately 109/67 = 1.63 seconds.

(iv) Can this code be improved?
Yes! Using recursion to compute Fibonacci number is brain dead. A simple iterative algorithm starting from $n = 1$ would take linear time.

End of question 2

Q2: (a) $\frac{}{5}$ (b) $\frac{}{3}$ (c) $\frac{}{5}$ (d) $\frac{}{5}$ (e) $\frac{}{5}$ (f) $\frac{}{10}$ Total $\frac{}{33}$

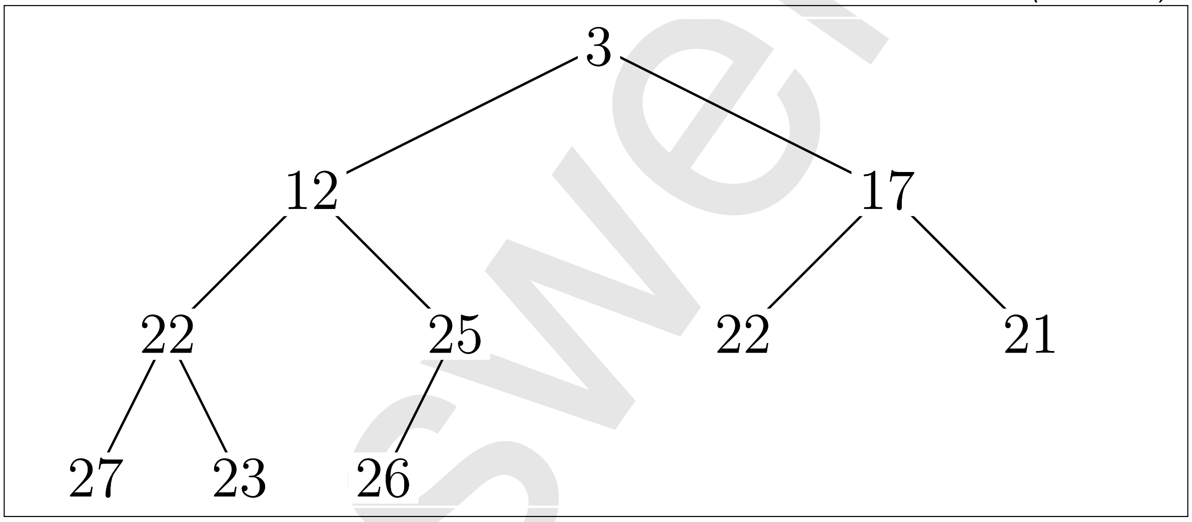
• Do not write in this space •

Question 3

(a) Show how the **heap** shown below is represented as a binary tree.

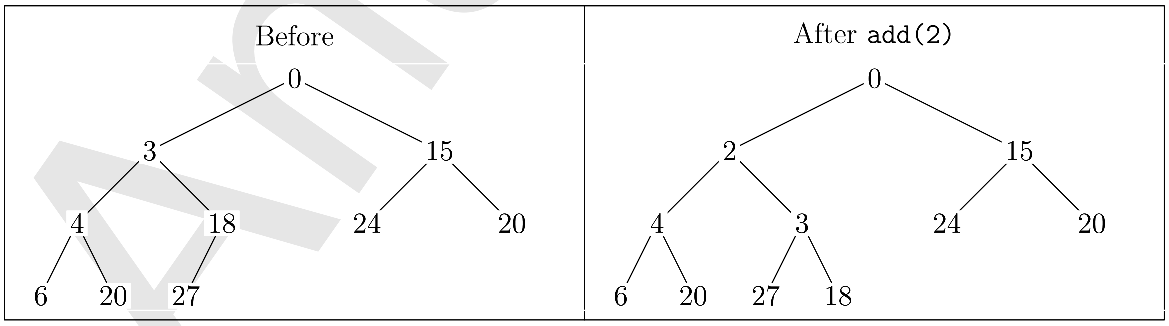
3	12	17	22	25	22	21	27	23	26

(3 marks)



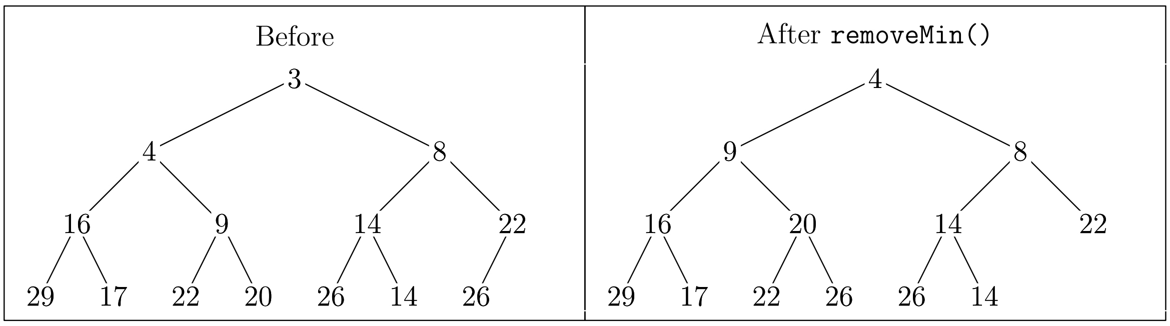
$\overline{}$ 3

(b) Show what happens if you add 2 to the heap shown below. *(3 marks)*



$\overline{}$ 3

(c) Show what happens if you remove the minimum element from the heap shown below. *(3 marks)*



$\overline{}$ 3

(d) Explain how *heap sort* works *(4 marks)*

$\overline{4}$

Heap sort works by putting an array on a heap and then taking it off the heap. The priority is the field being sorted.

(e) What is the time complexity of heap sort. Explain your answer. *(4 marks)*

$\overline{4}$

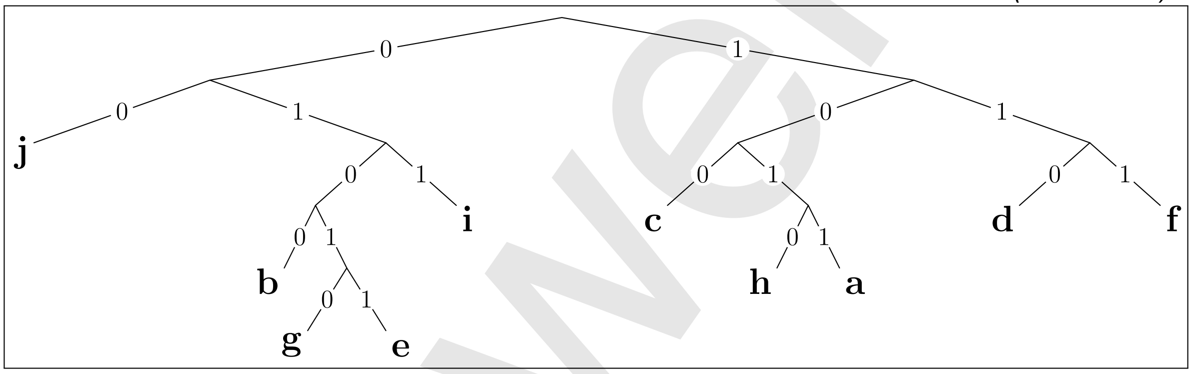
Heap sort is an $O(n\log(n))$ algorithm. This is because adding an element to a heap and taking an element off the heap is $O(\log(n))$. We have to do this n times.

• Do not write in this space •

(f) The following table gives a list of frequency of letters that occur in a document.

Letter	a	b	c	d	e	f	g	h	i	j
Frequency	5	3	7	9	2	11	1	4	6	12

Construct a Huffman tree for this set of letters. *(10 marks)*



$\overline{10}$

(g) Use your tree to encode the word "deface". *(6 marks)*

110010111111011110001011
(Note that the Huffman tree is not unique so this will differ.)

$\overline{6}$

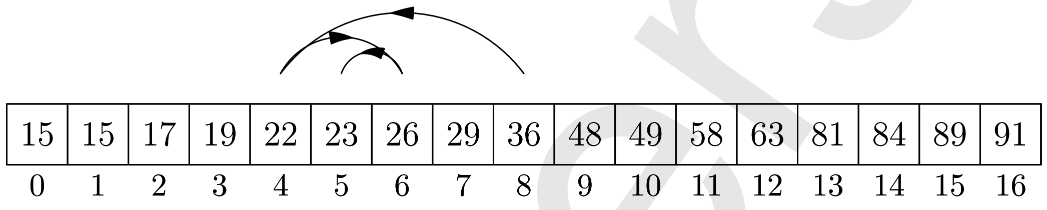
End of question 3

Q3: (a) $\frac{}{3}$ (b) $\frac{}{3}$ (c) $\frac{}{3}$ (d) $\frac{}{4}$ (e) $\frac{}{4}$ (f) $\frac{}{10}$ (g) $\frac{}{6}$ Total $\frac{}{33}$

• Do not write in this space •

Question 4 This question covers many different topics.

(a) Show what comparisons would be done by *binary search* to find the number 23 in the sorted array shown below. *(3 marks)*



15	15	17	19	22	23	26	29	36	48	49	58	63	81	84	89	91
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

$\overline{\ }$
3

(b) Describe the queue abstract data type. *(3 marks)*

It is a model of a last in last out memory with the methods `enqueue(item)` which puts the `item` in the queue, `dequeue()` which returns the longest existing item on the queue, and `isEmpty()` which returns true if the queue is empty and false otherwise.

$\overline{\ }$
3

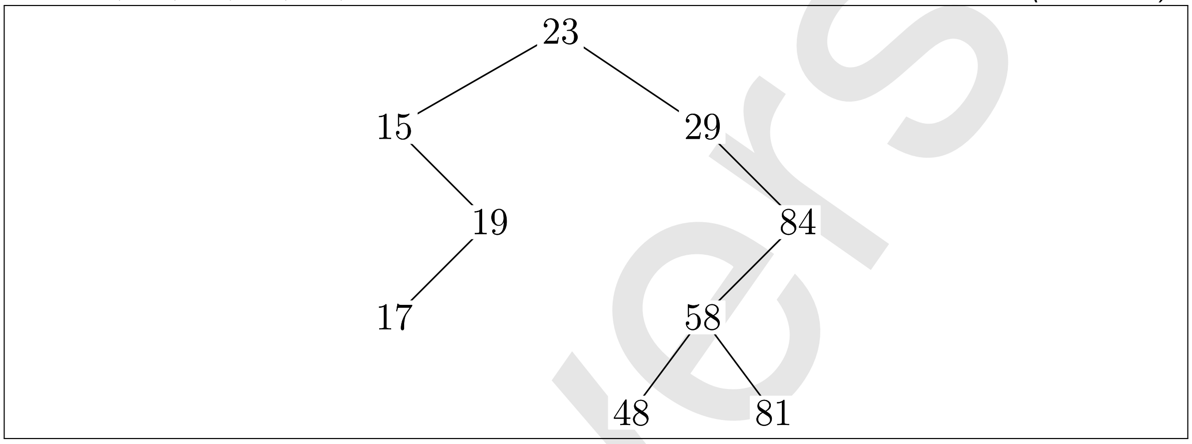
(c) Explain the purpose of the `Comparator` class. *(3 marks)*

The comparator lets the user either replace the standard method for a comparable object or to compare objects that are not comparable (i.e. don't implement the Comparable interface).

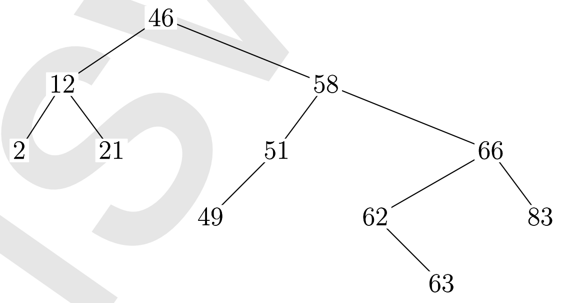
$\overline{\ }$
3

(d) Show the binary search tree that is obtained when we add 23, 29, 84, 15, 58, 19, 81, 17, 48, and 15. *(3 marks)*

$\overline{}$
3

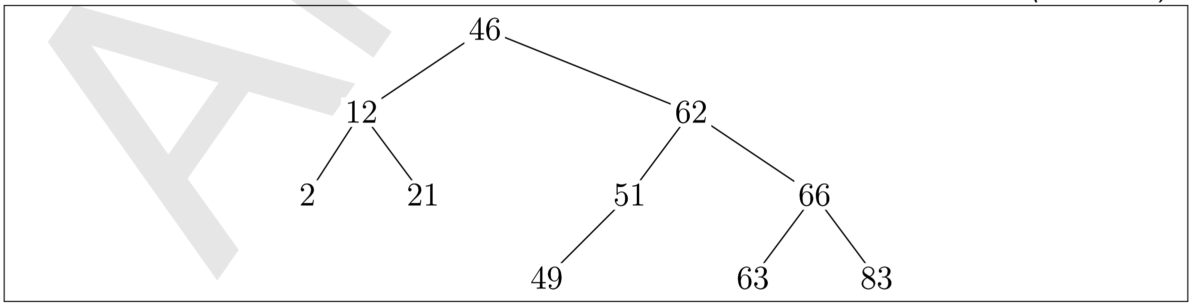


(e) Show what happens when you delete 58 from the binary search tree shown below



(6 marks)

$\overline{}$
6



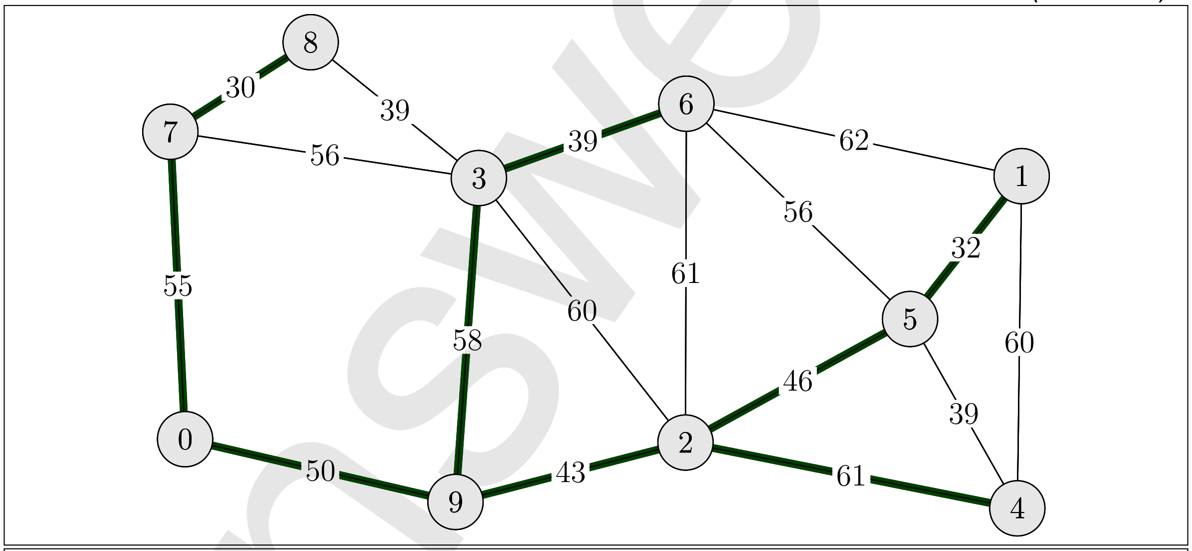
(f) Why is balancing a binary search tree so important. *(2 marks)*

Without balancing the shape of a binary search tree depends on the order of the inputs. If the inputs are entered in order (or reverse order) or nearly so then the resulting tree would be extremely poorly balanced and look up will take linear time. Such inputs are not too uncommon.

$\overline{}$
2

(g) Show the tree of edges found by Dijkstra's algorithm from node 0 and write down the order of the edges and the distance of the node to the source node. *(7 marks)*

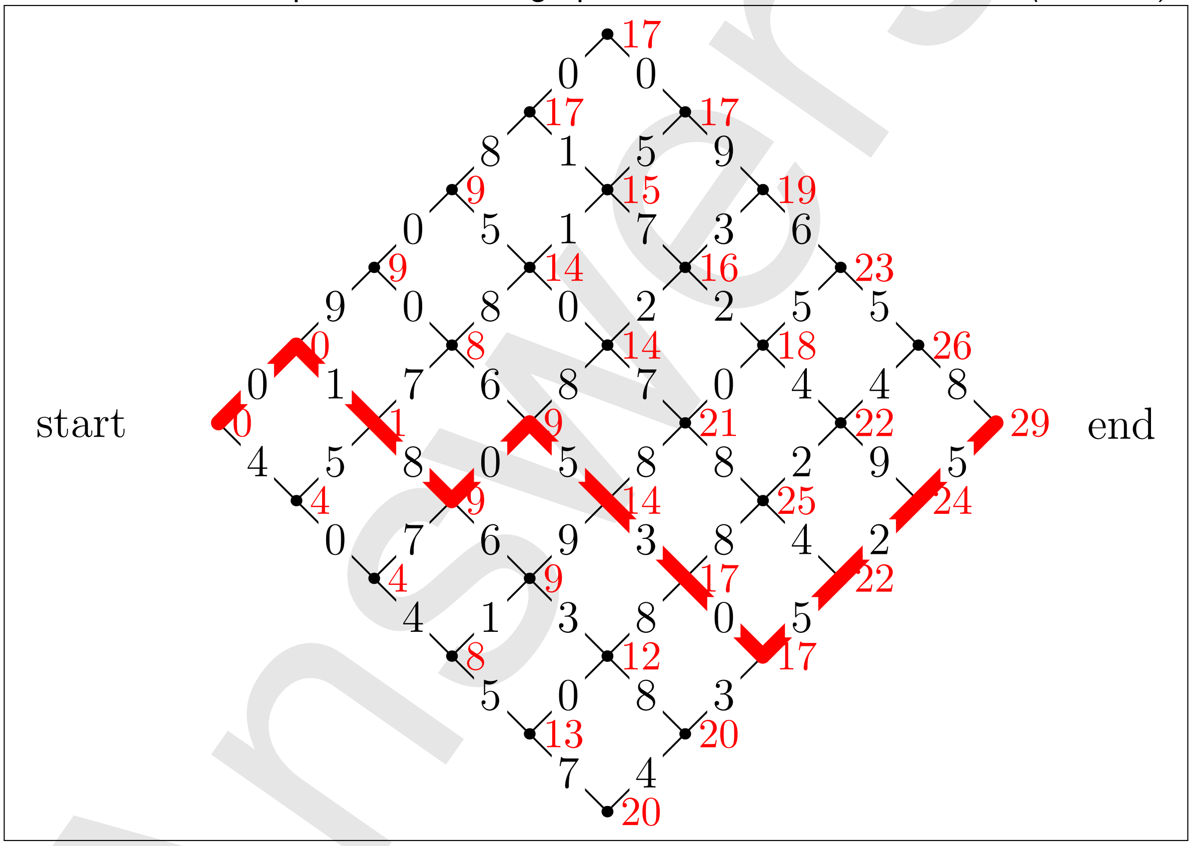
$\overline{}$
7



1. (0,9) 50	2. (0,7) 55	3. (7,8) 85
4. (2,9) 93	5. (3,9) 108	6. (2,5) 139
7. (3,6) 147	8. (2,4) 154	9. (1,5) 171

(h) Use the dynamic programming forward algorithm to compute the minimum cost of each path from the left most node to each other node where the cost of moving along an edge is equal to the number shown. An edge can only be traversed from left to right. Use the backwards algorithm to find the minimum cost path across the graph. *(6 marks)*

$\overline{}$ 6



End of question 4

Q4:	(a) $\frac{}{3}$	(b) $\frac{}{3}$	(c) $\frac{}{3}$	(d) $\frac{}{3}$	(e) $\frac{}{6}$	(f) $\frac{}{2}$	(g) $\frac{}{7}$	(h) $\frac{}{6}$	Total $\frac{}{33}$

END OF PAPER