UNIVERSITY OF SOUTHAMPTON COMP1009W1

SEMESTER 2 EXAMINATION 2007/2008

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 12

Question 1

(a) Give six container classes that are part of the Java collection and describe briefly what they do and how they are implemented *(12 marks)*

(Question tests software engineering aspects of data structures)

(i) `ArrayList<T>` **a list implemented using an array**

(ii) `LinkedList<T>` **a list implemented using a linked list**

(iii) `TreeSet<T>` **a set (collection where the order of insertion doesn't matter) implemented using a binary (red-black) tree**

(iv) `HashSet<T>` **a set implemented as a hash table**

(v) `TreeMap<K,V>` **a map (a collection of key-values pairs) implemented using a binary tree**

(vi) `HashMap<K,V>` **a map implemented using a hash table**

(b) Describe the advantages of using generics rather than building containers for the type `Object` *(4 marks)*

(Test basic understanding of generics)

Generic containers allow type checking which lets many errors be picked up at compiler time rather than at run time. It also makes it unnecessary to cast when extracting objects from a container.

(c) Explain how the iterator pattern works in Java collections. In particular describe the methods used in the iterator interface *(6 marks)*

The iterator interface is implemented by the Java collection classes. It provides methods for iterating through each member of the container. The iterator interface, `iterator<T>`**, has three methods**

(i) `boolean hasNext()` **which returns true if there is another member of the container which has not been visited and false otherwise**

(ii) `T next()` **returns the next element. If there is no next element it throws an exception**

(iii) `void remove()` **removes the last element visited by** `next()`**. It throws an exception if** `next()` **has not been called.**

(d) Explain what the `Comparable<T>` interface is and how it is used to make a class have comparable instances? *(5 marks)*

The `Comparable<T>` interface contains a single method

- `int compareTo(T o)`.

A class that allows comparisons of instances will implement this interface. `compareTo(T o)` returns either a negative integer, zero or a positive integer depending on whether the object is "less than", "equal to", or "greater than" the argument. This defines the natural ordering of the objects in a class. (It should be antisymmetric, transitive and usually compatible with equality).

(e) Explain why it is necessary also to have a `Comparator<T>` interface and how it is used? *(6 marks)*

The `Comparator<T>` interface is necessary when you want to sort objects that do not have a natural ordering defined or when you want to change the ordering (e.g. reverse the natural ordering). It is again a one method interface

- `int compare(T o1, T o2)`

which is analogous to `int compareTo(T o)`, although it is not a method of the object. The `Comparator<T>` class is passed to a method or class requiring ordering, for example, to sort or TreeSet.

End of question 1

Q1: (a) $\frac{}{12}$	(b) $\frac{}{4}$	(c) $\frac{}{6}$	(d) $\frac{}{5}$	(e) $\frac{}{6}$	Total $\frac{}{33}$

• Do not write in this space •

Question 2 Merge sort has the form

```
MergeSort(A[1:n]) {
   if (n>1) {
      B = A[1:n/2]
      C = A[n/2+1:n]
      MergeSort(B)
      MergeSort(C)
      Merge(B,C,A)
   }
}
```

The number of comparison operations to merge two arrays of length $n/2$ is n.

(a) Let $C(n)$ be the number of comparison operations. Write down a recurrence relation for $C(n)$ valid if $n = 2^m$ *(4 marks)*

$$C(n) = 2\,C(n/2) + n$$

(b) Write down the boundary condition $C(1)$ and use the recurrence relation to compute $C(2)$, $C(4)$, and $C(8)$ *(4 marks)*

$$C(1) = 0$$

$$C(2) = 2$$

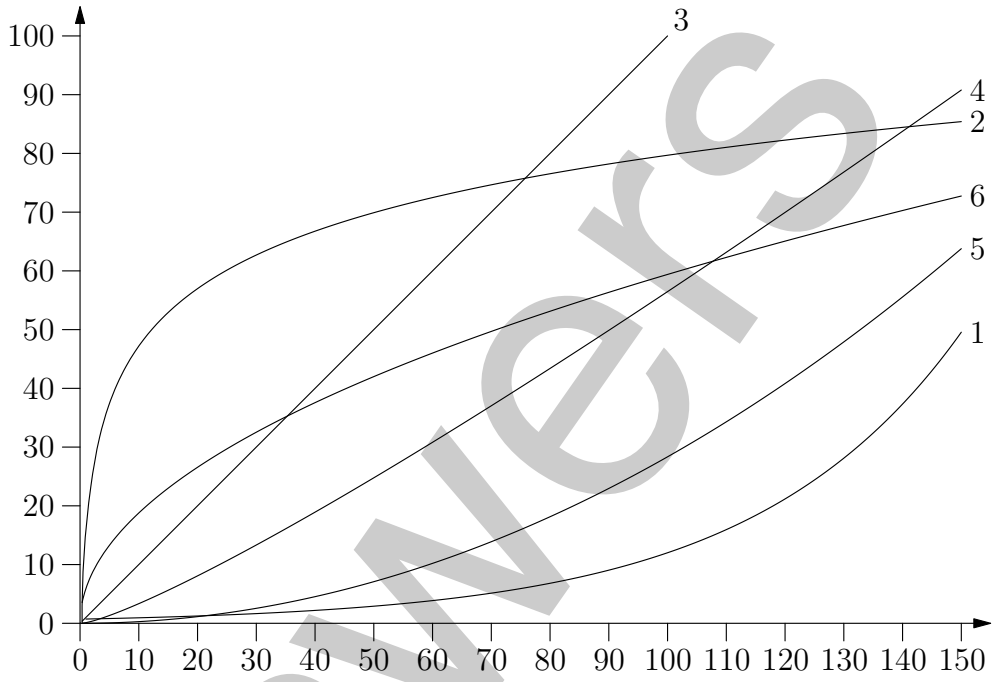
$$C(4) = 2 \times 2 + 4 = 8$$

$$C(8) = 2 \times 8 + 8 = 24$$

(c) Demonstrate, for $n = 2^m$, that $f(n) = n \log_2(n)$ satisfies the recurrence relation in part (a) *(6 marks)*

$$\begin{aligned} C(n) &= 2\,C(n/2) + n \\ &= 2\,\frac{n}{2}\log_2\left(\frac{n}{2}\right) + n \\ &= n\left(\log_2(n) - \log_2(2)\right) + n \\ &= n\left(\log_2(n) - 1\right) + n = n\,\log_2(n) \end{aligned}$$

(d) The graph below shows the time complexity for the following algorithms (a) $\Theta(e^{a\,n})$, (b) $\Theta(n^2)$, (c) $\Theta(n\,\log(n))$, (d) $\Theta(n)$, (e) $\Theta(\sqrt{n})$, and (f) $\Theta(\log(n))$. Match the time complexity classes with the curves on the graph.



(6 marks)

1.	(a) $\Theta(e^{a\,n})$	**2.**	(f) $\Theta(\log(n))$
3.	(d) $\Theta(n)$	**4.**	(c) $\Theta(n\,\log(n))$
5.	(b) $\Theta(n^2)$	**6.**	(e) $\Theta(\sqrt{n})$

(e) Which of the following statements are true? Give reasons why (marks will only be awarded if correct reasons are given) *(8 marks)*

(i) All $\Theta(n\,\log(n))$ algorithms are faster than all $\Theta(n^2)$ algorithms

False, this is only true when n is large

(ii) All $O(n^2)$ algorithms run slower than all $O(n)$ asymptotically

False, $O(n^2)$ is an upper bound on the time complexity and includes algorithms of $\Theta(1)$ which can be faster than some $O(n)$ algorithms

(iii) A $\Theta(n!)$ algorithm runs slower than any exponential algorithm in the limit of large n

True, $n!$ grows faster with n than any exponential

TURN OVER

(iv) An $O(n/\log(n))$ algorithm will run faster than a $\Omega(n\ \log(n))$ algorithm for sufficiently large n

---

True, the worst case $O(n/\log(n))$ algorithm runs in $n/\log(n)$ which is faster than the best case $\Omega(n\ \log(n))$ algorithm which runs in $n\ \log(n)$ (i.e. $\log^2(n)$ times longer).

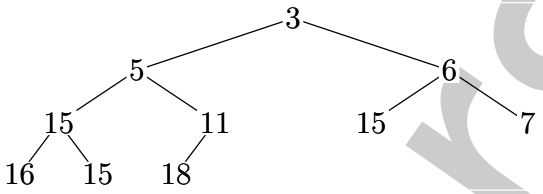
(f) Why is it widely believed that $NP \neq P$? *(5 marks)*

There is a large class of problems, the NP-complete problems with the property that if any could be solved in polynomial time then all problems in NP can be solved in polynomial time, thus NP would equal P. But, so far no one has found a polynomial algorithm for a single NP-complete problems.

End of question 2

Q2: (a) $\frac{}{4}$	(b) $\frac{}{4}$	(c) $\frac{}{6}$	(d) $\frac{}{6}$	(e) $\frac{}{8}$	(f) $\frac{}{5}$	Total $\frac{}{33}$

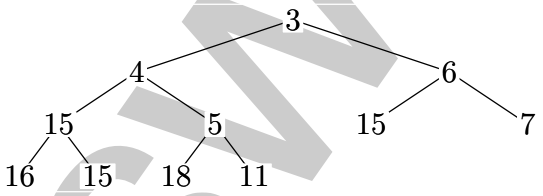
Question 3 Consider the **heap** represented as a binary tree



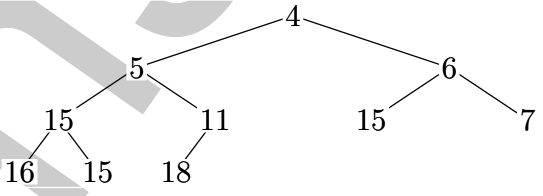
(a) Show how the heap would be stored in the compute memory *(3 marks)*

3	5	6	15	11	15	7	16	15	18

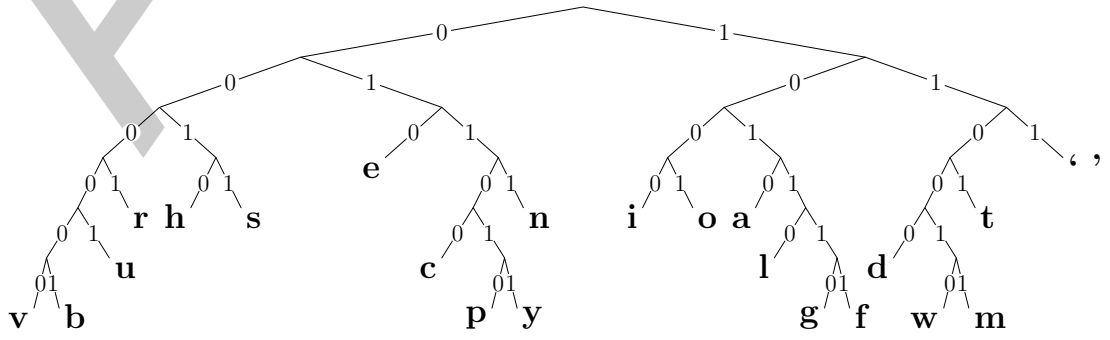
(b) Draw a binary tree representing the heap after we add 4 to it *(5 marks)*



(c) Draw a binary tree representing the heap above when we run `removeMin()` *(5 marks)*



(d) Given the Hoffman tree



decode 01101010110000010111101110010 *(6 marks)*

---

plunge

TURN OVER

(e) Give a high level description (ignoring implementation details) of how a Huffman tree is constructed for the set of English letters. *(8 marks)*

A Huffman tree starts by computing the frequency of occurrence of English letters. From this it constructs a tree by combining pairs of trees. Initially we start with one tree for each letter. Each tree has assigned to it a value equal to the sum of occurrences of the letters that occur in the tree. The two trees containing the most infrequently occurring letters are combined iteratively until only one tree is left which is the Huffman tree.

(f) Describe how a heap is used to construct the Huffman tree *(4 marks)*

The heap is used to order the trees. The heap takes trees ordered by the sum of occurrences of letters in the tree. At each iteration the two trees which occur least often are taken off the tree. They are then joined together and added back to the heap. This is repeated until there is only one tree remaining

(g) Describe how a heap is used in Heap Sort *(2 marks)*

In heap sort an array is sorted by putting every every element of the array onto the heap. When this is done elements are taken off the heap and put back into the array, but now they will be in sorted order.

End of question 3

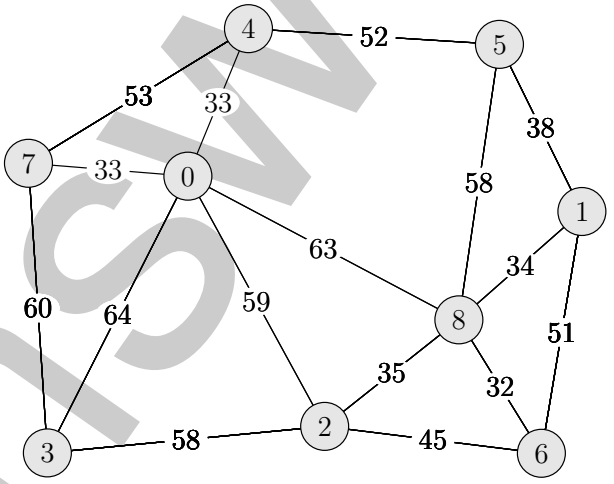
Q3: (a) $\frac{}{3}$	(b) $\frac{}{5}$	(c) $\frac{}{5}$	(d) $\frac{}{6}$	(e) $\frac{}{8}$	(f) $\frac{}{4}$	(g) $\frac{}{2}$	Total $\frac{}{33}$

Question 4

(a) Describe Kruskal's algorithm for finding minimal spanning trees *(10 marks)*

Kruskal's algorithm works by adding the shortest remaining edge in the graph to the minimal spanning tree provided it does not form a loop with edges previously added. The algorithm ends when $n-1$ edges have been added. The edges are sorted, usually by a heap. To check that the edges don't form a loop a union find algorithm is used.

(b) For the following graph, show in what order Kruskal's algorithm would add edges to the minimum spanning tree



(8 marks)

1.	$(6,8)$	2.	$(0,4)$ or $(0,7)$
3.	$(0,7)$ or $(0,4)$	4.	$(1,8)$
5.	$(2,8)$	6.	$(1,5)$
7.	$(4,5)$	8.	$(2,3)$

The disjoint set class is described by the following program

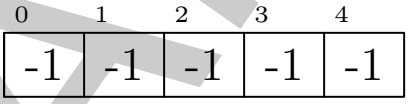
```
public class DisjSets
{
    private int[] s;

    public DisjSets(int numElements) {
        s = new int[numElements];
```

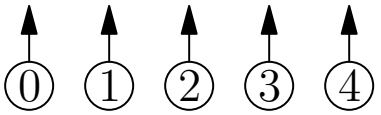
```
        for(int i=0; i<s.length; i++)
            s[i] = -1;
    }

    public void union(int root1, int root2) {
        if (s[root2]<s[root1]) {
            s[root1] = root2;
        } else {
            if (s[root1]==s[root2])
                s[root1]--;
            s[root2] = root1;
        }
    }

    public int find(int x) {
        if (s[x]<0)
            return x;
        else
            return s[x] = find(s[x]);
    }
}
```

We assume that we have created an instance of the disjoint sets class

```
DisjSets disjset = new DisjSets(5);
```

Below we show the initial settings of the array `s` and a graphical representation of the forest (set of trees) representing the array.

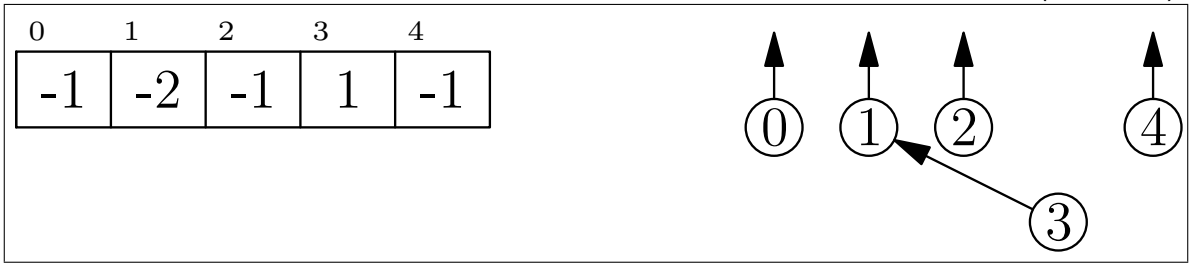




(c) Show the state of array and the forest after performing the following operation

```
disjset.union(disjset.find(1), disjset.find(3));
```

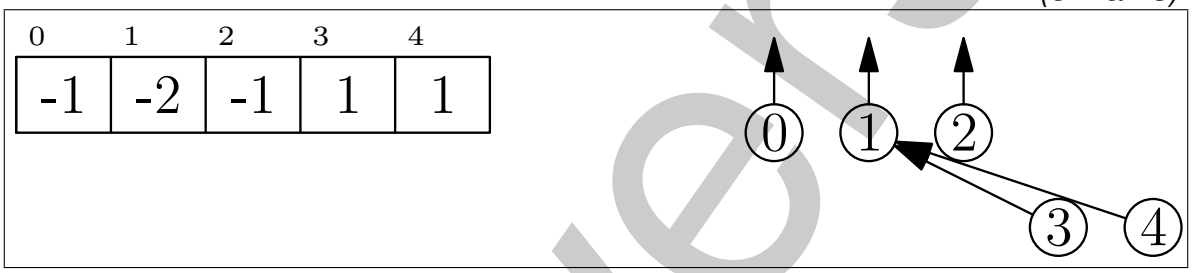
(3 marks)



(d) Show the state of array and the forest after performing the following operation

```
disjset.union(disjset.find(4), disjset.find(3));
```

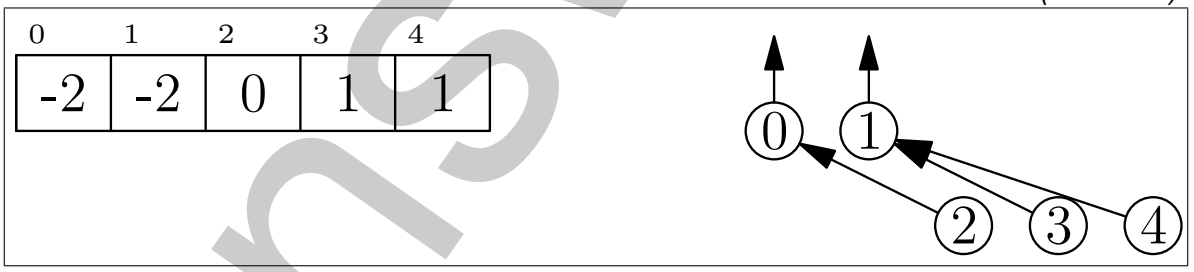
(3 marks)



(e) Show the state of array and the forest after performing the following operation

```
disjset.union(disjset.find(0), disjset.find(2));
```

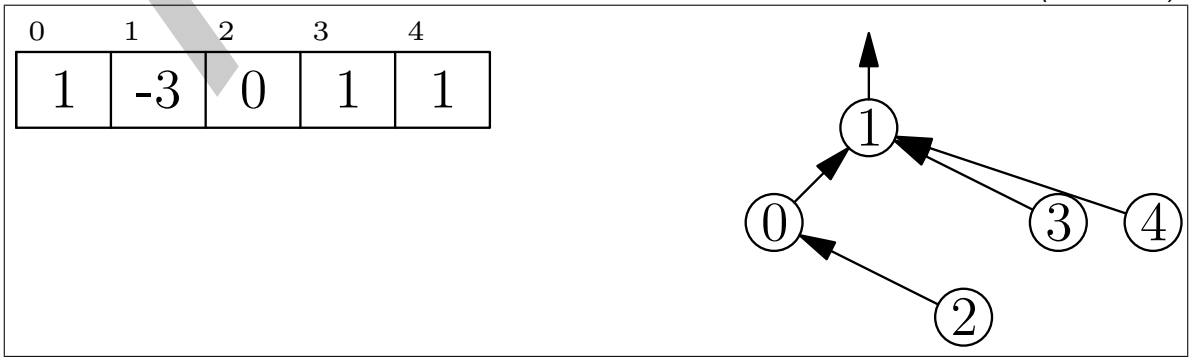
(3 marks)



(f) Show the state of array and the forest after performing the following operation

```
disjset.union(disjset.find(4), disjset.find(2));
```

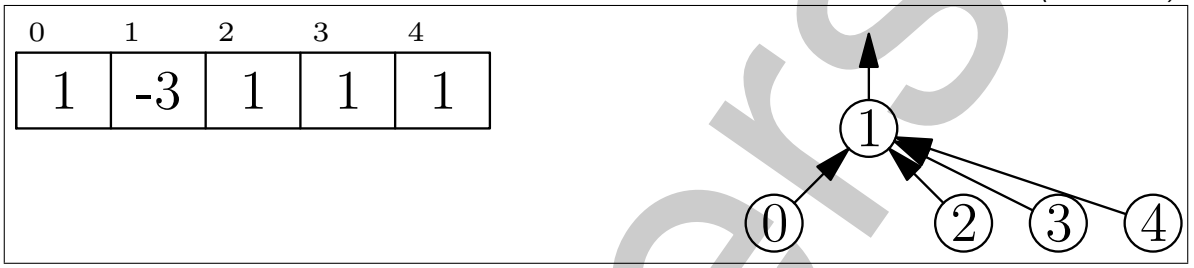
(3 marks)



(g) Show the state of array and the forest after performing the following operation

```
disjset.find(2);
```

(3 marks)



End of question 4

Q4: (a) $\frac{}{10}$	(b) $\frac{}{8}$	(c) $\frac{}{3}$	(d) $\frac{}{3}$	(e) $\frac{}{3}$	(f) $\frac{}{3}$	(g) $\frac{}{3}$	Total $\frac{}{33}$

END OF PAPER