UNIVERSITY OF SOUTHAMPTON COMP1009W1

SEMESTER 2 EXAMINATION 2009/2010

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 five container classes that are part of the Java collection and describe briefly what they do and how they are implemented. *(10 marks)*

(Question tests software engineering aspects of data structures)

Any five of

(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**

The algorithm for bubble sort (not covered in the course) to sort an array $\boldsymbol{a} = (a_0, a_1, \ldots, a_{n-1})$ is given by

```
BUBBLESORT(a)
  for i←0 to n-2
    for j←0 to n-2-i
      if a_{j+1} < a_j
         swap a_j and a_{j+1}
       endif
     endfor
  endfor
```

(b) Write a Java method to performing bubble sort on an array of integers. *(7 marks)*

Students haven't been shown details of bubble sort although they had a very quick description of it. Simple exercise showing understanding of pseudo-code.

```
public static void sort(int[] a) {
  for (int i=0; i<a.length-1; i++) {
    for (int j=0; j<a.length-i-1; j++)
```

```
        if (a[j+1]<a[j]) {
          int swap = a[j];
          a[j] = a[j+1];
          a[j+1] = swap;
          }
        }
      }
    }
```

(c) Show how you could make your sort program generic so that it will sort an array of comparable objects. *(5 marks)*

Test knowledge of generics and Comparable.

```
public static <T extends Comparable<? super T> > void sor
  for (int i=0; i<a.length-1; i++) {
    for (int j=0; j<a.length-i-1; j++)
    if (a[j+1].compareTo(a[j])<0) {
      T swap = a[j];
      a[j] = a[j+1];
      a[j+1] = swap;
      }
    }
  }
}
```

(d) What is the time complexity of bubble sort? Explain your answer. *(3 marks)*

Test simple loop counting.

The time complexity is $O(n^2)$ since there are two nested "for loops" which both are $O(n)$ long.

(e) Java uses quick sort for sorting arrays of primitive types and merge sort to sort array lists. Explain why Java makes this choice. In your answer you should make reference to the time complexity of the algorithms and whether the algorithms are stable and in-place. *(8 marks)*

Test understanding of the strength and weaknesses of different algorithms.

TURN OVER

Quick sort is the fastest general purpose sort on average. It beats merge sort in part because it is in-place. It is therefore a natural choice for sorting primitive types. However it is not stable. This makes no different for primitive types but might be important for sorting objects. Java therefore uses merge sort for array lists as this is stable (just allowing multiple sorts on different attributes). Merge sort is slower than quick sort in part because it is not in-place. Both sorts have an average time complexity of $\Theta(n \log(n))$, although quick sort has a worst case time complexity of $\Theta(n^2)$.

End of question 1

Q1: (a) $\frac{}{10}$	(b) $\frac{}{7}$	(c) $\frac{}{5}$	(d) $\frac{}{3}$	(e) $\frac{}{8}$	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 $T(n)$ be the number of comparison operations. Write down a recurrence relation for $T(n)$ valid if $n = 2^m$ *(4 marks)*

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

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

$$T(1) = 0$$

$$T(2) = 2 \times 0 + 2 = 2$$

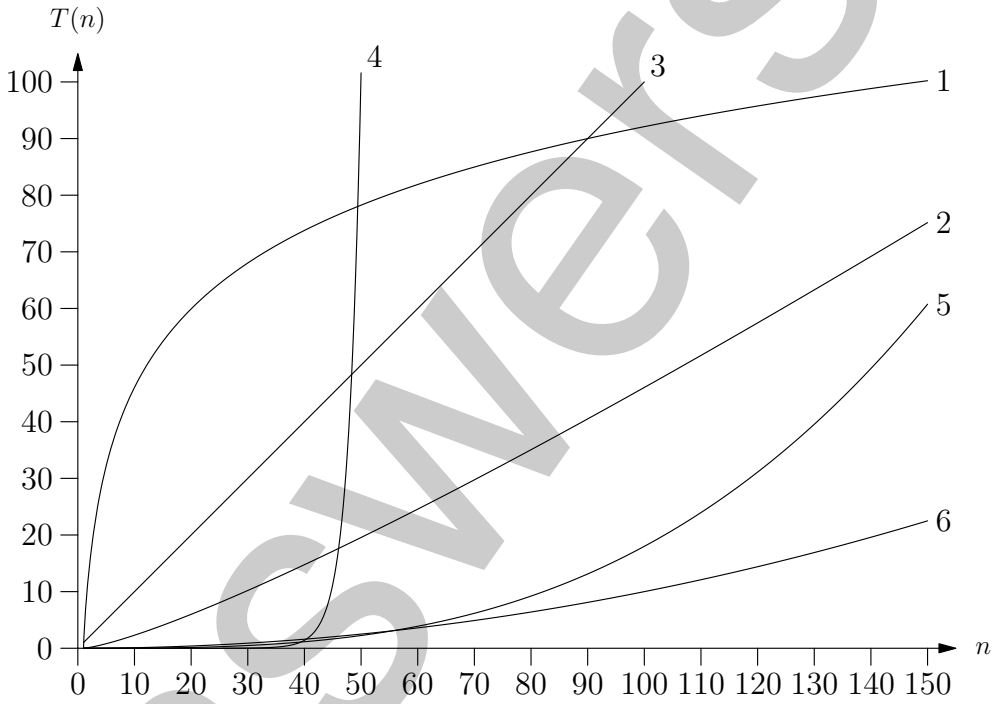
$$T(4) = 2 \times 2 + 4 = 8$$

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

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

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

(d) The graph below shows the time complexity for the following algorithms (a) $\Theta((n/a)!)$, (b) $\Theta(n^2)$, (c) $\Theta(n\ \log(n))$, (d) $\Theta(n)$, (e) $\Theta(n^3)$, and (f) $\Theta(\log(n))$. Match the time complexity classes with the curves on the graph.



(6 marks)

1.	(f) $\Theta(\log(n))$	**2.**	(c) $\Theta(n\ \log(n))$
3.	(d) $\Theta(n)$	**4.**	(a) $\Theta((n/a)!)$
5.	(e) $\Theta(n^3)$	**6.**	(b) $\Theta(n^2)$

(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 $\Theta(n^2)$ algorithms are faster than all $\Theta(n^3)$ algorithms

False, this is only true when n is large

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

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

(iii) All $O(n^3)$ algorithms run slower than all $O(n^2)$ asymptotically

False, $O(n^3)$ is an upper bound on the time complexity and includes algorithms of $\Theta(n)$ which can be faster than some $O(n^2)$ algorithms

(iv) 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

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

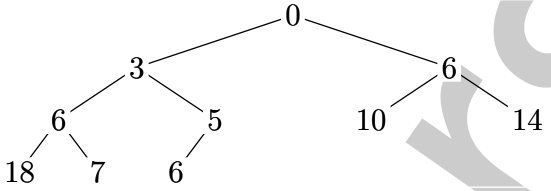
There is a large class of problems, the NP-complete problems with the property that if any could be solved in polynomial time then all problems in NP can be solved in polynomial time, thus NP would equal P. But, so far no one has found a polynomial algorithm for a single NP-complete problems.

End of question 2

Q2: (a) ___	(b) ___	(c) ___	(d) ___	(e) ___	(f) ___	Total ___
/4	/4	/6	/6	/8	/5	/33

• Do not write in this space •

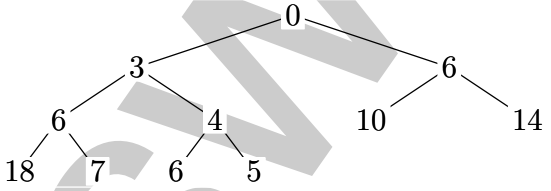
Question 3 Consider the **heap** represented as a binary tree



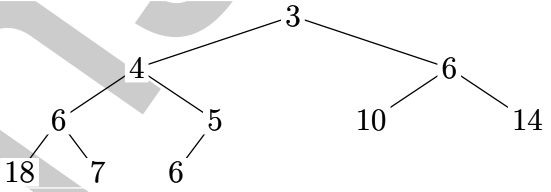
(a) Show how the heap would be stored in the computer memory *(3 marks)*

0	3	6	6	5	10	14	18	7	6

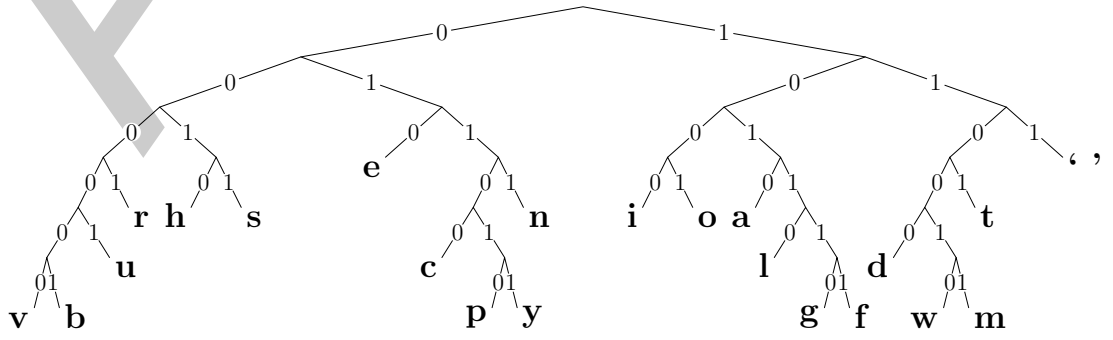
(b) Draw a binary tree representing the heap after we add 4 to it *(5 marks)*



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



(d) Given the Huffman tree



decode 00111011000001110001011100 10 *(6 marks)*

---

sludge

(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 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 $\frac{}{33}$

• Do not write in this space •

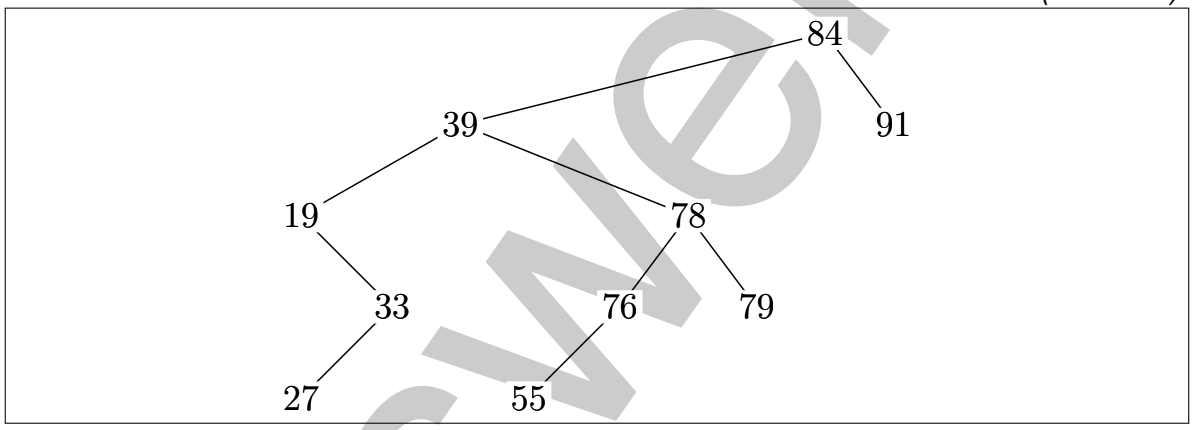
TURN OVER

Question 4

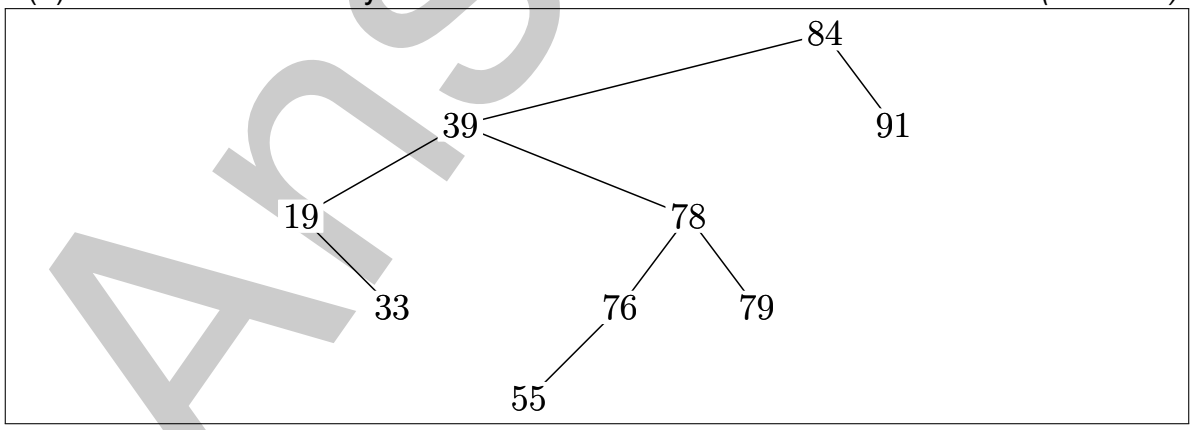
(a) Draw the binary search tree which results from inserting the following list of numbers into the tree

$$84, 39, 78, 79, 91, 19, 33, 76, 27, 55.$$

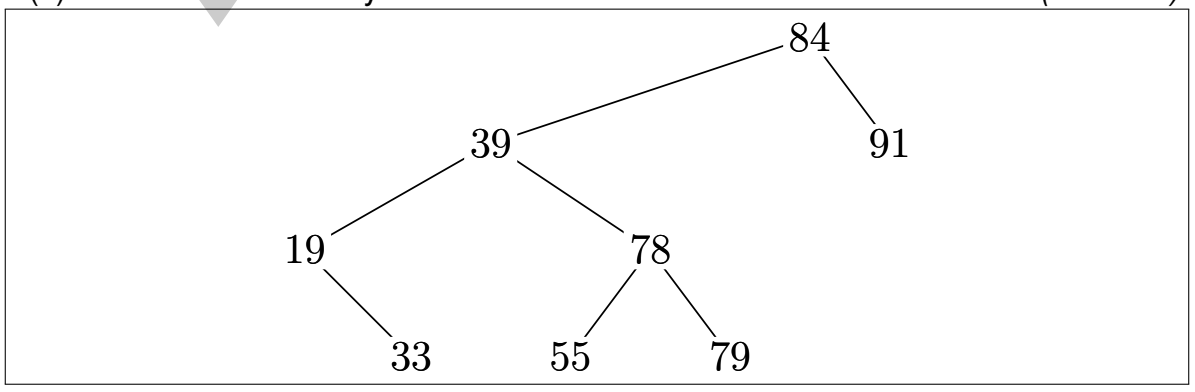
(5 marks)



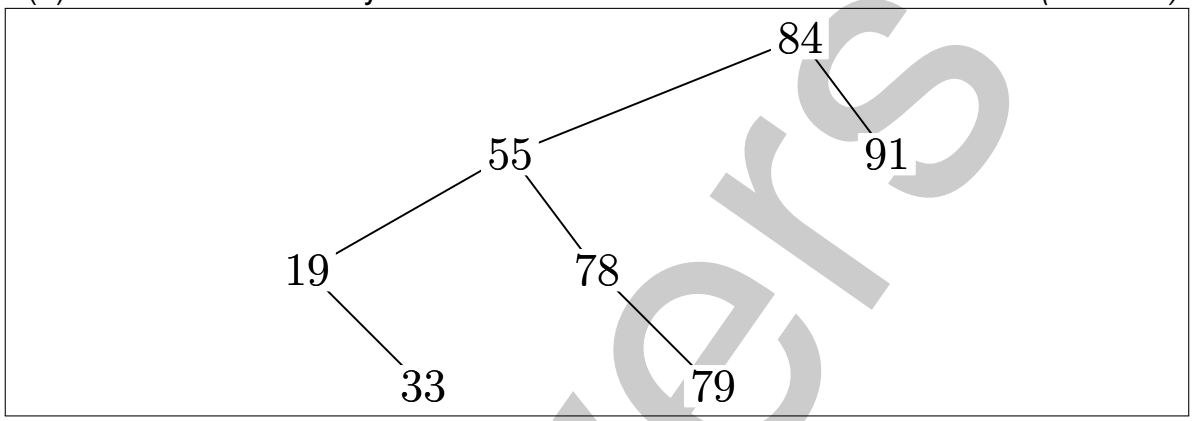
(b) Draw the tree after you delete 27. *(3 marks)*



(c) Draw the tree after you delete 76. *(3 marks)*



(d) Draw the tree after you delete 39. *(4 marks)*



(e) Derive the typical and worst case time complexities for insertion. Briefly explain the strategies used to reduce the worst case time complexity. *(10 marks)*

This tests an understanding of the theoretical aspects underlying trees.

The time complexity for insertion depends on the depth of the search tree, which in turn depends on the number of entries n. In the typical case the tree is approximately balanced and the depth of any branch is $O(\log(n))$, which is therefore the typical case time complexity.

In the worst case the tree is completely unbalanced. In which case the tree turns into a linked list. This would happen for example if the entries were inserted in order. In this case each insertion would take $O(n)$ operations.

To avoid poorly balanced trees, the trees are rebalanced using rotation operations. The usual strategy is to maintain some measure of how unbalanced the tree becomes, when the tree is too unbalance a rotation takes place to re-balance the tree. Two common mechanisms to measure how poorly balanced a tree is are AVL trees and red-black trees. Both strategies ensure that the depth of the tree is $O(\log(n))$.

(f) Explain why binary search trees are commonly used to represent sets. What alternative data structure is also used for this purpose? How do these data structures compare? *(8 marks)*

The requirement of a set is that it has fast search, insertion and deletion. A binary search tree performs all these operations in $O(\log(n))$ time.

The other commonly used data structure for representing sets are hash tables. This has constant time search, insertion and deletion, although a hash function evaluation is required and there is a hidden cost of collision avoidance when the table becomes heavily loaded.

TURN OVER

One feature of binary search trees which may be an advantage is that the iterator visits each element in order.

End of question 4

Q4: (a) $\frac{}{5}$ (b) $\frac{}{3}$ (c) $\frac{}{3}$ (d) $\frac{}{4}$ (e) $\frac{}{10}$ (f) $\frac{}{8}$ Total $\frac{}{33}$

END OF PAPER