UNIVERSITY OF SOUTHAMPTON COMP1009W1

SEMESTER 2 EXAMINATION 2008/2009

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 14

Question 1 Below we show part of a binary search tree class for storing integers

```
public class BinarySearchTree {
   Node root;
   int number_of_elements;

   private static class Node {
      int element;
      Node left;
      Node right;
      Node parent;
   }

}
```

(a) Write a recursive method `sum()` which sums all the elements stored in the tree. Approximately correct code is sufficient. *(5 marks)*

(Test understanding of binary tree, recursion and coding.)

```
public int sum() {
  return sum_recursive(root);
}

private int sum_recursive(Node current) {
  if (current!=null) {
     return current.element + sum_recursive(current.left)
                            + sum_recursive(current.right);
  } else { return 0; }
}
```

(b) Show how you would make the BinarySearchTree into a generic class. *(5 marks)*

(Test understanding of making data structures generic.)

```
public class BinarySearchTree<E extends Comparable<? super E>> {
   Node<E> root;
   int number_of_elements;

   private static class Node<T> {
```

```
        T element;
        Node<T> left;
        Node<T> right;
        Node<T> parent;
    }

}
```

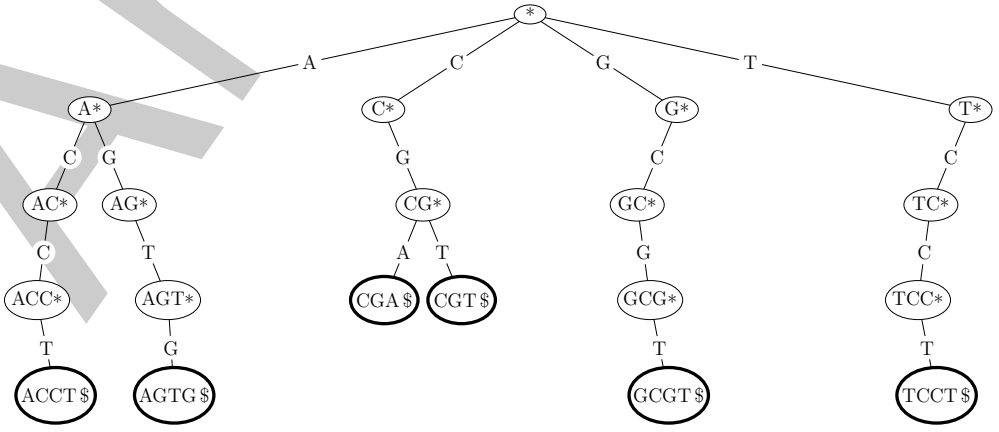
(The exact signature *T extends Comparable<? super T>* ***is not required for full marks.)***

• Do not write in this space •

(c) Write a method `getNode(E obj)` which returns the node with element `obj` if it exists otherwise it returns `null`. You may assume `E` extends the Comparable interface. *(8 marks)*

```
public Node<E> getNode(E obj)
{
    Node<E> e = root;
    while (e != null) {
        int comp = obj.compareTo(e.element);
        if (comp == 0)
            return e;
        else if (comp < 0)
            e = e.left;
        else
            e = e.right;
    }
    return null;
}
```

(d) A digital tree or trie is a multi-way tree which can also be used to implement a set. An illustration of a trie that could be used for storing DNA sequences is shown below. This is a 4-way trie storing the sequences "ACCT$", "AGTG$", "CGA$", "CGT$", "GCGT$" and "TCCT$". Describe how the tree can be modified to be more efficient (turned into a Patricia trie).



(4 marks)

(Tests knowledge of more advanced data structure.)

Where a non-terminal node has only one child, it can be amalgamated with it child so that the edges represent more than one symbol.

• Do not write in this space •

(e) Give the retrieval time of an element from a `TreeSet`, a `HashSet` and a Trie. For full marks give a brief explanations of your answer. *(6 marks)*

(i) The `TreeSet` has a retrieval time of $O(\log(n))$ where n is the number of elements in the set. This is because `TreeSet` use binary trees. The time it takes to reach a leaf is the depth of the tree which is $O(\log(n))$.

(ii) The `HashSet` has a retrieval time of $O(1)$. It uses a hash table which is indexed by a hash function.

(iii) The Trie will have a retrieval time of $O(\log_4(m))$ where m is the length of the string. Being a four way tree it has the potential of taking half the time of the `TreeSet`.

• Do not write in this space •

Pseudo code for a possible hashing function for DNA sequences is shown below

```
hash = 17;
foreach base b in a sequence
  hash = 31*hash + integer(b)
return hash
```

where *integer(b)* turns the bases, A, C, G, T, into integers 0, 1, 2, 3.

(f) Explain, why a `HashSet` may not be any faster than the `Trie` at inserting or retrieving data. *(5 marks)*

Such a hash function would take as many operations as there are symbols in the string. But this is the same number of operations as used by a Trie to insert and retrieve data.

End of question 1

Q1: (a) $\frac{}{5}$	(b) $\frac{}{5}$	(c) $\frac{}{8}$	(d) $\frac{}{4}$	(e) $\frac{}{6}$	(f) $\frac{}{5}$	Total $\frac{}{33}$

• Do not write in this space •

Question 2 The Towers of Hanoi problem is solved by the program

```
hanoi(n, A, B, C)
{
   if (n>0) {
     hanoi(n-1, A, C, B);
     move(A, C);
     hanoi(n-1, B, A, C);
   }
}
```

(a) Let $T(n)$ be the number of times `move` is called to solve the Hanoi problem of size n. Write down a recurrence relation for $T(n)$. *(4 marks)*

$$T(n) = 2\,T(n-1) + 1$$

(b) Write down the boundary condition $T(1)$ and use the recurrence relation to compute $T(2)$, $T(3)$, and $T(4)$ *(4 marks)*

$$T(1) = 1$$

$$T(2) = 2 \times 1 + 1 = 3$$

$$T(3) = 2 \times 3 + 1 = 7$$

$$T(4) = 2 \times 7 + 1 = 15$$

(c) Prove by induction that, $f(n) = 2^n - 1$ satisfies the recurrence relation in part (a) *(8 marks)*

Base case: $f(1) = 2^1 - 1 = 1 = T(n)$
Recursive case: Assume $T(n-1) = f(n-1) = 2^{n-1} - 1$

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

Here is an algorithm for performing sort

```
public static void Sort(double[] data) {
    for (int i = 1; i < data.length; i++) {
        if (data[i]<data[i-1]) {
            double temp = data[i];
```

```
                int j = i;
                do {
                    data[j] = data[j-1];
                    j--;
                } while (j>0 && data[j-1] > temp);
                data[j] = temp;
            }
        }
    }
```

(d) Give upper and lower bounds on the time complexity of the algorithm? Explain your reasoning. *(9 marks)*

(This is actually insertion sort, although that may not be obvious).

The time complexity is $O(n^2)$ and $\Omega(n)$. The program has a while loop embedded in the for loop. The for loop goes through n iterations and the while loop potentially goes through $O(n)$ iterations giving $O(n^2)$. However the while loop could go through $O(1)$ iterations (which would happen if the data was ordered) giving $O(n)$ operations for the whole algorithm.

(e) Briefly explain how binary search works and explain its time complexity *(8 marks)*

(i) Binary search is used to find elements in a sorted array. It works by comparing the middle element of the array with the element being searched. If it fails it the needs to search only the front half or back half of the array depending on whether the element being search if smaller or greater than the middle element. It repeats this recursively.

(ii) The time complexity is $O(\log_2(n))$. After each comparison the number of elements that need to be searched is reduced by two. Thus the total number of comparisons to search an array of n elements is no more than m where $2^m < n < 2^{m+1}$ or $m = \log_2(\lceil n \rceil)$.

End of question 2

Q2: (a) $\frac{}{4}$	(b) $\frac{}{4}$	(c) $\frac{}{8}$	(d) $\frac{}{9}$	(e) $\frac{}{8}$	Total $\frac{}{33}$

• Do not write in this space •

TURN OVER

Question 3

(a) Briefly describe the quick sort algorithm. *(5 marks)*

(Test standard background knowledge.)

Quick sort iteratively divides up an array into those elements that are greater than a pivot value and those elements that are less than or equal to a pivot value.

(b) Explain why choosing a pivot for quick sort is crucial to the performance of the algorithm and explain a common method for choosing a good pivot. *(6 marks)*

(Requires more subtle understanding of the algorithm.)

(i) For quick sort to be efficient it has to split the array into roughly equal parts. If the pivot is ill chosen then the split can be extremely poor. This could happen if we chose the pivot to be the first element in each sub-array and the array was already ordered.

(ii) A common strategy is to choose the pivot to be the median of the first, last and middle element of the array.

(c) What is the worst case and average case time complexity of quick sort. Give a justification for your answer. *(6 marks)*

(Test understanding of time complexity.)

The time complexity depends on the number of splits. After each split all elements have to be moved leading to a time complexity of $n\times$ the number of splits. If the splits are very unfortunate so that only a very small number of elements ends up on one side, then there would be $\Theta(n)$ such splits leading to a worst case tie complexity of n^2. Usually, the splits are benign dividing the array into two, so that there are approximately $\log(n)$ such splits leading to an average time complexity of $n\ \log(n)$.

(d) Explain why no algorithm using binary comparisons can sort an array in less than $\Theta(n \log(n))$ operation on average. *(10 marks)*

(This is a theoretical argument which historically students have struggled with).

To sort n elements it is necessary to obtain every possible permutation of the elements—otherwise, some arrangements cannot be sorted. The number of permutations is $n!$. We can represent the possible decisions of any sorting algorithm using binary comparisons as a binary tree, where the leaves of the tree represent all possible ways of sorting the array. There therefor has to be $n!$ such leaves. The expected number of operations will be the depth of the tree. But a binary tree with k nodes will have an average depth of at least $\log_2(k)$. So the average depth of the decision tree is at least $\log_2(n!) = O(n \log(n))$.

• Do not write in this space •

(e) Describe very briefly how radix sort works on a set of integers. *(4 marks)*

(Test student knowledge of a different sort.)

In radix sort we successively sort on the lower radix (i.e. digit) using a stable sort. Once we the algorithms has worked through all possible digits the numbers will be sorted.

(f) What is the time complexity of radix sort? Why does this not contradict part (c) *(2 marks)*

(Test understanding of an apparent contradiction.)

(i) $\Theta(n)$ **(or more precisely** $\Theta(n \log_r(W))$ **where** W **is the largest number being sorted and** r **is the radix).**

(ii) Radix sort does not use binary comparisons. Its algorithm could not be represented by a *binary* decision tree.

End of question 3

Q3: (a) $\frac{}{5}$	(b) $\frac{}{6}$	(c) $\frac{}{6}$	(d) $\frac{}{10}$	(e) $\frac{}{4}$	(f) $\frac{}{2}$	Total $\frac{}{33}$

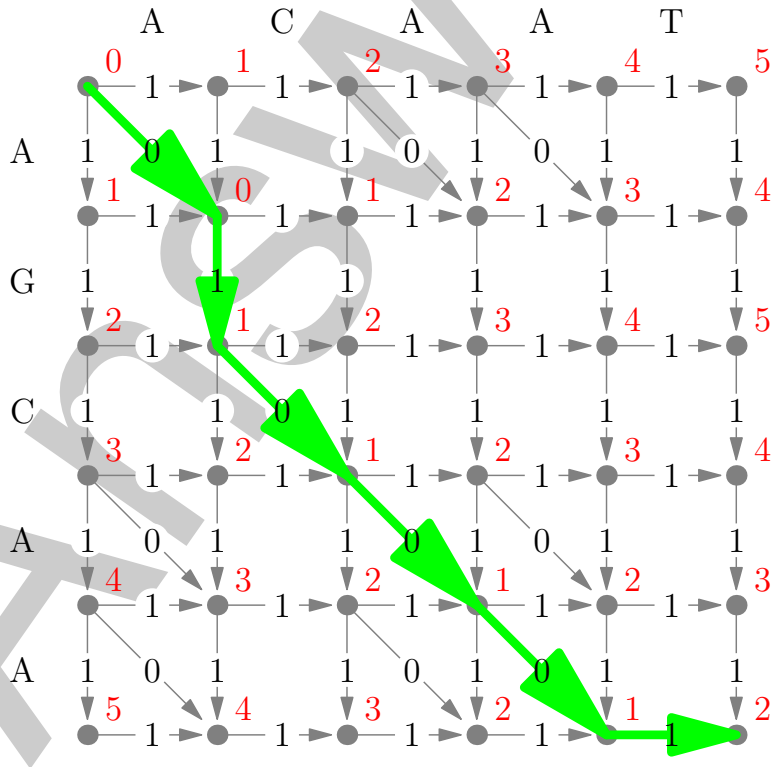
Question 4

(a) Explain the dynamic programming strategy. *(5 marks)*

(Test broad knowledge of dynamic strategy before testing detailed knowledge in the next part.)

In dynamic programming a set of optimal costs is built up, starting from a source, until the destination is reached. At each step of the algorithm the optimal costs are computed from those that have already been computed.

(b) The edit distance between two strings is the number of edit operations needed to go from one string to another. In the simplest case we can consider the only edit operation to be insertion. In this case, the edit distance between the string "AAC" and "ACT" is two as each string would have to have an insertion to make them identical (i.e. both strings could be turned into "AACT"). The edit distance can be computed by finding the lowest cost path on a edit graph starting from the top left hand corner and ending at the bottom right hand corner. The edit graph for two strings "ACAAT" and "AGCAA" is shown below. Using dynamic programming compute the costs of reaching each node (write the costs next to the nodes). Using the backwards algorithm to find the optimal path (show this on the graph). Write down the closest common string.



Closest Common String: AGCAAT

(16 marks)

(c) Briefly describe the following strategies, giving examples of their use. 1) Brute force, 2) Branch and bound, 3) Divide and conquer and 4) Greedy method. *(12 marks)*

(Test broad knowledge of strategies.)

(i) Brute force is a method were the algorithm tries all possible solutions to find the best solution. An example would be to colour a graph by considering all possible colourings.

(ii) Branch and bound is a modification of the brute force where the search tree is pruned by removing partial solutions which cannot be as good as the best solution found so far. Branch and bound could also be applied to graph colouring, eliminating partial colourings with colour conflicts that exceed the best found solution.

(iii) Divide and conquer algorithms attempt to solve a problem, by recursively breaking them into simpler smaller problems. The final solution is found be combining the solutions of the smaller problem. Examples include quick sort, merge sort, binary search, etc.

(iv) The greedy strategy attempts to solve a problem by making locally optimal moves. For some problems, including important problems this leads to a globally optimal solution. Examples include Prim's, Kruskal's and Dijkstra's algorithm as well as Huffman Trees.

End of question 4

Q4: (a) $\frac{}{5}$	(b) $\frac{}{16}$	(c) $\frac{}{12}$	Total $\frac{}{33}$

END OF PAPER