The University of Nottingham

SCHOOL OF COMPUTER SCIENCE

A LEVEL 2 MODULE, AUTUMN SEMESTER, 2018-2019

ALGORITHMS CORRECTNESS AND EFFICIENCY

Time allowed TWO HOURS

Candidates may complete the front cover of their answer book and sign their desk card but must NOT write anything else until the start of the examination period is announced

Answer ALL FIVE questions.

Marks available for sections of questions are shown in brackets in the right-hand margin

No calculators are permitted in this examination.

Dictionaries are not allowed with one exception. Those whose first language is not English may use a standard translation dictionary to translate between that language and English provided that neither language is the subject of this examination. Subject specific translation dictionaries are not permitted.

No electronic devices capable of storing and retrieving text, including electronic dictionaries, may be used.

DO NOT turn your examination paper over until instructed to do so

INFORMATION FOR INVIGILATORS: Students should write their answers to these questions in the answer book. The exam paper should be collected and placed inside the answer book.

1. This question concerns the 'big-Oh' family and recurrence relations. (20 marks total)

   (a) Give the definitions of big-Oh, big-Omega, big-Theta, and little-oh by completing each of the following:

   i) 'big-Oh': f(n) is O(g(n)) ….

   ii) 'big-Omega': f(n) is Ω(g(n)) ….

   iii) 'big-Theta': f(n) is Θ(g(n)) ….

   iv) 'little-oh': f(n) is o(g(n)) ….

   (4 marks)

   (b) Consider the function

$$f(n) = \begin{cases} 2\,n & \text{if } n \text{ is even} \\ 3\,n^2 & \text{otherwise (i.e. if } n \text{ is odd)} \end{cases}$$

   From the definitions:

   i) Prove or disprove that f(n) is O(n^2) (`Big-Oh of n squared').

   ii) Prove or disprove that f(n) is Ω(n^2) (`Big-Omega of n squared').

   iii) Prove or disprove that f(n) is o(n^2) (`little-oh of n squared')

   (9 marks)

   (c) Consider the recurrence relation

$$T(n) = 2\,T(n-1) \quad \text{with} \quad T(1) = 1$$

   Solve the relation exactly; including giving a formula for T(n).

   Then prove your answer is correct using induction.

   (7 marks)

2. This question concerns correctness of algorithms. (20 marks total)

(a) What is meant by the terms *partial correctness* and *total correctness* in relation to the correctness of algorithms?

(2 marks)

(b) *A* is an array of integers of length *A.length*. If *A* is indexed from 1 to *A.length*, state an assertion (in Predicate Calculus) that expresses that the elements in *A* are ordered in non-decreasing order.

(4 marks)

(c) Recall that the Assignment axiom in Hoare's proof calculus is of the form

$$\frac{}{\{\psi[E/x]\}\ \mathtt{x} = E\ \{\psi\}}\ \text{Assignment}$$

Given the program statement

y = y – 1

Use the Assignment axiom to derive the precondition corresponding to each of the following postconditions (note: it is not necessary to simplify the precondition):

(i) y = 0

(ii) x = y

(iii) y + 1 = y * 2

(iv) x < y ^ y > z

(4 marks)

(d) Consider the algorithm `DecValues` below that decrements each value in an array A by one, and returns the modified array as result (arrays are indexed from 1 to A.length):

```
DecValues(A)
   for(i = A.length downto 1)
     A[i] = A[i] - 1
   return A
```

(i) State a suitable loop invariant for the **for** loop in `DecValues` (in English).

(2 marks)

(ii) Give a proof of correctness of `DecValues`. (Hint: state a suitable pre- and postcondition (in English) and use the loop invariant to show that the postcondition follows necessarily from the precondition).

(8 marks)

3. This question concerns binary search trees, amortised complexity, and heaps (20 marks total)

(a) Define the properties that make a binary tree into a Binary Search Tree (BST).
How would a "removeMin()" operation be implemented for a BST? That is, show how to find and remove the minimum element of a BST.
What is the Big-Oh complexity of removeMin(), in terms of the height h, and/or number of nodes n, of the BST?

(5 marks)

(b) Suppose that a sorting procedure, "BST-Sort", is implemented using a BST – in the same fashion as HeapSort, but instead using a BST. In this procedure, n elements are inserted one at a time into an initially-empty BST, and then removed from the BST using removeMin() until the BST is again empty.

Assume the BST is a simple one with no mechanism to maintain balance.

What is the worst-case complexity of BST-Sort?
Justify your answer and give a simple example with n=4.

(5 marks)

(c) Define the properties that make a binary tree into a heap - specifically, a "minheap" with the root containing the minimum value.
Also, give the standard system for converting a binary tree to be stored in an array.

(4 marks)

(d) Suppose that the array for the heap is implemented using an array that is resized each time it gets full (in the same fashion as a Vector).
What is the amortised complexity of inserting elements into the heap, when following each of the two schemes for resizing:

i) Doubling the array size each resize?

ii) Increasing the array size by a constant each resize?

Hint: Consider the insertion of an element, and the up-heap operation separately.

You can answer just in terms of whether the amortised complexity is O(n) or O(log n). You should justify your answers but do not need to provide full proofs.

(6 marks)

4. This question concerns string matching. (20 marks total)

(a) The algorithm `NumMatches` below returns the number *k* of (possibly overlapping) occurrences of a pattern *P* in a text *T*.

```
NumMatches(T,P)
  k = 0
  for(s = 0 to T.length - P.length)
    j = 1
    while(j =< P.length and P[j] == T[s + j])
      j++
    if(j == P.length + 1)
      k++
  return k
```

State suitable invariants for the inner and outer loops (in English) and show initialisation and maintenance of the outer **for** loop (hint: to show initialisation and maintenance of the outer loop, you will need to show initialisation and maintenance of the inner loop; note that you *don't* need to show correctness of the whole algorithm, or termination).

(10 marks)

(b) Modify the `NumMatches` algorithm given in part (a) to return the number of *non-overlapping* occurrences of a pattern in a text, that is, for the pattern *P* = aba and the text *T* = ababa the modified algorithm should return 1 rather than 2 as for `NumMatches`.

(2 marks)

(c) Assuming the modified algorithm is only called when the length of the pattern is less than or equal to the length of the text:

(i) what is the *best case* big-Oh complexity of the modified algorithm; briefly explain your answer

(2 marks)

(ii) what is the *worst case* big-Oh complexity of the modified algorithm; briefly explain your answer

(2 marks)

(d) Recall that the Knuth-Morris-Pratt string matching algorithm makes use of an auxiliary prefix function π. Given a pattern $P[1 \ldots m]$, the prefix function for *P* is the function $\pi:\{1, 2, \ldots m\} \rightarrow \{0, 1, \ldots m - 1\}$ such that

$$[q] = \max\{\, k : k < q \text{ and } P_k \sqsupset P_q \,\}$$

that is, $\pi[q]$ is the length of the longest prefix P_k, that is a proper suffix of P_q.

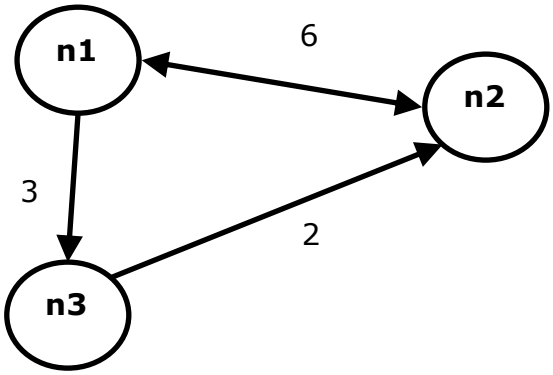
Compute the prefix function for the pattern *P* = ababaaba. Hint: for each prefix P_q of *P* in turn, find the length of the longest string that is both a proper prefix and suffix of P_q.

(4 marks)

5. This question concerns dynamic programming methods. (20 marks total)

(a) Explain, and give pseudo-code for, the Floyd-Warshall algorithm to find the lengths of the shortest paths between each pair of vertices in a graph. The graph can have a mix of directed and bidirectional edges, with distance d(i, j) from node i to node j. You can assume the distances are such that there are no negative cycles.

Then use the algorithm on the (directed) graph below to find the matrix of distances for all-pairs of shortest paths. Show your working.



(10 marks)

(b) Consider a container collection problem in which containers are to be collected from different cities and to be returned to a central depot. Each trip to pick up a container must be made using a separate round trip, and taking a given distance.
In order to be paid the driver must cover at least a target distance T, and they cannot travel any further than this due to regulations on the total distance travelled. Since the driver is paid per container collected they want to satisfy the total travelling distance being T, but visiting the maximum number of cities.

Give and explain an algorithm to solve this problem. (Your algorithm must be better than simply enumerating all combinations.)

Illustrate how your algorithm works, using a target T=5, and the following set of cities and round trip distances:

City	Round trip distance
A	4
B	1
C	2
D	2

(10 marks)

END