UNIVERSITY OF WESTMINSTER

School of Computer Science and Engineering

Module:	**Algorithms: Theory, Design and Implementation**
Module Code:	**5SENG001W**
	5SENG002C (Sri Lanka)
Exam period:	**May 2019**
Duration:	**90 minutes**

Instruction:

You are advised (but not required) to spend the first ten minutes of the examination reading the questions and planning how you will answer those you have selected.

Calculators may be used provided they are silent, cordless, not pre-programmed by the candidate and cannot receive or transmit data remotely.

THIS PAPER MUST NOT BE TAKEN OUT OF THE EXAMINATION ROOM

DO NOT TURN OVER THIS PAGE UNTIL THE INVIGILATOR INSTRUCTS YOU TO DO SO

SECTION A: THEORY OF ALGORITHMS [40 marks]

ANSWER **ALL** QUESTIONS IN SECTION A.

Q1 [5 marks]: Arrange the following algorithmic performances in increasing order of growth rate, n standing for the input data size. You should simply put forward the sequence of characters from smallest to largest, e.g., a->b->c->d->e

a) $\sqrt{n}$
b) 10^n
c) $n^{1.5}$
d) $2^{\sqrt{\log(n)}}$
e) $n^{5/3}$

A1: d->a->c->e->b **[5 marks]**

Q2 [11 marks]: Suppose that the performance, in terms of asymptotic analysis, of two algorithmic designs for the same problem solution has been estimated as being

a) **$O(n^2 \log n)$ / $\Omega(n)$ / $\Theta(n^2)$**
b) **$O(n^2 \log n)$ / $\Omega(\log n)$ / $\Theta(n^2)$**

Which of the two algorithmic designs looks more promising in terms of a better performance? Justify your answer by putting forward the definitions as boundaries of the algorithmic performance for the three asymptotics Big-O, Big-Omega and Big-Theta.

A2:

Right answers: b **[2 marks]**

Justification:

Big-O notation can be interpreted as posing an upper bound of the performance of an algorithm, meaning that the algorithm can not perform an order of growth rate **worse** than the stated one. **[2 marks]**

Big-Ω notation can be interpreted as posing a lower bound of the performance of an algorithm, meaning that the algorithm cannot perform an order of growth rate **better** than the stated one. **[2 marks]**

Big-Θ notation can be interpreted as posing an indicative performance of an algorithm designated **between the upper and lower bounds** denoted by the Big-O and Big-Ω asymptotics. **[2 marks]**

Given that the only difference between the asymptotics for the two algorithms is in terms of the Big-Ω notation, algorithm **b** denotes a lower bound (logarithmic) than the one for algorithm **a** (linear), hence, algorithm (b) can be further pushed towards better performance. **[3 marks]**

Q3 [8 marks]: Let **$T(n) = ½ n^2 + 3n$** be the estimated performance of an algorithm, with ***n*** standing for the input data size and **T** for the time being spent running and terminating the algorithm. Which of the following statements hold true? Provide a justification to your answers.

1. $T(n) = O(n)$
2. $T(n) = \Omega(n)$
3. $T(n) = \Theta(n^2)$
4. $T(n) = O(n^3)$

A3:

Right answers: 2, 3, 4 **[2 marks]**

Justification:

3: T of N is definitely a quadratic function. We know that the linear term doesn't matter much as it grows, as N grows large. So since it has quadratic growth, then the third response should be correct. It's theta of N squared. **[2 marks]**

2: Omega of N is not a very good lower bound on the asymptotic rate of growth of T of N, but it is legitimate. Indeed, as a quadratic growing function, it grows at least as fast as a linear function. So, it's Omega of N. **[2 marks]**

4: For the same reason, big O of N cubed, it's not a very good upper bound, but it is a legitimate one, it is correct. The rate of growth of T of N is at most cubic. In fact, it's at most quadratic, but it is indeed, at most, cubic as well. **[2 marks]**

Q4 [5 marks]: Suppose that instead of dividing in half at each stage of Merge-Sort, you divide into thirds, sort each third and finally combine all of them using a three-way merge subroutine. What is the overall asymptotic running time of this algorithm? You should assume that the merge step can still be implemented in O(n) time. Justify your answer.

1. **$n^2 \log(n)$**
2. **n**
3. **$n \log(n)$**
4. **$n (\log(n))^2$**

A4:

Right answer: **n log(n)** **[2 marks]**

Justification: There is still a logarithmic number of levels and the overall amount of work at each level is still linear. **[3 marks]**

Q5 [11 marks]: Suppose you are asked to design and implement a machine learning algorithm for weather predictions over the UK. The predictions should be *sunny, rainy, stormy, foggy, cloudy.*

a) Which category of algorithms should be considered for your approach when it comes to selecting your algorithm?
b) How shall we measure the algorithmic performance?

A5:

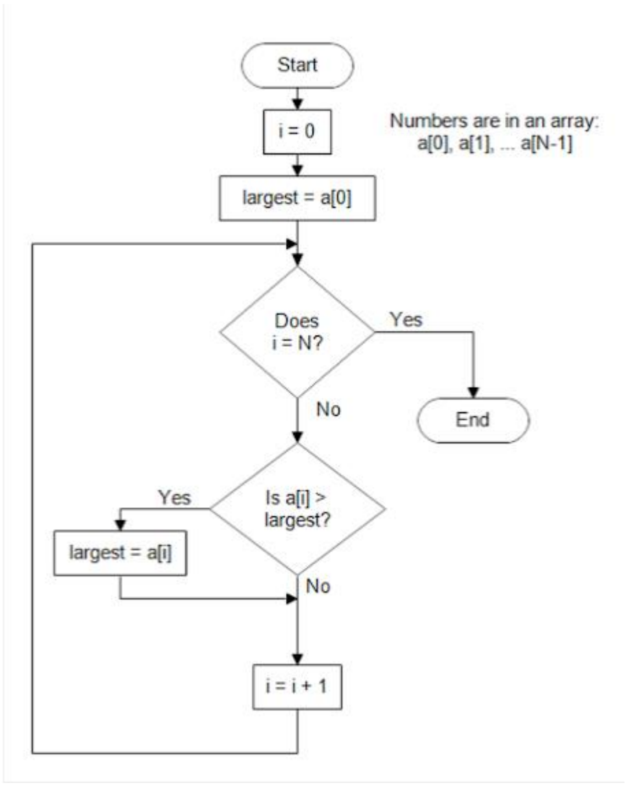
a) Given that the type of problem is a classification (predict a category) one and not regression (predict a real value) one, we should target classification algorithms, including logistic regression algorithms. **[4 marks]**
b) Design and execute controlled experiments with the selected machine learning algorithms and then analyse the results in terms of accuracy in their predictions against, for instance, test data [*4 marks*]. This can be measured in terms of the metrics *Precision, Recall and F-measure* [*3 marks*]. **[7 marks]**

SECTION B: DESIGN AND IMPLEMENTATION OF ALGORITHMS [60 marks]

ANSWER **ALL** QUESTIONS IN SECTION B.

Q6 [6 marks]: Design an algorithm to find the largest number in an array of N integers, a[0], a[1], ….a[N-1]. Use a flowchart diagram with the following two shapes: *rectangle* for initialisations and variable settings, *diamond* for decision making. *Arrows* should be connecting the shapes demonstrating the flow of the algorithm. You may also use structured plain English as pseudo-code with statements as close as possible to the main statements of programming languages (e.g., SET, WHILE…DO, IF…THEN…ELSE)

A6: **[1 mark per correct statement]**



Q7 [14 marks]: Design an algorithm to find the set of triples of integers the sum of which equals to zero. Assume that your input data is an array a[0], a[1], ….a[N-1] of N integers, both positive and negative, which are randomly generated, i.e., numbers are not sorted. The algorithm must be more effective than O(N^3) as of a three nested loop, exhaustive search and brute-force approach. Use structured plain English as pseudo-code with statements as close as possible to the main statements of programming languages (e.g., SET, WHILE…DO, IF…THEN…ELSE)

A7:

- Sort the input array in ascending order; [2 marks]
- SET index i = 0; [1 mark]
- SET index j = 1; [1 mark]
- FOR (i == 0; i less than N, Increment i by one) [1 mark]
    - FOR (j == 1; less than N, increment j by one) [1 mark]
        - *Pair* = a[i] + a[j] [2 marks]
        - *ThirdNumber* = –(a[i]+a[j]) [2 marks]
        - Apply *binary search* for *ThirdNumber* on array *a from j+1 onwards* [2 marks]
        - IF *ThirdNumber* exists THEN
            - Return triple {a[i], a[j], ThirdNumber} [2 marks]

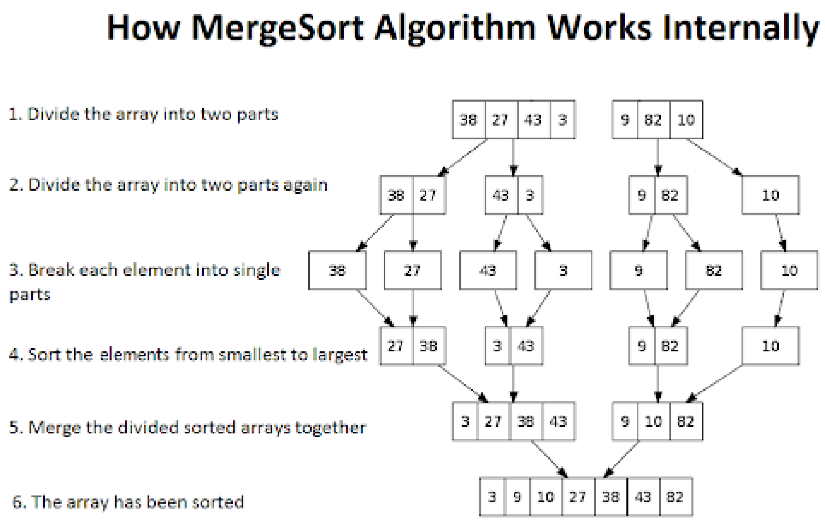
Q8 [12 marks]: You have given an unordered list of integers, which must be rearranged in ascending order by applying the *Merge Sort* algorithm.

Input Array: {38, 27, 43, 3, 9, 82, 10}

Output Array: {3, 9, 10, 27, 38, 43, 82}

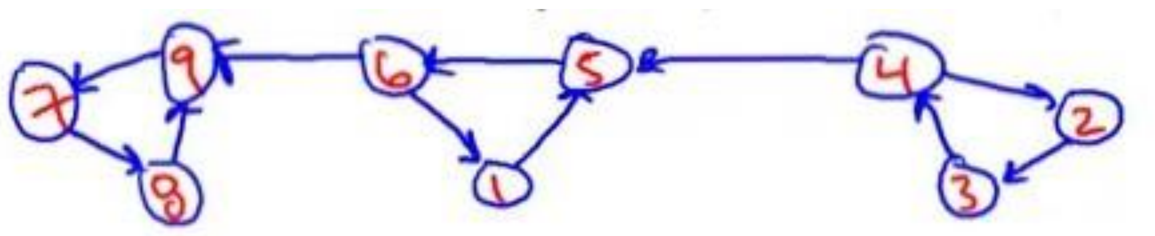
Draw the tree of all sub-arrays, which will be generated by the algorithm starting with the divisions of the input array and terminating with the synthesised output array with the sorted integers after all necessary mergers of sub-arrays.

A8:


How MergeSort Algorithm Works Internally
1. Divide the array into two parts
2. Divide the array into two parts again
3. Break each element into single parts
4. Sort the elements from smallest to largest
5. Merge the divided sorted arrays together
6. The array has been sorted
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

[2 marks per level]

Q9 [14 marks]: Suppose you are given the following directed graph as input for a graph traversal algorithm based on DFS (Depth-First-Strategy).



Outline the labelled nodes to be visited while searching for node labelled 3. You should assume that the lead node to enter the graph will be the node with the highest label (number), which has not been visited yet and has outgoing edges.

A9:

1. Visit lead node: 9	[1 mark]
2. Visit Node 7	[1 mark]
3. Visit Node 8	[1 mark]
4. Visit Node 9 (already visited)	[1 mark]
5. Visit New lead node: 6 (since nodes 8 and 7 are already visited)	[2 marks]
6. Visit Node 1	[1 mark]
7. Visit Node 5	[1 mark]
8. Visit Node 6 (already visited)	[2 marks]
9. Visit New lead node: 4 (since nodes 5 and 1 are already visited)	[2 marks]
10. Visit Node 2	[1 mark]
11. Visit Node 3 (FOUND)	[1 mark]

Q10 [14 marks]: Suppose you are given the following term – document incidence matrix indicating the appearance of a keyword (vertical axis) with the writing and theatrical play (horizontal axis) as an entry marked by one (1).

Suppose you are also asked to design a keywords-based search algorithm as *Boolean* operators returning as results the set of documents within which a set of given keywords occurs. For instance, the search *Brutus AND Julius Caesar* shall return documents {*Antony and Cleopatra, Julius Caesar, Hamlet*}

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

You quickly notice, however, that this matrix-based data structure is not the optimal solution for large matrices (e.g., case of real libraries) as there are many zero entries.

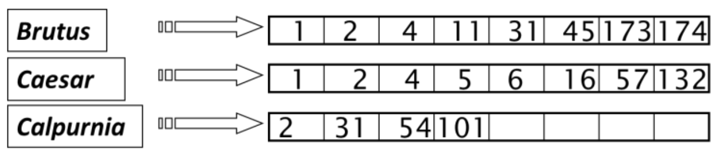
- Suggest a more efficient data structure, which does not waste memory with too many zero entries.
- Subsequently, suggest the key concept of the algorithm and outline the main algorithmic approach as pseudo-code in plain English for returning documents where AND connected key words are given as input, e.g., *Brutus AND Julius Caesar.*

For the sake of simplicity, suppose that your search algorithm relies on exact matching and shall work with AND connected keywords only.

A10:

Identify each document by a docID, a document serial number; [2 marks]

For each term t, create a list, e.g., an array or linked list, of all documents that contain t; [3 marks]



The key idea of the algorithm is to merge the sets of documents within which each key word appears and subsequently build the intersection set by merging the two lists. [5 marks]

An exemplary design of the algorithm for two key words is given below. [4 marks]

```
INTERSECT($p_1, p_2$)
 1  $answer \leftarrow \langle\ \rangle$
 2  while $p_1 \neq$ NIL and $p_2 \neq$ NIL
 3  do if $docID(p_1) = docID(p_2)$
 4        then ADD($answer, docID(p_1)$)
 5             $p_1 \leftarrow next(p_1)$
 6             $p_2 \leftarrow next(p_2)$
 7        else if $docID(p_1) < docID(p_2)$
 8               then $p_1 \leftarrow next(p_1)$
 9               else $p_2 \leftarrow next(p_2)$
10  return $answer$
```

END OF EXAM PAPER