UNIVERSITY OF LONDON

CM2035

BSc EXAMINATION

COMPUTER SCIENCE

Algorithms and Data Structures II

Release date: Wednesday 16 March 2022 at 12:00 midday Greenwich Mean Time

Submission date: Thursday 17 March 2022 by 12:00 midday Greenwich Mean Time

Time allowed: 24 hours to submit

INSTRUCTIONS TO CANDIDATES:

Section A of this assessment paper consists of a set of **TEN** Multiple Choice Questions (MCQs) which you will take separately from this paper. You should attempt to answer **ALL** the questions in Section A. The maximum mark for Section A is **40**.

Section A will be completed online on the VLE. You may choose to access the MCQs at any time following the release of the paper, but once you have accessed the MCQs you must submit your answers before the deadline or within **4 hours** of starting, whichever occurs first.

Section B of this assessment paper is an online assessment to be completed within the same 24-hour window as Section A. We anticipate that approximately **1 hour** is sufficient for you to answer Section B. Candidates must answer **TWO** out of the THREE questions in Section B. The maximum mark for Section B is **60**.

Calculators are not permitted in this examination. Credit will only be given if all workings are shown.

You should complete **Section B** of this paper and submit your answers as **one document**, if possible, in Microsoft Word or a PDF to the appropriate area on the VLE. Each file uploaded must be accompanied by a coversheet containing your **candidate number** written clearly at the top of the page before you upload your work. Do not write your name anywhere in your answers.

UL22/0102

SECTION A

Candidates should answer the **TEN** Multiple Choice Questions (MCQs) quiz, **Question 1** in Section A on the VLE.

SECTION B

Candidates should answer any **TWO** questions in Section B.

Question 2

This question is about sorting and hashing.

(a) Explain a situation where it is better to use Counting Sort rather than Merge Sort to sort an array in ascending order.

(4 marks)

(b) Explain a situation where it is better to use Merge Sort rather than Counting Sort to sort an array in ascending order.

(4 marks)

(c) Briefly explain why hash tables are useful for storing many integers.

(4 marks)

(d) Consider the following piece of pseudocode:

```
function Sort(A,N,k,a,b)
  C=new array(N) of zeroes
  R=new array(N) of zeroes
  pos=0
  for 0 <= j < N
     C[(a*A[j]+b)%N]=C[(a*A[j]+b)%N]+1
  for 0 <= i < N
     for pos <= r < pos+C[(a*i+b)%N]
        R[r]=i
     pos=r
  return R
end function
```

The inputs are an array A of non-negative integers of length N, k is the largest integer value in A, and a and b are positive integers. A colleague has proposed this as a sorting algorithm that utilises a hash table.

i. What is the worst-case time complexity of Sort? Use Theta notation, and explain your answer.

(5 marks)

ii. Describe general input arrays of length N for which this pseudocode will correctly sort an input array in ascending order.

(5 marks)

iii. Explain why this algorithm will not sort an array in general.

(8 marks)

Question 3

The following two algorithms corresponding to functions A1 and A2 claim to solve the same problem. To do so, they receive as input arguments:

root: the root of a binary search tree (BST) storing positive integer numbers greater than 0
B: non-negative integer

The operation `floor(x)` returns the largest integer smaller than or equal to `x`.

ALGORITHM 1

```
function A1(root,B)
  Q = new Queue()
  ENQUEUE(Q,root)
  while !ISEMPTY(Q) do
     t = PEEK(Q)
     if(t->data==B):
        return A1a(t)
     else:
        ENQUEUE(Q,t->left)
        ENQUEUE(Q,t->right)
        DEQUEUE(Q)
  end while
  return -1
end function

function A1a(root)
  S=new Stack()
  i=0
  x=root
  while !ISEMPTY(S) or x!=NULL
do
    if(x!=NULL):
       PUSH(S,x)
       x=x->left
    else:
       x=PEEK(S)
       POP(S)
       i=i+1
       x=x->right
  return i
end function
```

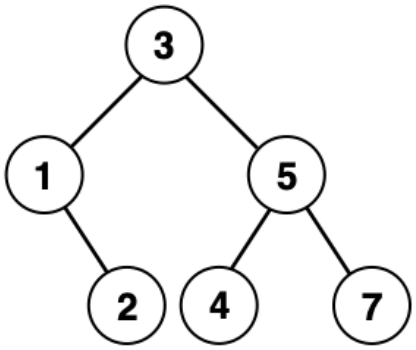
Note: the function ENQUEUE only inserts a new element in the queue if this element is different from NULL

ALGORITHM 2

```
function A2(root,B)
  if(root==NULL):
     return -1
  if(B == root->data):
     return A2a(root)
  else:
     if(B<root->data):
        return A2(root->left,B)
     else:
        return A2(root->right,B)
end function

function A2a(root)
  if(root == NULL)
     return 0
  return 1+A2a(root->left)+A2a(root-
 >right)
end function
```

Consider the following BST:



(a) For this BST, what is returned by the function call A2(root,5)?

(2 marks)

(b) For this BST, what is returned by the function call A1(root,3)?

(2 marks)

(c) What is the task performed by algorithms A1 and A2?

(4 marks)

(d) What is the worst-case time complexity of A1 for a BST of N nodes? Use Theta notation (1 mark) and briefly explain your reasoning.

(5 marks)

(e) What is the best-case time complexity of A2 for a BST of N nodes? Use Theta notation (1 mark) and briefly explain your reasoning.

(5 marks)

(f) Which of the two algorithms would you recommend to use? Explain your answer.

(6 marks)

(g) One of your colleagues has claimed that they can find an algorithm that performs the same task as A1 and A2, but the worst-case time complexity is $\Theta(\log N)$, for N nodes in a BST. Do you agree or disagree with your colleague? Explain the reasoning behind your opinion.

(6 marks)

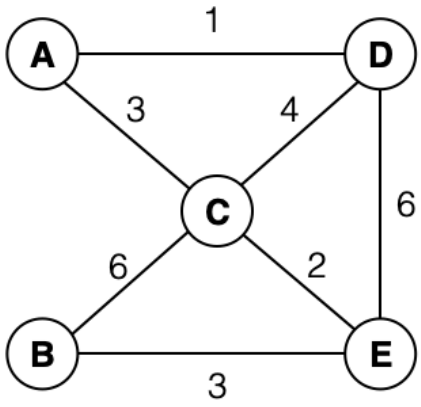
Question 4

This question is about graphs.

(a) When is it better to use an adjacency list to represent a graph rather than using an adjacency matrix? Explain your reasoning.

(6 marks)

(b) Consider the following weighted, undirected graph:



By hand, go through an implementation of Dijkstra's algorithm on this graph to find the shortest path between nodes A and B. Your implementation should produce both the path's length and the nodes in it.

(8 marks)

(c) Explain why a binary min heap is used in Dijkstra's algorithm.

(6 marks)

(d) Instead of finding the shortest path between two nodes in a weighted, undirected graph, we want to find the longest path between two nodes. That is, the path between two nodes where each node is visited at most once (or not at all) with the largest sum of weights. One way to solve this problem is to change each weight *w* in the graph to *-w* (negate the weights), and then find the shortest path in the graph.

i. Can Dijkstra's algorithm be used to find the shortest path on the graph with negated weights, and thus solve the longest path problem on the original graph? Explain your answer.

(3 marks)

ii. A colleague claims that they found another algorithm to solve the longest path problem on a weighted, undirected graph. They also claim that it has the same worst-case time complexity as Dijkstra's algorithm. What is your opinion of their claim? Explain your answer.

(7 marks)

END OF PAPER