UNIVERSITY OF LONDON

CM1035

BSc EXAMINATION

COMPUTER SCIENCE

Algorithms and Data Structures I

Release date: Wednesday 8 September 2021 at 12:00 midday British Summer Time

Submission date: Thursday 9 September 2021 by 12:00 midday British Summer Time

Time allowed: 24 hours to submit

INSTRUCTIONS TO CANDIDATES:

Section A of this assessment paper consists of a set of **10** 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 PDF to the appropriate area on the VLE. You are permitted to upload 30 documents. However, we advise you to upload as few documents as possible. 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.

UL21/1047

SECTION B

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

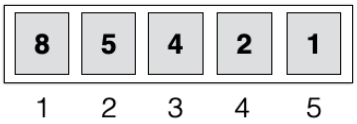
Question 2

This question is about sorting and searching.

(a) From the point of view of searching a vector of integers for a particular integer, briefly explain why it is better if the vector is already sorted.

(4 marks)

(b) Consider the following vector of integers:



You need to sort this vector by hand using an algorithm so that the smallest value is in the first element and the largest value is in the last element. Showing your working and how the vector changes in each step of the algorithm, implement the Bubble Sort algorithm on this vector.

(7 marks)

(c) Briefly explain why the vector in part (b) of this question is an example of a worst-case input for the Bubble Sort algorithm.

(4 marks)

(d) Another sorting algorithm is Quicksort. Explain a situation where you would prefer to use Bubble Sort rather than Quicksort.

(5 marks)

(e) Imagine you work for a website called ‘NastyVegetables.com’ that is making a list of the most popular movies from 2019 based on public votes. Two votes were taken at different times in 2020 and you need to combine the results.

You are given two unsorted JavaScript arrays called `firstVote` and `secondVote`, both of length n with each element storing a two-element array of the form:

```
[“Movie 2”, 167]
```

Where the name of the movie is stored in the first element and the number of votes it got is stored in the second element. Each movie appears only once in each of the two arrays.

You are given the following piece of JavaScript code:

```
function makeOneList (firstVote, secondVote) {
     var n = firstVote.length;
     var finalList = [];
     var votes = [];
     var sum = 0;
     for (var i = 0; i < n; i++) {
          votes = [];
          for (var j = 0; j < n; j++) {
               if (secondVote[j][0] == firstVote[i][0]) {
                    votes.push (firstVote[i]);
                    sum = firstVote[i][1] + secondVote[j]
               [1];
                    votes.push (sum);
               }
          }
          finalList.push(votes);
     }
     return finalList;
}
```

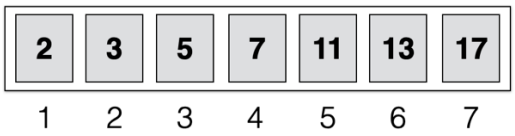
1. Briefly explain why the worst-case time complexity of the algorithm being implemented by this function if $O(n^2)$ for two arrays both of length n. We can assume that the length of the arrays is stored in memory and can be retrieved in a constant number of steps. (4 marks)

2. A colleague of yours has claimed that they have an algorithm that can also create a single array just as `makeOneList` does, but it works for two sorted input arrays and its worst-case time complexity is $O(\log_2 n)$ for two arrays both of length n. Briefly explain whether you think your colleague's claim is correct, and explain your reasoning. (6 marks)

Question 3

This question is about searching and solving a problem in modular arithmetic.

(a) Consider the following vector of integers:



By hand, directly run through the Binary Search algorithm on this vector searching for the value **5**. Show your working and how you choose elements to inspect.

(7 marks)

(b) In can be argued that the Binary Search algorithm is optimal. Very briefly explain what this means. You do not have to provide an argument, just explain what the statement means.

(3 marks)

(c) A modular square root of a non-negative integer *y* modulo *N* is a non-negative integer *x* such that $x^2 \equiv y \pmod{N}$, i.e. x^2 and *y* are congruent modulo *N*. For instance, if *y* is 2 and *N* is 7, then *x* could be 3 since 9 mod 7 is 2, or *x* could be 4 since 16 mod 7 is also 2. That is, there can be multiple square roots. (There is a short revision on modular arithmetic at the end of this question.) The modular square roots *x* and *y* are also assumed to be integers less than *N*. Note that *N* is always assumed to be greater than 0.

Consider the following piece of pseudocode that finds the smallest square root of an integer *y* modulo *N*:

```
function RecMod(a, N)
    if a < N then
        return a
    end if
    return RecMod(a − N, N)
end function

function ModSquareRoot(y, N)
    for 0 ≤ x ≤ N do
        if RecMod(x², N) = y then
            return x
        end if
    end for
end function
```

1. Briefly explain why the worst-case time complexity of RecMod(*a, N*) is *O(a/N)* for inputs *a* and *N.* (5 marks)

2. What is the worst-case time complexity of MODSQUAREROOT(*y, N*) in Big O notation and in terms of the inputs *N* and/or *y*? (2 marks)

3. Briefly explain your solution to part 2 of (c), i.e. the worst-case time complexity of MODSQUAREROOT(*y, N*). (5 marks)

(d) Consider the following piece of pseudocode that claims to find the smallest square root of an integer *y* modulo *N*:

```
function MODNEW(a, N)
    r ← ⌊a/N⌋
    return a − (r×N)
end function

function MODROOTNEW(y, N)
    a ← 0
    b ← N
    while a ≤ b do
        q ← ⌊ N/2 ⌋
        if MODNEW(q², N) = y then
            return q
        else if MODNEW(q², N) < y then
            a ← q + 1
        else
            b ← q − 1
        end if
    end while
end function
```

If this algorithm worked, in general, it would give an algorithm that could find the smallest modular square root of an integer *y* in time that is *O(log N)* for integer input *N*. Explain why this algorithm cannot work. You can use the internet to perform research for your answer, as long as you appropriately reference the materials used.

(8 marks)

Short Modular Arithmetic Revision

A non-negative integer is an integer greater than or equal to 0. Every non-negative integer *a* can be written as $a = pN + q$ where *p*, *q* and *N* are all non-negative integers and $N > 0$, $q < N$. In modular arithmetic modulo *N* we restrict to arithmetic over non-negative integers between (and including) 0 and $N - 1$. For every non-negative integer *a*, the integer *a* (mod *N*) is exactly equal to the value q above, i.e. once we delete the part of *a* that can be perfectly divided by *N*, which is *pN*, we are left with *q*. For instance, 17 (mod 4) is equal to 1 since 17 = 4x4 + 1. We never take *N* to be zero since we cannot divide a number by 0.

Question 4

This question is about recursion.

(a) Very briefly explain how a stack can be used to implement recursive functions. (4 marks)

(b) Consider the following piece of pseudocode:

```
1: function RecSum(n)
2:     if n = 0 then
3:         return 0
4:     end if
5:     return n + RecSum(n − 1)
6: end function
```

This function returns the sum of all non-negative integers from 0 to n, given the input parameter n.

1. What is returned by the function call RecSum(3)? (1 mark)

2. Very briefly say what is the role of lines 2 to 3 (inclusive) in the pseudocode above. (3 marks)

(c) Consider the following piece of incomplete pseudocode:

```
function Sum(n)
    a ← 0
    while n > 0 do
        a ← a + n
        n ← MISSING
    end while
    return a
end function
```

When completed, this function should return the same values that RecSum(n) returns for input parameter n.

1. What should go in the place of MISSING to complete this pseudocode? (1 mark)

2. Briefly explain why the worst-case space complexities of Sum(n) and RecSum(n) are O(1) and O(n) respectively for input parameter n. (5 marks)

(d) Consider the following piece of JavaScript code:

```
function sumOfArray(array, l, r) {
     if (l > r) {
          return 0;
     } else if (l == r) {
          return array[l];
     }
     var m = Math.floor((l + r)/2);
     var sumL = sumOfArray(l, m);
     var sumR = sumOfArray(m + 1, r);
     return sumL + sumR;
}

function sumOfNumbers(n) {
     if (n == 0) {
          return 0;
     }
     for (var i = 1; i <= n; i++){
          array.push(i);
     }
     return sumOfArray(array, 0, n – 1);
}
```

1. Briefly explain why the function `sumOfArray` is an example of an implementation of a divide and conquer algorithm. (3 marks)

2. What is the worst-case time complexity of the algorithm being implemented by `sumOfArray`, written in Big O notation and in terms of n, the length of the input array? Assume that when `sumOfArray` is called, the input parameters are: the array, `l` is 0 and `r` is n - 1. Also assume computing the floor takes constant time. (2 marks)

3. Briefly explain your answer to part 2 of question (d). (5 marks)

4. The algorithm implemented by the function `sumOfNumbers` calculates the sum of all non-negative integers from 0 to n (inclusive), just like RECSUM from question (b). Explain which of the two algorithms you would prefer to use to compute this sum, and why. In particular, compare the worst-case time and space complexities to make your choice. (6 marks)

END OF PAPER