UNIVERSITY
OF LONDON

CM1035

BSc EXAMINATION

COMPUTER SCIENCE

Algorithms and Data Structures I

Time allowed: 2 hours

DO NOT TURN OVER UNTIL TOLD TO BEGIN

INSTRUCTIONS TO CANDIDATES:

This examination paper is in two parts: Part A and Part B. You should answer **ALL** of question 1 in Part A and **TWO** questions from Part B. Part A carries 40 marks, and each question from Part B carries 30 marks. If you answer more than **TWO** questions from **Part B** only your first **TWO** answers will be marked.

All answers must be written in the answer books; answers written on the question paper will not be marked. You may write notes in your answer book. Any notes or additional answers in the answer book(s) should be crossed out.

The marks for each part of a question are indicated at the end of the part in [.] brackets. There are 100 marks available on this paper.

Calculators are not permitted in this examination.

© University of London 2020

PART A

Candidates should answer **ALL** of Question 1 in Part A.

Question 1   If instructed to choose only one option, do not choose multiple options. If you select multiple options, your answer will be marked as 0. If otherwise instructed to select all options that apply, there will be multiple correct options, but you can lose marks for selecting incorrect options.

(a) Consider the following problem:

Given a non-negative integer n, is n^2 odd?

Which of the following is the type of the expected output for this problem? Choose only one option.   [3]

  i. Integer

  ii. A fraction

  iii. Boolean

  iv. None of the above.

(b) Which of the following statements describes what an algorithm is? There is only one correct answer.   [2]
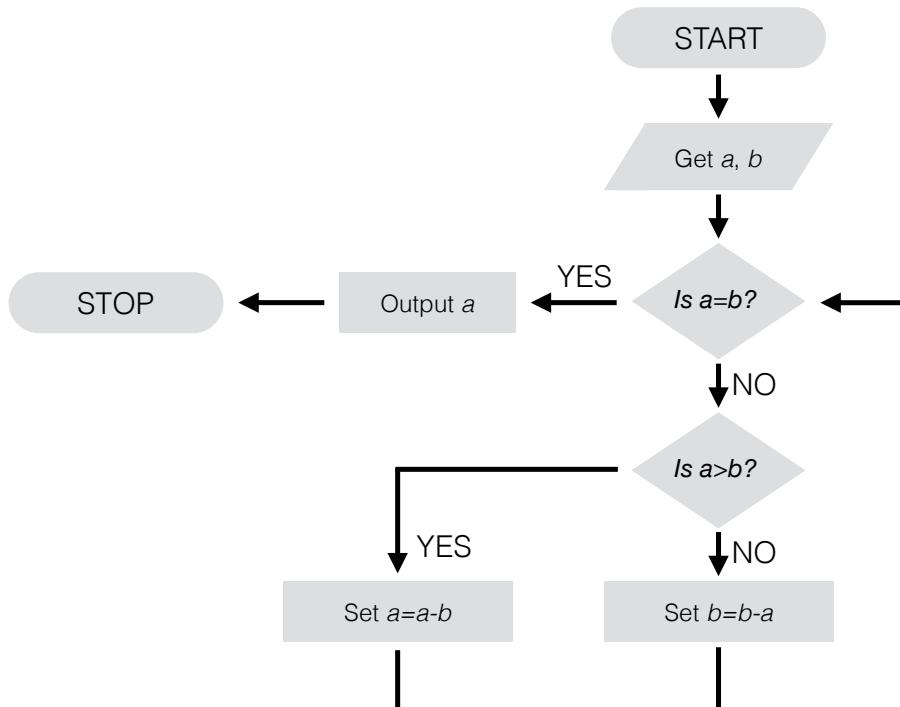
  i. An algorithm is only a method for checking if computer programs are correct.

  ii. An algorithm is only a method for finding if a number is an integer.

  iii. An algorithm is a method of step-by-step basic instructions which, if followed, solves a problem.

  iv. An algorithm is a set of problems that can be solved by a computer.

(c) Consider the following piece of pseudocode:

```
 1: x ← 2
 2: x ← x + 2
 3: function CHECK(n)
 4:     if n = 2 then
 5:         n ← n + 1
 6:     end if
 7:     return n − 1
 8: end function
 9: x ← CHECK(x)
10: x ← CHECK(x)
```

   i. What is the value of x at the end of line 1? [1]

   ii. What is the value of x at the end of line 2? [1]

   iii. What is returned by CHECK(2)? [1]

   iv. What is the value of x at the end of line 9? [1]

   v. What is the value of x at the end of line 10? [1]

(d) Consider the following flowchart:

The following incomplete pseudocode should implement the algorithm in this flowchart:

```
 1: function EUCLID(a,b)
 2:     while a ≠ b do
 3:         if MISSING then
 4:             a ← a − b
 5:         else
 6:             b ← b − a
 7:         end if
 8:     end while
 9:     return a
10: end function
```

Which of the following should go in the place of MISSING? Choose only one option. [3]

   i. $b < a$

  ii. $b \neq a$

 iii. $b > a$

(e) From where are elements removed in a stack? Choose only one option. [3]

   i. Pop

  ii. Head

 iii. Tail

 iv. Top

(f) Consider the following piece of pseudocode:

```
 1: new DynamicArray d
 2: store ← 0
 3: for 1 ≤ i ≤ 5 do
 4:     d[i] ← i
 5: end for
 6: store ← store + LENGTH[d]
 7: store ← store + d[4] + d[5]
```

   i. What is the value of *store* at the end of line 6? [3]

  ii. What is the value of *store* at the end of line 7? [3]

(g) Consider the following piece of incomplete Javascript:

```
1  var d = [];
2  var store = 0;
3  for (var i = 0; i < 5; i++) {
4    MISSING1;
5  }
6  store = MISSING2;
7  store = store + d[3] + d[4];
```

When completed, this code should implement the pseudocode in question 1(f). What should go in the place of MISSING1 to complete this code? Choose only one option. [2]

   i. `d[i] = i;`

  ii. `d(i) = i;`

 iii. `d.i = i;`

(h) What should go in the place of MISSING2 to complete the code in part (g) of this question? Choose only one option. [2]

   i. `store + d[4].length`

  ii. `store + d.length`

 iii. `store + d[4]`

(i) Which of the following describes the difference between a dynamic array and a stack? Choose only one option. [3]

   i. The number of elements in a stack is fixed, but in a dynamic array the number of elements can change

  ii. A stack has the LENGTH operation, but a dynamic array does not

 iii. The number of elements in a dynamic array is fixed, but in a stack the number of elements can change

 iv. A dynamic array has the LENGTH operation, but a stack does not

(j) Which of the following is the worst-case time complexity in 'Big O' notation of the linear search algorithm in n for a vector of length n? Choose only one option. [4]

    i. O(log n)

    ii. O(n^2)

    iii. O(n)

    iv. None of the rest

(k) Consider the following piece of incomplete pseudocode for a function SUM(n), which takes the number n as an input parameter, and should return the sum of all integers from 0 to n.

> **function** SUM(n)
>   **if** n $= 0$ **then**
>   **return** 0
>   **end if**
>   MISSING
> **end function**

Which of the following should replace MISSING to complete this recursive function? Choose only one option. [4]

    i. **return** n + SUM(n $-$ 1)

    ii. **return** n + SUM(n)

    iii. **return** SUM(n)

    iv. **return** SUM(n $-$ 1)

(l) You have a large set of data with n elements, where n will tend to get very large. There is an algorithm that takes at most $0.1n^2 + 10n \log n$ time-steps to complete a task. Which of the following is the worst-case time complexity of this algorithm? Choose only one option. [3]
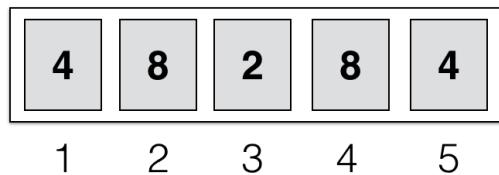
    i. O($n \log n$)

    ii. O(n^2)

    iii. None of the above

PART B

Candidates should answer any **TWO** questions from Part B.

Question 2   This question is about checking if a vector is a palindrome. A vector is a palindrome if it is exactly the same after the order of the elements is reversed (the vector is flipped). An empty vector is a palindrome.

(a) The following vector is an example of a palindrome:

Give an example of a vector that is **not** a palindrome.   [3]

(b) To check if a vector of length n is a palindrome, one can first create an empty stack, and then push the first floor(n/2) values in the vector to it. Here floor(x) means the largest integer smaller than or equal to x. Consider the following piece of pseudocode:

```
1: function MAKESTACK(vector)
2:     new Stack s
3:     n ← LENGTH[vector]
4:     for 1 ≤ i ≤ floor(n/2) do
5:         PUSH[vector[i], s]
6:     end for
7:     return s
8: end function
```

Draw a picture of the returned stack after calling this function MAKESTACK on the example palindrome vector provided in part (a) of this question.   [4]

(c) A linked list can be used to implement the stack in part (b) of this question. Draw the corresponding linked list of the final stack in part (b).   [4]

(d) To complete the algorithm for deciding if a vector is a palindrome, after creating the stack using MAKESTACK, the contents of the stack are compared with the remaining values in the vector, starting from index ceil(n/2) + 1. Here ceil(x) is the smallest integer larger than or equal to x (the ceiling). Consider the following piece of incomplete pseudocode:

```
 1: function ISPALINDROME(vector)
 2:     if LENGTH[vector] = 1 then
 3:         return TRUE
 4:     end if
 5:     new Stack s ← MAKESTACK(vector)
 6:     n ← LENGTH[vector]
 7:     for ceil(n/2) + 1 ≤ i ≤ n do
 8:         if TOP[s] ≠ vector[i] then
 9:             return MISSING1
10:         end if
11:         MISSING2
12:     end for
13:     return TRUE
14: end function
```

This function should return TRUE if the input vector is a palindrome, or return FALSE otherwise. What should go in the place of MISSING1 and MISSING2 to complete this function? [4]

(e) Given an argument for why the worst-case time complexity of implementing ISPALINDROME is $O(n)$ for an input vector of length n. [5]

(f) Another method for deciding if an input vector is a palindrome is through recursion. Consider the following piece of incomplete pseudocode:

```
 1: function ISPALNEW(vector)
 2:     n ← LENGTH[vector]
 3:     return RECPALINDROME(vector, 1, n)
 4: end function
 5: function RECPALINDROME(vector, left, right)
 6:     if right ≤ left then
 7:         return TRUE
 8:     end if
 9:     if vector[left] ≠ vector[right] then
10:         return MISSING1
11:     end if
12:     MISSING2
13: end function
```

The function ISPALNEW should return TRUE if the input vector is a palindrome, and FALSE otherwise - the function calls RECPALINDROME to do this.

   i. What should go in the place MISSING1? [2]
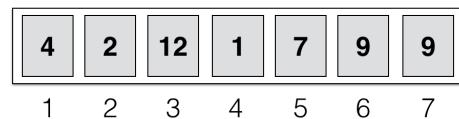
ii. What should go in the place of MISSING2? [4]

(g) Briefly explain why the worst-case time complexity of implementing IsPalNew is also O(n) for an input vector of length n. [4]
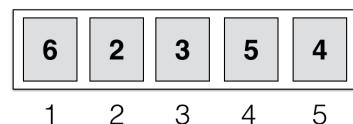
Question 3   This question is about sorting algorithms.

(a) Consider the following vector of integers:

4	2	12	1	7	9	9
1	2	3	4	5	6	7

    i. By hand, work through the standard bubble sort algorithm on this vector, explicitly showing how it changes in the algorithm. You should sort the vector in ascending order so the lowest value is in the first element. [7]

    ii. What is the worst-case time complexity in n of the Bubble Sort algorithm for a vector of length n? [3]

(b) Consider the following vector of integers:

6	2	3	5	4
1	2	3	4	5

    i. By hand, work through the Quicksort algorithm on this vector where the pivot is the mid-point (calculated as the floor of half the sum of the left index plus the right index). Explicitly show how the vector changes in the algorithm. You should sort the vector in ascending order so the lowest value is in the first element. [6]

    ii. Give an example of a worst-case input vector of four elements for Quicksort where the pivot is the mid-point. Each element should store an integer and assume sorting in ascending order. Very briefly explain why it is a worst-case input. [5]

   iii. Briefly explain the role of recursion in the Quicksort algorithm. [4]

(c) Consider the following piece of incomplete JavaScript:

```
1  function swap(array, i, j) {
2    var store = array[i];
3    array[i] = array[j];
4    array[j] = store;
5    return array;
6  }
7
8  function sort(array, right) {
9    if (right <= 0) {
10     return MISSING1;
11   }
12   for (var i = 0; i < right; i++) {
13     if (array[i+1] < array[i]){
14       swap(array, i, i+1);
15     }
16   }
17   return MISSING2;
18 }
19
20 function sorter(array) {
21   return sort(array, array.length - 1);
22 }
```

When completed this code should implement a recursive version of a sorting algorithm on a JavaScript array. The function `sorter` will return the sorted version of the argument `array`.

  i. What should go in the place of `MISSING1`? [2]

  ii. What should go in the place of `MISSING2`? [3]

Question 4   This question is about determining if a number is the sum of two square numbers. A square number is the square of an integer. The number 0 and 1 are both square numbers.

(a) The following vector stores all square numbers from 1^2 to 7^2:

1	4	9	16	25	36	49
1	2	3	4	5	6	7

    i. You are tasked with algorithmically searching this vector to see if it has an element with the value 1. Directly run through the binary search algorithm by hand on this vector, and return the index where the value is stored. Show explicitly each step taken in the algorithm.   [7]

    ii. Briefly explain why the vector above with the value 1 is an example of a worst-case input for the binary search algorithm?   [3]

(b) One method to determine if an integer n is the sum of two square numbers is to try all combinations of the sum of two square numbers. Consider the following piece of incomplete pseudocode:

```
1: function SUMSQUARES(n)
2:     if n = 0 then
3:         return TRUE
4:     end if
5:     for 0 ≤ i ≤ n do
6:         for 0 ≤ j ≤ n do
7:             if MISSING = n then
8:                 return TRUE
9:             end if
10:        end for
11:    end for
12:    return FALSE
13: end function
```

This function, when completed, should return TRUE if n is the sum of two square numbers, and FALSE otherwise.

    i. What should go in the place of MISSING to complete this function?   [3]

    ii. What is the worst-case time complexity of this function?   [4]

(c) Another method to solve the same problem as in part (b) of this question is to first create a vector of all square numbers from 0 to n^2 and then for each value i^2 stored in the vector, search the vector for the value $n - i^2$. Since the values in the vector will be sorted, the binary search algorithm can be used. Consider the following piece of pseudocode:

```
1: function BINARYSQUARES(n)
2:     if n = 0 then
3:         return TRUE
4:     end if
5:     new Vector(n+1) squares
6:     for 0 ≤ j ≤ n do
7:         squares[j] ← j²
8:     end for
9:     for 0 ≤ i ≤ n do
10:        if BINARYSEARCH(squares, n − i²) = TRUE then
11:            return TRUE
12:        end if
13:    end for
14:    return FALSE
15: end function
```

This function will return TRUE if n is the sum of two squares, and FALSE otherwise. It assumes that the binary search algorithm is implemented by a function BINARYSEARCH(vector, x), which returns TRUE if x is stored in the input vector, and FALSE otherwise.

i. Write the pseudocode function called BINARYSEARCH(vector, x), which will implement the binary search algorithm. [9]

ii. Explain why the worst-case time complexity of implementing the function BINARYSQUARES(n) is $O(n\log n)$. [4]

END OF PAPER

.