THIS PAPER IS NOT TO BE REMOVED FROM THE EXAMINATION HALL

UNIVERSITY OF LONDON

CM1035

BSc EXAMINATION

COMPUTER SCIENCE

Algorithms and Data Structures I

Practice Paper

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.

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 positive integer n, what is 3n?

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

i. Boolean

ii. Integer

iii. An empty stack

iv. None of the above.

ii

(b) Consider the following piece of pseudocode:

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

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

1

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

2

iii. What is returned by CHANGE(0)? [1]

1

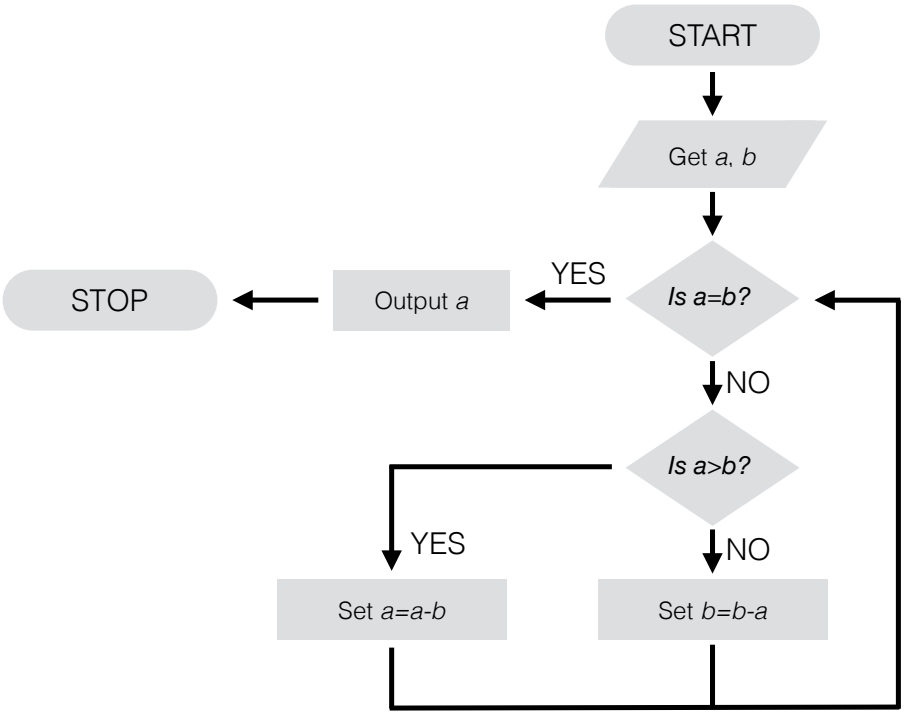
iv. What is the value of x at the end of line 11? [1]

3

v. What is the value of x at the end of line 12? [1]

2

(c) Consider the following flowchart:



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

```
1: function EUCLID(a,b)
2:     while MISSING1 do
3:         if a > b 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 MISSING1? Choose only one option. [3]

i. a $=$ b

ii. a $\neq$ b

iii. a $>$ b

ii

(d) Which of the following describes the operation that adds a new element with the value o to a stack? Choose only one option. [3]

i. dequeue![o]

ii. push![o]

iii. pop![o]

iv. enqueue![o]

ii

(e) Consider the following piece of pseudocode:

```
1: new Queue q
2: ENQUEUE[1,q]
3: ENQUEUE[2×HEAD[q],q]
4: store←HEAD[q]
5: DEQUEUE[q]
6: store←HEAD[q]
```

i. What is the value of *store* at the end of line 4? [2]

1

ii. What is the value of *store* at the end of line 6? [2]

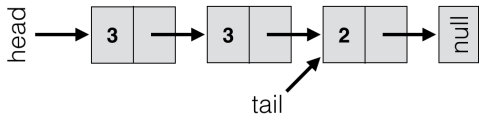
2

(f) Consider the following piece of pseudocode:

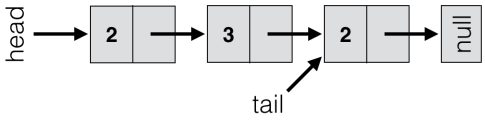
```
1: new Queue q
2: ENQUEUE[2,q]
3: ENQUEUE[3,q]
4: ENQUEUE[HEAD[q],q]
```

In a linked list implementation of this piece of pseudocode, which of the following images describes the linked list after this pseudocode is implemented? Choose only one option. [3]

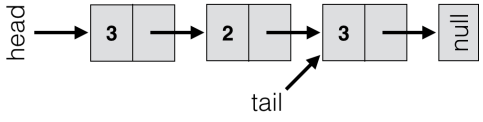
i.



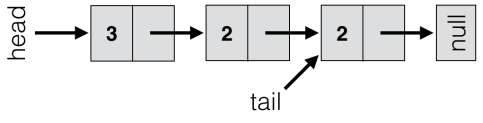
ii.



iii.



iv.



ii

(g) Which of the following describes the difference between a vector and a dynamic array? [3]

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

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

iii. A dynamic array has the SELECT[K] operation, but a vector does not

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

i

(h) 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 $\times$ SUM(n)

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

iii. **return** SUM(n)

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

iv

(i) 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(n)

ii. O(log n)

iii. O(n^2)

iv. O(n log n)

i

(j) Given an unsorted vector of length n, there are two methods to search it. The first method, called **Method A**, is to sort the vector using the insertion sort algorithm and then perform the binary search algorithm. The second method, called **Method B**, is to apply the linear search algorithm. Which of the following describes the worst-case time complexity of **Method A**? Choose only one option. [4]

i. O(n)

ii. O(log n)

iii. O(n^2)

iv. O(n log n)

iii

(k) Of the two methods for searching a vector once in Question 1 part (j), from the point of view of worst-case time complexity, which method should you use? Choose only one option. [2]

i. **Method A**

ii. **Method B**

ii

(l) Which of the following sorting algorithms would improve the worst-case time complexity of **Method A** in Question 1 part (j)? Choose only one option. [2]

i. Bubble sort

ii. Merge sort

iii. Quicksort

ii

PART B

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

Question 2 This question is about the Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, ..., where each new number is obtained by adding the previous two in the sequence. The nth Fibonacci number is denoted $\mathbf{F}_n$, and thus $\mathbf{F}_n = \mathbf{F}_{n-1} + \mathbf{F}_{n-2}$. By convention the first two Fibonacci numbers are $\mathbf{F}_0 = 0$ and $\mathbf{F}_1 = 1$.

(a) Consider the following piece of pseudocode:

```
function FIBONACCI(n)
    a ← 0
    b ← 1
    if n ≠ 0 then
        for 1 ≤ i ≤ n do
            b ← a + b
            a ← b − a
        end for
    end if
    return a
end function
```

Complete the following table for the first five values of n. In the second column the output of the function FIBONACCI(n) should be entered, and in the third column enter the final value of b achieved within the body of the function.

n	FIBONACCI(n)	b
0	0	1
1	1	1
2	1	2
3	2	3
4	3	5

[6]

(b) Briefly explain why the number of operations in a standard implementation of FIBONACCI(n) is in the 'Big O' class $O(\mathrm{n})$ for the input n. [4]

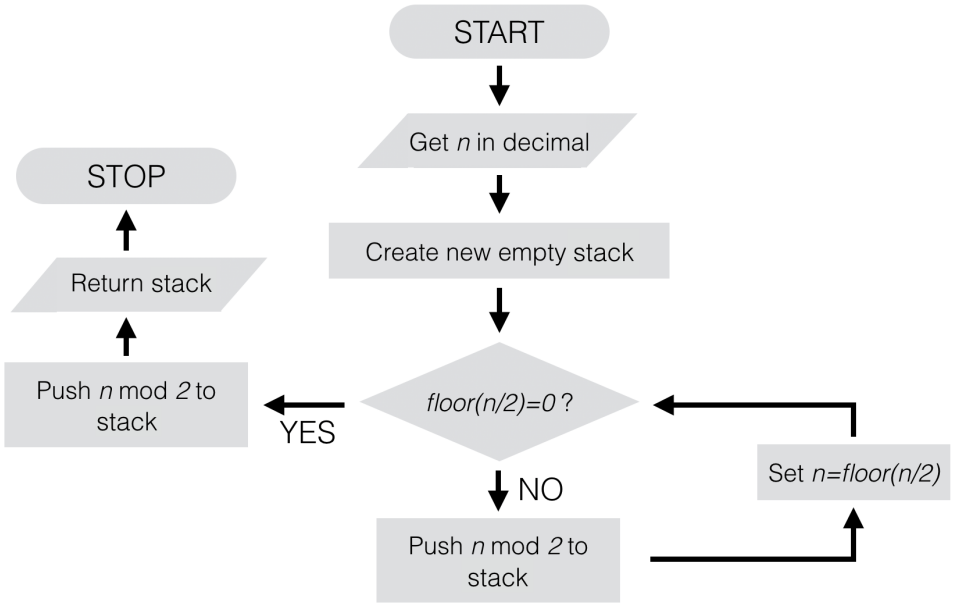
There are n iterations due to the for loop, in each loop there are a constant number of operations, so it is $O(n)$.

(c) Write a new pseudocode function called RECFIBONACCI(n), which takes the same input parameters and returns the same output as FIBONACCI(n)

in part (a) of this question (assume n is a non-negative integer). This new function should not use any form of iteration. [6]

```
function RECFIBONACCI(n)
    if n < 2 then
        return n
    end if
    return RECFIBONACCI(n-1) + RECFIBONACCI(n-2)
end function
```

(d) Consider the following flowchart. This algorithm will produce a stack that gives the binary representation of the number n. In this flowchart, $floor(y)$ gives the largest integer smaller or equal to y.



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

```
function BINARY(n)
    new Stack s
    while MISSING1 do
        PUSH[MISSING2, s]
        n ← floor(n/2)
    end while
    MISSING3
    return s
```

end function

What should go in the place of MISSING1, MISSING2, and MISSING3 to complete this pseudocode? [6]

MISSING1 = floor(n/2) $\neq$ 0
MISSING2 = n mod 2
MISSING3 = PUSH[n mod 2,s]

(e) Recall that the function FIBONACCI(n) in part (a) of this question calculates the Fibonacci number $\mathbf{F}_n$. An equivalent way of defining the Fibonacci number $\mathbf{F}_n$ for $n > 2$ is as the number of binary strings (or bit-strings) of length n $-$ 2 without consecutive 1's in the string. An example of a binary string without consecutive 1's is 1010. For $0 \leq n \leq 2$, we just note the regular values of $\mathbf{F}_n$. The following function NOTCONSECUTIVE(stack) takes a stack as input, and will return a Boolean: it will return TRUE if the stack has elements with no consecutive 1's, and FALSE otherwise.

```
function NOTCONSECUTIVE(stack)
    new Stack second
    while EMPTY[stack] = FALSE do
        PUSH[TOP[stack], second]
        POP[stack]
        if (TOP[stack] = 1) ∧ (TOP[second] = 1) then
            return FALSE
        end if
    end while
    return TRUE
end function
```

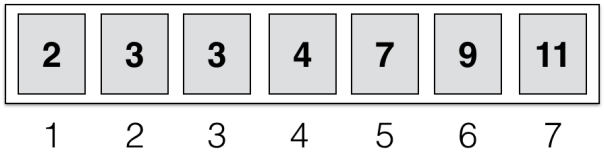
A new algorithm, called Algorithm **S** for calculating $\mathbf{F}_n$ is the following: for $n > 2$, one by one, generate all binary strings of length $n - 2$ and check if there are any consecutive 1's, and use this information to count the number of strings without consecutive 1's; for $0 \leq n \leq 2$, the algorithm will just output the Fibonacci numbers for those values of n.

Write a pseudocode function called ALGORITHMS(n) that implements the Algorithm **S**. Your function takes as the integer n for $\mathbf{F}_n$, and returns an output that is equal to $\mathbf{F}_n$. HINT: you may assume that functions BINARY(n) and NOTCONSECUTIVE(n) are already defined (and completed) and you may call them within your function, if you need them. Also, note that the binary string of all-ones of length n corresponds to the number $2^n - 1$. [8]

```
function AlgorithmS(n)
    if n ≤ 1 then
        return n
    end if
    if n = 2 then
        return 1
    end if
    count ← 0
    for 2^(n−2) ≤ i ≤ 2^(n−1) − 1 do
        stack ← Binary(n)
        if NotConsecutive(stack) = TRUE then
            count ← count + 1
        end if
    end for
    return count
end function
```

Question 3 This question is about searching and sorting elements of vectors.

(a) Consider the following array of integers:



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. Show explicitly each step taken in the algorithm. [7]

Initial mid-point is 4 (left pointer is 1 and right pointer is 7), which stores 4, it is not equal to 1, so update right pointer to be 3
New mid-point is 2 (left is 1, right is 3), which stores 3, update right pointer to be 1
New mid-point is 1 (left = right = 1), which stores 2, update right point to be 0
Return "not found" as right is less than left

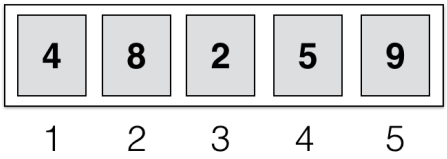
ii. What is the worst-case time complexity in n of the binary search algorithm for a sorted vector of length n? [3]

O(log n)

iii. Very briefly explain why the vector above and value being searched is an example of a worst-case input to binary search. [2]

Since the value is not in the vector, we will need to keep reducing the number of elements we need to consider by half until we get down to one element left. This is the most number of times the vector is halved.

(b) Consider the following vector of integers:



i. Implement 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]

First pass:
48259
42859
42589
Second pass:
42589
24589
Sorted

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

$O(n^2)$

(c) Consider the following piece of JavaScript:

```
1  function goldSearch(arr, el) {
2    var left = 0;
3    var right = arr.length - 1;
4    while (left<=right) {
5      var fir = left + Math.floor((right - left) / 3);
6      var sec = left + Math.floor(2 * (right - left) / 3);
7      if ((arr[fir] == el) || (arr[sec] == el)) {
8        if (arr[fir] == el) {
9          return fir;
10       } else {
11         return sec;
12       }
13     } else if (el < arr[fir]) {
14       right = fir - 1;
15     } else if (arr[sec] < el) {
16       left = sec + 1;
17     } else {
18       left = fir + 1;
19       right = sec - 1;
20     }
21   }
22   return false;
23 }
```

A new searching algorithm is proposed, called the London Search algorithm, which works for sorted vectors and arrays of numbers where the numbers go from lowest first to highest last. The JavaScript code above is an implementation of this algorithm.

i. In JavaScript, write a function called `isThere(arr, el)`, which takes an array `arr` and value `el` as input parameters and returns `true` or `false` if there is an element with the value `el` or not, respectively. You may assume that `goldSearch(arr, el)` is already defined and so you can call it within your code. [4]

```
function isThere(arr, el) {
if (!goldSearch(arr, el)) {
return false;
}
return true;
}
```

ii. From the point of view of worst-case time complexity, briefly explain why this algorithm will not be better than the binary search algorithm. [4]

The algorithm divides up the array into three parts, so in each iteration we will look at a particular third of the array. This is like binary search, so the time complexity will be O(log n) as well, since we do not need to worry about the base.

Question 4 This question is about stacks, linked lists and the factorial of a number. Recall that the factorial of an integer n is written $n!$ and is defined as $n! = n \times (n-1) \times (n-2) \times ... \times 2 \times 1$, and $0! = 1$.

(a) Consider the following piece of pseudocode:

```
function FacStack(n)
    new Stack s
    PUSH[1,s]
    if n = 0 then
        return s
    end if
    for 1 ≤ i ≤ n do
        x ← TOP[s]
        PUSH[i × x, s]
    end for
    return s
end function
```

i. This function takes an integer n as its input and returns a stack. For the function call FacStack(4), draw the stack that is returned. Clearly show the values stored in the elements, and point out the top. [4]

top$\rightarrow 24, 6, 2, 1, 1$

ii. To implement the stack created by the function FacStack(n), we can use an array. If we initially start with an array of ten elements, how many new arrays will need to be created to implement the stack for the function call FacStack(4)? [2]

We will not need to create any new arrays.

iii. Another way of implementing the stack created by FacStack(n) is to use a linked list. Draw the linked list that implements the stack returned by FacStack(3). Clearly show which element is the top with a pointer. [4]

top$\rightarrow 6| \rightarrow 2| \rightarrow 1| \rightarrow 1| \rightarrow$ *null*

(b) Consider the following piece of JavaScript:

```
1 function LLNode(data) {
2   this.data = data;
3   this.next = null;
4 }
5 var head = new LLNode(5);
6 console.log(head.next);
```

```
7 console.log(head.data);
```

This is a function that creates an object that creates a node of a linked list with the value `data` stored in the node. On line 5, the variable `head` will then store the object corresponding to the node.

i. What is printed to the console for line 6 of this code? [1]

null

ii. What is printed to the console for line 7 of this code? [1]

5

iii. In another piece of JavaScript code, construct the final linked list that was drawn in part a.(iii) of this question. Assume that the function `LLNode(data)` is already defined and use it to construct the linked list. Store the linked list as an object in the variable `head` (assume that head is not defined at the beginning of your code). [5]

```
1 var head = new LLNode(6);
2 head.next = new LLNode(2);
3 head.next.next = new LLNode(1);
4 head.next.next.next = new LLNode(1);
```

(c) Consider the following piece of JavaScript:

```
1  function Stack() {
2    this.head = null;
3    this.push = function(o) {
4      var node = new LLNode(o);
5      node.next = this.head;
6      this.head = node;
7    };
8    this.top = function() {
9      if (this.head === null) {
10       return null;
11     }
12     return this.head.data;
13   };
14   this.pop = function() {
15     if (this.head === null) {
16       return "Stack underflow!";
17     }
18     this.head = this.head.next;
19   };
20   this.isEmpty = function() {
```

```
21          return (this.head === null);
22      };
23  }
24  var stack = new Stack();
25  stack.push(1);
26  stack.push(1 * stack.top());
27  console.log(stack.top());
```

This is a function that creates an object that implements a stack using the function `LLNode(data)`.

i. What is printed to the console in line 27 of this code? [2]

1

ii. Write a JavaScript function called `facStack(n)`, which implements the pseudocode function FACSTACK(n) in part (a) of this question. You may assume the function `Stack()` is already defined, and you can therefore call it. To get full marks, use need to use the methods of the stack object generated by the function `Stack()`. [7]

```
1  function facStack(n){
2  var stack = new Stack();
3  stack.push(1);
4  if (n===0){
5  return stack
6  }
7  for (var i = 1; i <= n; i++) {
8  var x = stack.top();
9  stack.push(i * x);
10 }
11 return stack;
12 }
```

(d) Note that the methods of the objected generated by the function `Stack()` can all be implemented with a constant number of lines of code. Using this observation, what is the time complexity of implementing the function `facStack(n)`in the input `n`? Very briefly explain your answer. [4]

The time complexity is O(n) since there is a for loop with n iterations. In each iteration there are only a constant number of operations, and each method requires only a constant number of operations.

END OF PAPER

.