Question 1

[20 marks]

1 Write a Java program given the following specification and provide comments which explain how your algorithm works.

Problem Statement
The task is to take in a credit card number, and find out if it is a valid credit card number or not. Credit card numbers follow an algorithm called Luhn's algorithm.

The formula verifies a number against its check digit, which is the last digit. This number must pass the following test:

1. From the rightmost digit, which is the check digit, and moving left, double the value of every second digit. If the result of this doubling operation is greater than 9 (e.g., 8 × 2 = 16), then add the digits together (e.g., 16: 1 + 6 = 7, 18: 1 + 8 = 9).
2. Take the sum of all the digits.
3. If the total modulo 10 is equal to 0 (if the total ends in zero) then the number is valid according to the Luhn formula; else it is not valid.

Assume an example of an account number "7992739871" that will have a check digit added, making it of the form 7992739871x. The sum of all the digits, processed as per steps 1 and 2, is 67+x. Thus x must be 3 to bring the total to be modulo 10 = 0. If x is not 3, then this is not a valid credit card number.

Input Format
An *n*-digit credit card number, where the last digit is the check digit.

Output Format
Output "VALID" if it is a valid credit card number and "INVALID" if it is not.

Constraints
4<=*n*<=30

Sample Input
4539682995824395

Sample Output
VALID

```
import java.util.Scanner;

public class Q1 {
    public static void main (String args[]) {
        Scanner sc = new Scanner (System.in);
        String inputLine = sc.nextLine();
```

CS210-2018-January

```
        sc.close();
        System.out.println(LuhnAlgorithm(inputLine));
    }

    // Check whether your bank card number is valid.
    public static String LuhnAlgorithm(String creditCardNumbers) {
        StringBuffer sb = new StringBuffer(creditCardNumbers);
        // From rightmost digit -> reverse the String
        String reversedString = sb.reverse().toString();
        // STEP 1: double every second digit and take the sum.
        int doubleSumOfEvenDigit = 0;
        for(int i = 1; i < reversedString.length() ; i = i + 2 ) {
            int currentDigit =
            Character.getNumericValue(reversedString.charAt(i));
            currentDigit = currentDigit * 2;
            if(currentDigit >= 10) currentDigit = currentDigit - 9;
            doubleSumOfEvenDigit = doubleSumOfEvenDigit + currentDigit;
        }
        // STEP 2: take the sum of the rest digit.
        int sumOfOddDigit = 0;
        for(int i = 0; i < reversedString.length() ; i = i + 2 ) {
            int currentDigit =
            Character.getNumericValue(reversedString.charAt(i));
            sumOfOddDigit = sumOfOddDigit + currentDigit;
        }
        // STEP 3: Check whether total modulo 10 is equal to 0.
        int moduloSum = sumOfOddDigit + doubleSumOfEvenDigit;
        if(moduloSum % 10 == 0) {
            return "VALID";
        }
        else {
            return "INVALID";
        }
    }
}
```

CS210-2018-January

Question 2

[20 marks]

2 Write a Java program given the following specification and provide comments which explain how your algorithm works. Estimate the **Big O complexity** of your program and explain your reasoning clearly.

Problem Statement
The goal is to read in a number *N* and output the distance between the prime number that precedes it, and the prime that follows it. If the number itself happens to be prime, then output the distance to the subsequent prime. For example, if *N* is 7, then output 4, because the next prime is 11, which is 4 away.

Input Format
An integer *N*.

Output Format
The distance between the preceding and subsequent prime number at *N*.

Constraints
$2 \leq N \leq 1000$

Sample Input
10

Sample Output
4

```
import java.util.Scanner;

public class Q2 {
    public static void main (String args[]) {
        Scanner sc = new Scanner(System.in);
        int inputNum = sc.nextInt(); // input a number
        sc.close();

        int smallerPrime, biggerPrime;
        int distance;
        // If the number is Prime
        if(isPrime(inputNum)) {
            smallerPrime = inputNum;
            biggerPrime = findBiggerPrime(inputNum);
        }
        // If the number is not Prime
        else {
            smallerPrime = findSmallerPrime(inputNum);
            biggerPrime = findBiggerPrime(inputNum);
        }
```

```
        // Print out the distance
        distance = biggerPrime - smallerPrime;
        System.out.println(distance);
            }

    public static boolean isPrime (int number){
        if (number <= 1) return false;

        for (int i = 2; i< number; i++) { // O(n)
            if(number % i == 0) return false;
        }

        return true;
    }

    // Method: Get the Prime Number that precedes it
    public static int findSmallerPrime (int number) {
        int smallerPrime = number - 1;
        while(!isPrime(smallerPrime)) {
            smallerPrime--;
        }
        return smallerPrime;
    }

    // Method: Get the Prime Number that follows it
    public static int findBiggerPrime (int number) {
        int biggerPrime = number + 1;
        while(!isPrime(biggerPrime)) {
            biggerPrime++;
        }
        return biggerPrime;
    }
}
```

[Big O Complexity] - O(n)

In this program, I use three methods - findSmallerPrime, findBiggerPrime and isPrime.
The input variable is "number" a
1) findSmallerPrime/ findBiggerPrime - Time Complexity O(1)
2) isPrime - Time Complexity O(n) in the for loop is related with the input variable "number"
When we call 1), we must use 2) to check whether the input number is prime.

So the time complexity of the program is O(1) * O(n) = O(n)

CS210-2018-January

*Question 3

[20 marks]

3 Write a Java method that takes in a Linked List object (double ended and doubly-linked, with each link containing an `int`) and deletes any Link whose `int` value is less than 100. The method then returns the Linked List object. Provide comments which explain how your algorithm works. Estimate the **Big O complexity** of your program and explain your reasoning clearly.

```
class Node {
    int data;
    Node prev;
    Node next;

    public Node(int data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}

class DoublyLinkedList {
    Node head;
    Node tail;

    // Method to delete links with value less than 100
    public DoublyLinkedList deleteLinksLessThan100(DoublyLinkedList list) {
        Node currentNode = list.head;
        while (currentNode != null) {
            // Save the next node before deletion
            Node nextNode = currentNode.next;

            // Delete the link if its value is less than 100
            if (currentNode.data < 100) {
                // Adjust the previous and next pointers
                if (currentNode.prev != null) {
                    currentNode.prev.next = currentNode.next;
                } else {
                    list.head = currentNode.next;
                }

                if (currentNode.next != null) {
                    currentNode.next.prev = currentNode.prev;
```

CS210-2018-January

```
            } else {
                list.tail = currentNode.prev;
            }
        }

        // Move to the next node
        currentNode = nextNode;
    }
    // Return the modified doubly-linked list.
    return list;
  }
}
```

Time Complexity - O(N)

N is the number of nodes in the input doubly-linked list.
If you want to traverse the List, you should iterate N times.

CS210-2018-January

Question 4

Question a

[20 marks]

4 a) Identify the output that the following Java code produces and explain your reasoning clearly. [7 marks]

```
public class Recursion{

    public static void main(String[] args){
        System.out.println(mystery("Start"));
    }

    public static String mystery(String input){
        if(input.length()>10){
            return "Complete";
        }
        System.out.println("Running...");
        return (mystery(input+"x")+"OK");
    }
}
```

The program runs main function first, it will call `mystery("Start")`

1)mystery("Start").

length = 5, 5 < 10 => skip if statement

print out **"Running..."**, **then change line.**

return(mystery("Start" + "x")+"OK")

2) mystery("Startx").

length = 6, 6 < 10 => skip if statement

print out **"Running..."**, **then change line.**

return(mystery("Startx" + "x")+"OK")

3) mystery("Startxx").

length = 7, 7 < 10 => skip if statement

print out **"Running..."**, **then change line.**

return(mystery("Startxx" + "x")+"OK")

4) mystery("Startxxx").

length = 8, 8 < 10 => skip if statement

print out **"Running..."**, **then change line.**

return(mystery("Startxxx" + "x")+"OK")

CS210-2018-January

5) mystery(“Startxxxx”).
length = 9, 9 < 10 => skip if statement
print out “Running...”, then change line.
return(mystery(“Startxxxx” + “x”)+"OK")

6) mystery(“Startxxxxx”).
length = 10, 10 = 10 => skip if statement
print out “Running...”, then change line.
return(mystery(“Startxxxxx” + “x”)+"OK")

7) mystery(“Startxxxxxx”).
length = 11, 11 > 10 => run if statement
Return “Complete”

8) mystery(“Startxxxxx”) = (“Startxxxxxx”)+"OK" = “CompleteOK”
Get “CompleteOK”

9) mystery(“Startxxxx”) = (“Startxxxxx”)+"OK" = “CompleteOKOK”
Get “CompleteOKOK”

10) mystery(“Startxxx”) = (“Startxxxx”)+"OK" = “CompleteOKOKOK”
Get “CompleteOKOKOK”

11) mystery(“Startxx”) = (“Startxxx”)+"OK" = “CompleteOKOKOKOK”
Get “CompleteOKOKOKOK”

12) mystery(“Startx”) = (“Startxx”)+"OK" = “CompleteOKOKOKOKOK”
Get “CompleteOKOKOKOKOK”

13) mystery(“Start”) = (“Startx”)+"OK" = “CompleteOKOKOKOKOKOK”
Get “CompleteOKOKOKOKOKOK”
Finally, print out “CompleteOKOKOKOKOKOK”, then change line.

Therefore, the Java Program outputs

```
Running...
Running...
Running...
Running...
Running...
Running...
CompleteOKOKOKOKOKOK
```

when it runs.

CS210-2018-January

Question b

b) Identify the output that the following Java code produces and explain your reasoning clearly. [7 marks]

```
public class BitManipulation{

    public static void main(String[] args){
        System.out.println((((7&19)|23))<<2);
    }
}
```

The program will print out the equation

(((7&19)|23)<<2)

Step 1: 7 & 19

$(7)_{10}$	$=(00000111)_2$	
$(19)_{10}$	$=(00010011)_2$	&
	$`(00000011)_2$	$= (3)_{10}$

Step 2: 3 | 23

$(3)_{10}$	$=(00000011)_2$	
$(23)_{10}$	$=(00010111)_2$	|
	$`(00010111)_2$	$= (23)_{10}$

Step 3: 23 << 2

$(00010111)_2 << 3 = (01011100)_2 = (92)_{10}$

Therefore, the Java Program outputs 92 when it runs.

*Question c

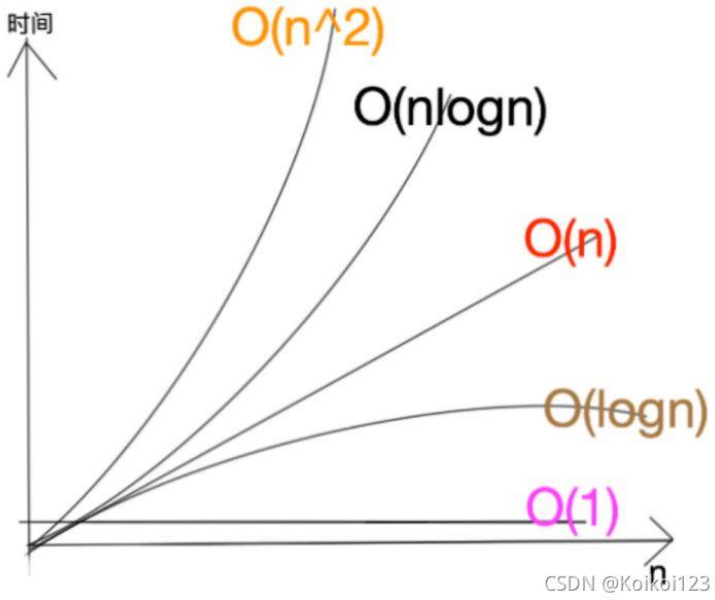
c) Show how the following numbers would be sorted by mergesort. [6 marks]
State the **Big O complexity** of mergesort and explain why it is more efficient than bubble sort.

84 25 83 96 36 10 57 29

[84, 25, 83, 96, 36, 10, 57, 29]

- Divide the list: [84], [25], [83], [96], [36], [10], [57], [29]
- Merge pairs and sort: [25, 84], [83, 96], [10, 36], [29, 57]
- Merge sublists and sort: [25, 83, 84, 96], [10, 29, 36, 57]
- Merge the two sorted sublists: [10, 25, 29, 36, 57, 83, 84, 96]

The Big O Complexity of mergesort is O(n * log(n))
The Big O Complexity of bubblesort is $O(n^2)$
As we can see in the graph, if n is the same value, $O(n^2) > O(n * log(n))$.
Therefore, mergesort use less time, it is more efficient than bubblesort.



CS210-2018-January