# Question 1

文本, 信件

描述已自动生成

**import** java.util.Scanner;

**public** **class** Q1 {

**public** **static** **void** main (String args[]) {

Scanner sc = **new** Scanner (System.***in***);

String inputLine = sc.nextLine();

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";

}

}

}

# Question 2

文本, 信件

描述已自动生成

**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)**

# \*Question 3

文本

描述已自动生成

**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;

} **else** {

list.tail = currentNode.prev;

}

}

// Move to the next node

currentNode = nextNode;

}

// Return the modified doubly-linked list.

**return** list;

}

}

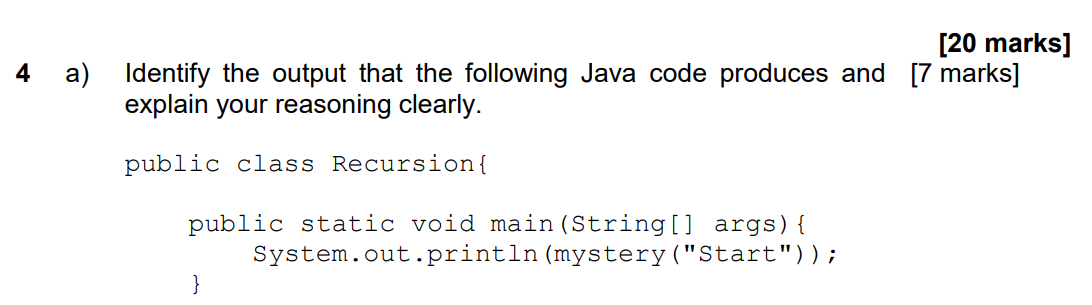
**Time Complexity - O(N)**

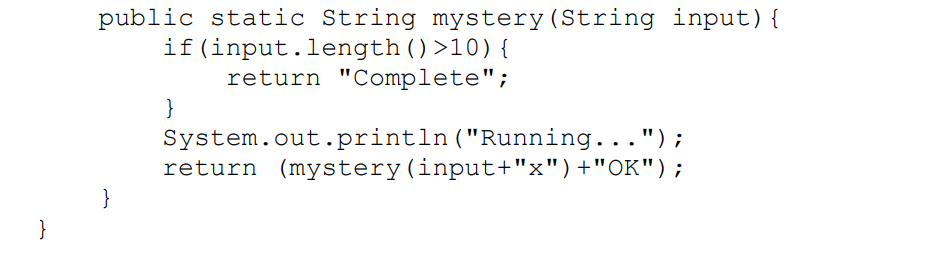
N is the number of nodes in the input doubly-linked list.

If you want to traverse the List, you should iterate N times.

# Question 4

## Question a





**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")**

**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.**

## Question b

文本

描述已自动生成

**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

图形用户界面, 文本, 应用程序

描述已自动生成

**[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(n2)**

**As we can see in the graph, if n is the same value, O(n2) > O(n \* log(n)).**

**Therefore, mergesort use less time, it is more efficient than bubblesort.**

