C-DAC Mumbai

Date 25/09/2024

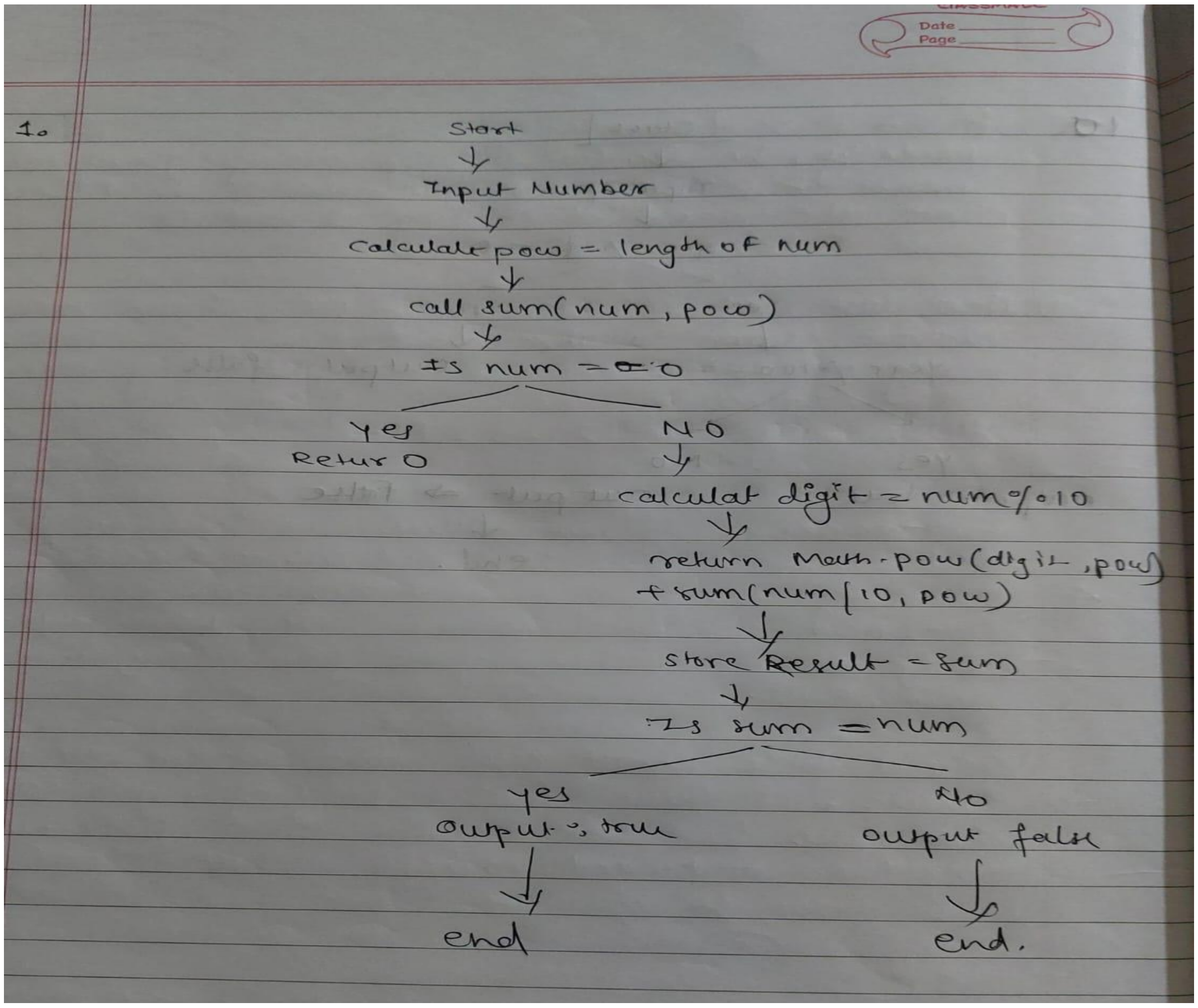
Subject: Algorithm and Data Structure
Assignment 1

1**. Armstrong Number**
Problem: Write a Java program to check if a given number is an Armstrong number.

Test Cases:

Input: 153
Output: true
Input: 123
Output: false

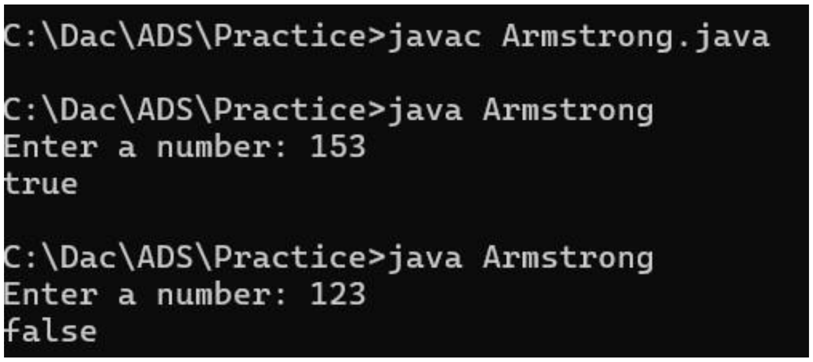
FLOWCHART:



PROGRAM

```
//Amstrong Number
import java.util.Scanner;

class Armstrong{
        static int sum(int num, int pow){
                if(num == 0)
                        return 0;
                else{
                        int digit = num % 10;
                return (int) Math.pow(digit,pow) + sum(num / 10, pow);
                }
        }

        public static void main(String args[]){
                Scanner sc = new Scanner(System.in);

                System.out.print("Enter a number: ");
                int num = sc.nextInt();

                int pow = String.valueOf(num).length();

                int sum = sum(num, pow);

                if (sum == num)
                        System.out.println(true);
                else
                        System.out.println(false);

                sc.close();
        }
}
```

OUTPUT:



```
C:\Dac\ADS\Practice>javac Armstrong.java

C:\Dac\ADS\Practice>java Armstrong
Enter a number: 153
true

C:\Dac\ADS\Practice>java Armstrong
Enter a number: 123
false
```

EXPLANATION:

An Armstrong number is a number equal to the sum of its own digits raised to the power of the number of digits.

For example, for 153, 13+53+33=1531^3 + 5^3 + 3^3 = 15313+53+33=153, making it an Armstrong number.
Using recursion, we break down the number into its last digit, compute the power, and repeat the process for the remaining digits.

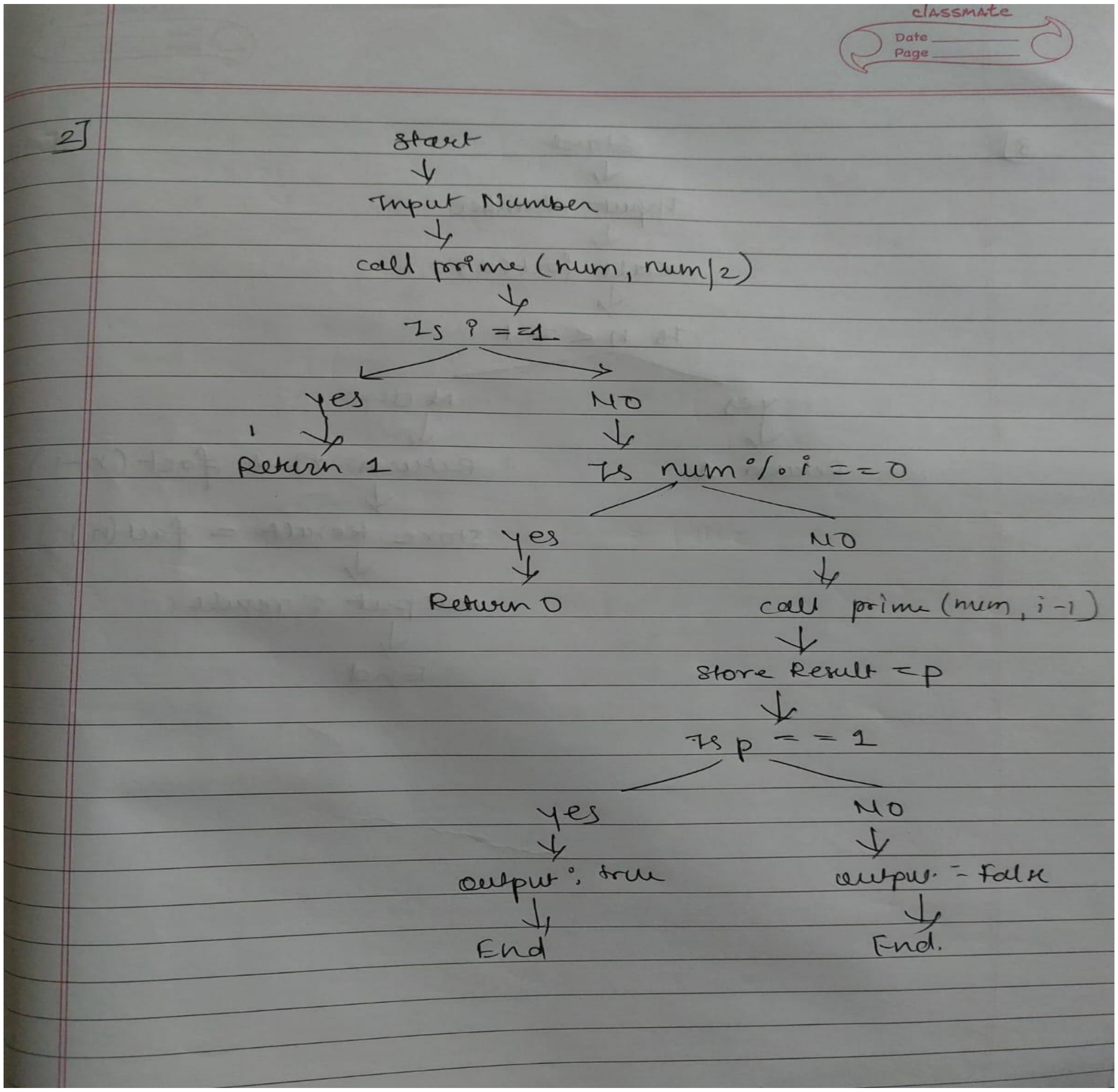
Time Complexity: O(n)
Space Complexity: O(n)

2. Prime Number
Problem: Write a Java program to check if a given number is prime.

Test Cases:

Input: 29
Output: true
Input: 15
Output: false

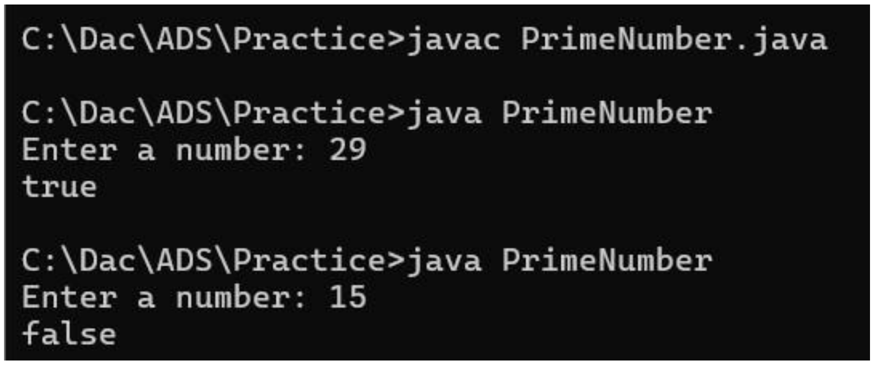
FLOWCHART:



PROGRAM:

```
//PrimeNumber

import java.util.Scanner;

class PrimeNumber{

        static int prime(int num, int i){
                if(i==1)
                        return 1;
                if(num%i==0)
                        return 0;
                        return prime(num, i-1);
        }

        public static void main(String args[]){
                System.out.print("Enter a number: ");
                Scanner sc = new Scanner(System.in);
                int num = sc.nextInt();

                int p = prime(num, num/2);

                if(p==1)
                        System.out.println("true");
                else
                        System.out.println("false");

     sc.close();
        }
}
```

OUTPUT:



```
C:\Dac\ADS\Practice>javac PrimeNumber.java

C:\Dac\ADS\Practice>java PrimeNumber
Enter a number: 29
true

C:\Dac\ADS\Practice>java PrimeNumber
Enter a number: 15
false
```

EXPLANATION:

A prime number is only divisible by 1 and itself.
Here, Base case is (i==1) :
If i is 1, then no divisors were found for n (other than 1 and itself), so the number is prime. The function returns 1, indicating the number is prime.

Divisibility Check:
If num % i == 0, it means num is divisible by i, so it is not a prime number, and the function returns 0.
Recursion Case:
The function calls itself with i - 1 to check divisibility by the next smaller number until it reaches i == 1.

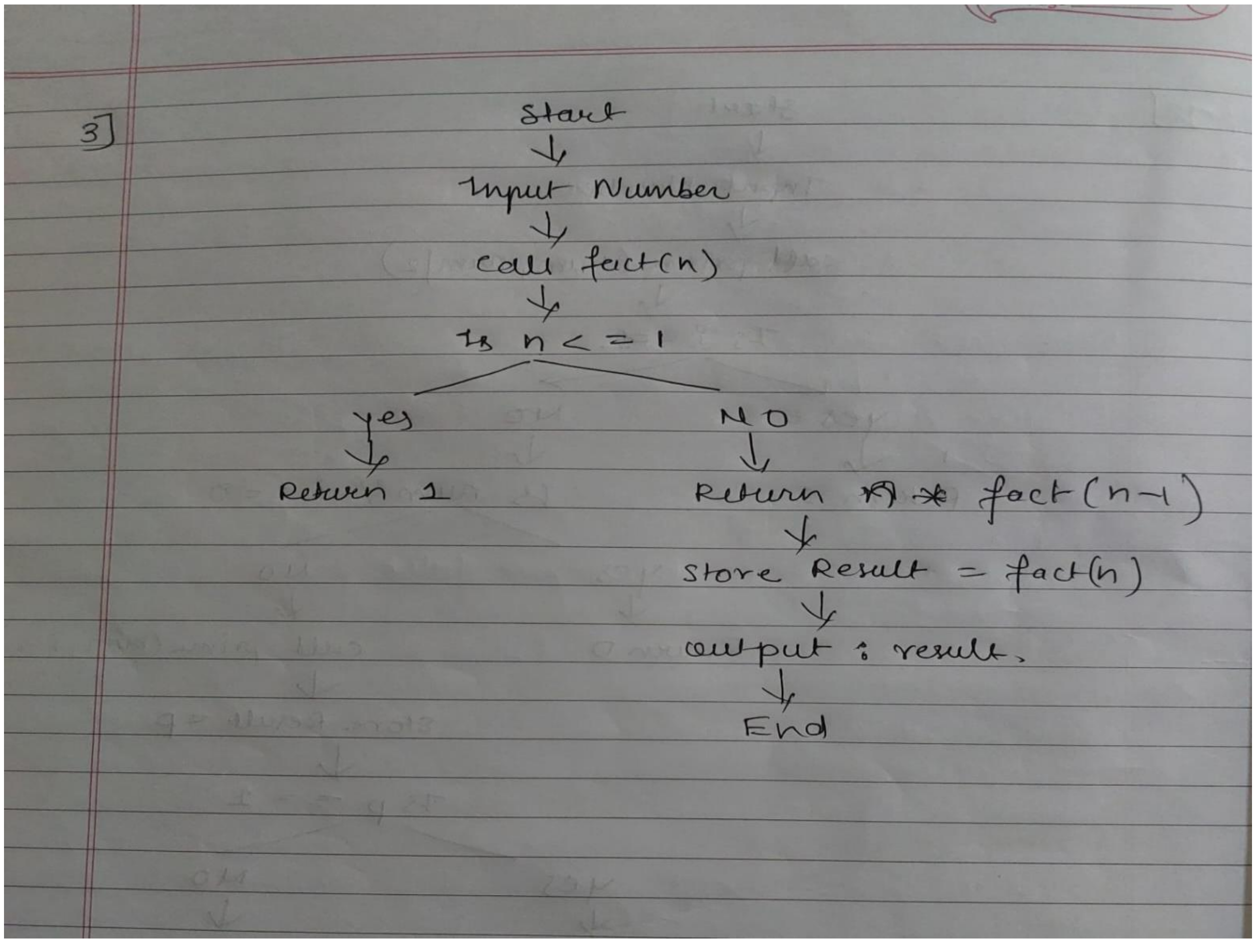
Time Complexity: O(n)
Space Complexity: O(n)

3. Factorial
Problem: Write a Java program to compute the factorial of a given number.

Test Cases:

Input: 5
Output: 120
Input: 0
Output: 1

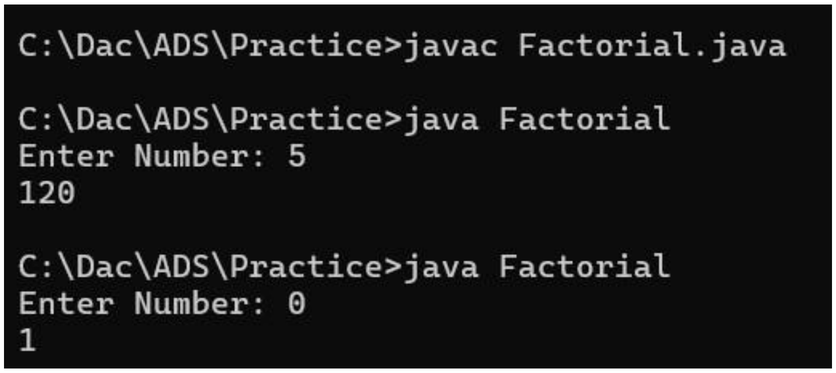
FLOWCHART:



PROGRAM:

```
//Print Factorial
import java.util.Scanner;
class Factorial{

	static int fact(int n){
		if(n <= 1)
			return 1;
		else
		return n*fact(n-1);
	}

	public static void main(String arg[]){
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter Number: ");
		 int n = sc.nextInt();

		 System.out.println(fact(n));
	}
}
```

OUTPUT:



```
C:\Dac\ADS\Practice>javac Factorial.java

C:\Dac\ADS\Practice>java Factorial
Enter Number: 5
120

C:\Dac\ADS\Practice>java Factorial
Enter Number: 0
1
```

EXPLANATION:
Base Case:
If n is less than or equal to 1, the function returns 1.

Recursive Case:
For n>1n > 1n>1, the function recursively calls itself with n−1n - 1n−1, multiplying the result by n. This builds up the factorial by breaking the problem down into smaller subproblems.

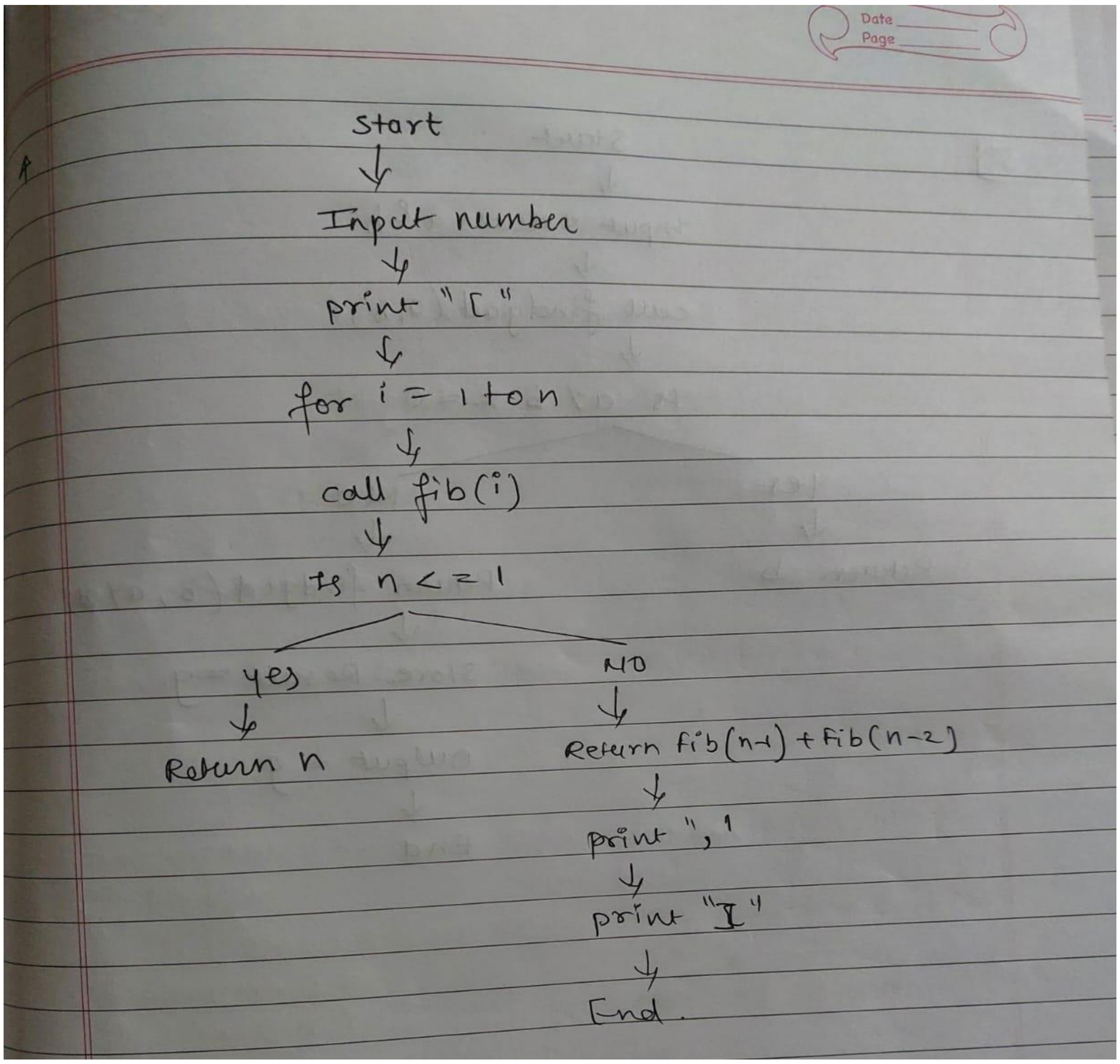
Time Complexity: O(n)
Space Complexity: O(n)

4. Fibonacci Series
Problem: Write a Java program to print the first n numbers in the Fibonacci series.

Test Cases:

Input: n = 5
Output: [0, 1, 1, 2, 3]
Input: n = 8
Output: [0, 1, 1, 2, 3, 5, 8, 13]

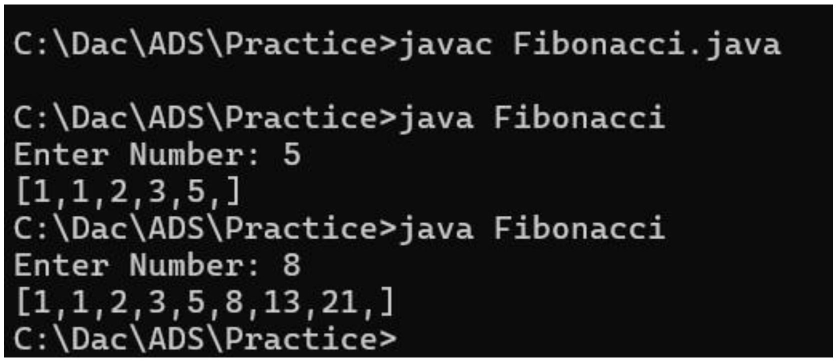
FLOWCHART:



PROGRAM:

```
import java.util.Scanner;

class Fibonacci{
	static int fib(int n){
		if(n<=1)
			return n;
			return fib(n-1)+fib(n-2);
	}

	public static void main(String args[]){
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter Number: ");

		int n = sc.nextInt();

		System.out.print("[");
		for(int i=1; i<=n; i++){
			System.out.print(fib(i)+",");
		}
		System.out.print("]");
	}
}
```

OUTPUT:



```
C:\Dac\ADS\Practice>javac Fibonacci.java

C:\Dac\ADS\Practice>java Fibonacci
Enter Number: 5
[1,1,2,3,5,]
C:\Dac\ADS\Practice>java Fibonacci
Enter Number: 8
[1,1,2,3,5,8,13,21,]
C:\Dac\ADS\Practice>
```

EXPLANATION:
Base Case:
If (n<=1), the function returns n. This is because:
Recursive Case:
If n>1n > 1n>1, the function calculates fib(n) as the sum of the two previous Fibonacci numbers: fib(n-1) and fib(n-2).

Fibonacci Sequence: The for loop generates the first n Fibonacci numbers. It starts at i = 0 to include fib(0) = 0 and continues up to n - 1.

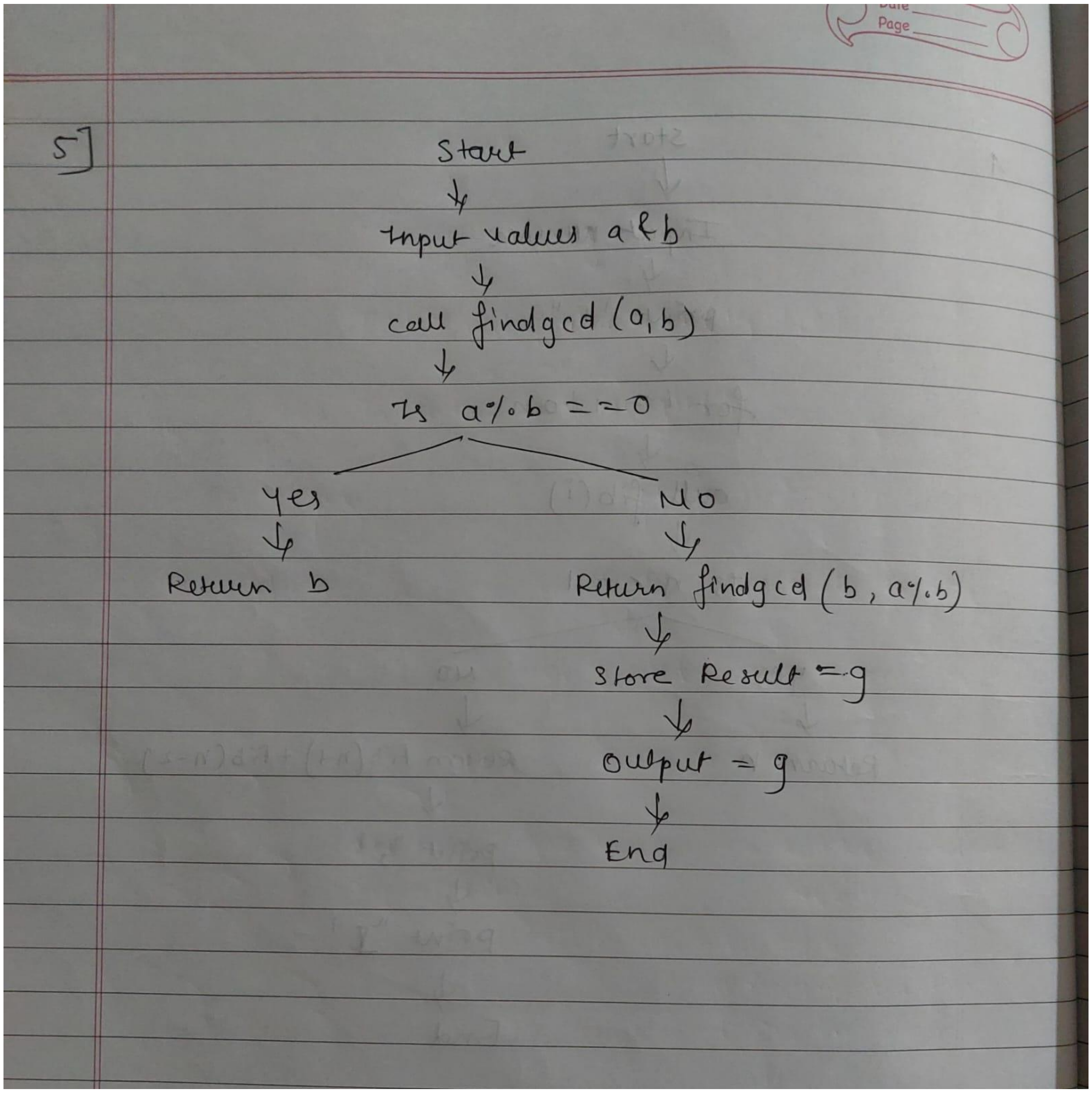
Time Complexity: O(n)
Space Complexity: O(1)

5. Find GCD
Problem: Write a Java program to find the Greatest Common Divisor (GCD) of two numbers.

Test Cases:

Input: a = 54, b = 24
Output: 6
Input: a = 17, b = 13
Output: 1

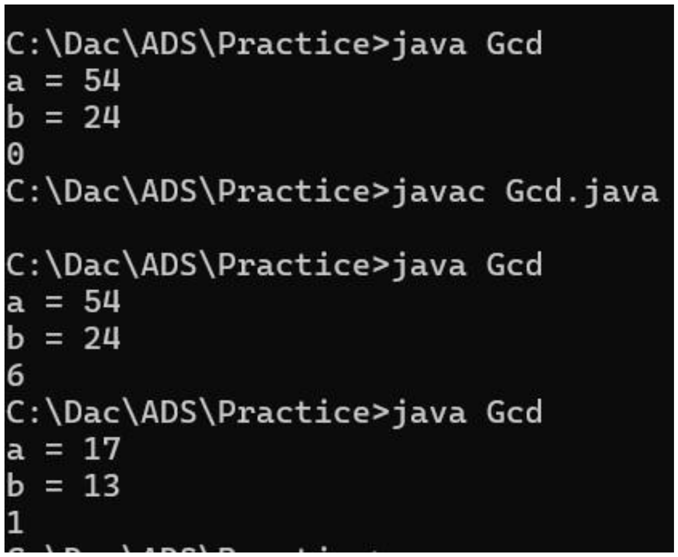
FLOWCHART:



PROGRAM:

```
import java.util.Scanner;
public class Gcd{

        static int findgcd(int a, int b){
                if(a%b==0)
                        return b;
                        return findgcd(b,a%b);
        }

        public static void main(String args[]){
                Scanner sc = new Scanner(System.in);
                System.out.print("a = ");
                int a = sc.nextInt();
                System.out.print("b = ");
                int b = sc.nextInt();
                int g = findgcd(a,b);
                System.out.print(g);
        }
}
```

OUTPUT:



```
C:\Dac\ADS\Practice>java Gcd
a = 54
b = 24
0
C:\Dac\ADS\Practice>javac Gcd.java

C:\Dac\ADS\Practice>java Gcd
a = 54
b = 24
6
C:\Dac\ADS\Practice>java Gcd
a = 17
b = 13
1
```

EXPLANATION:
Base Case:
If (a%b==0) it means that b divides a perfectly, & b is the GCD.
Recursive Case:
If (b≠0), the function calls itself recursively with arguments b and a% b .

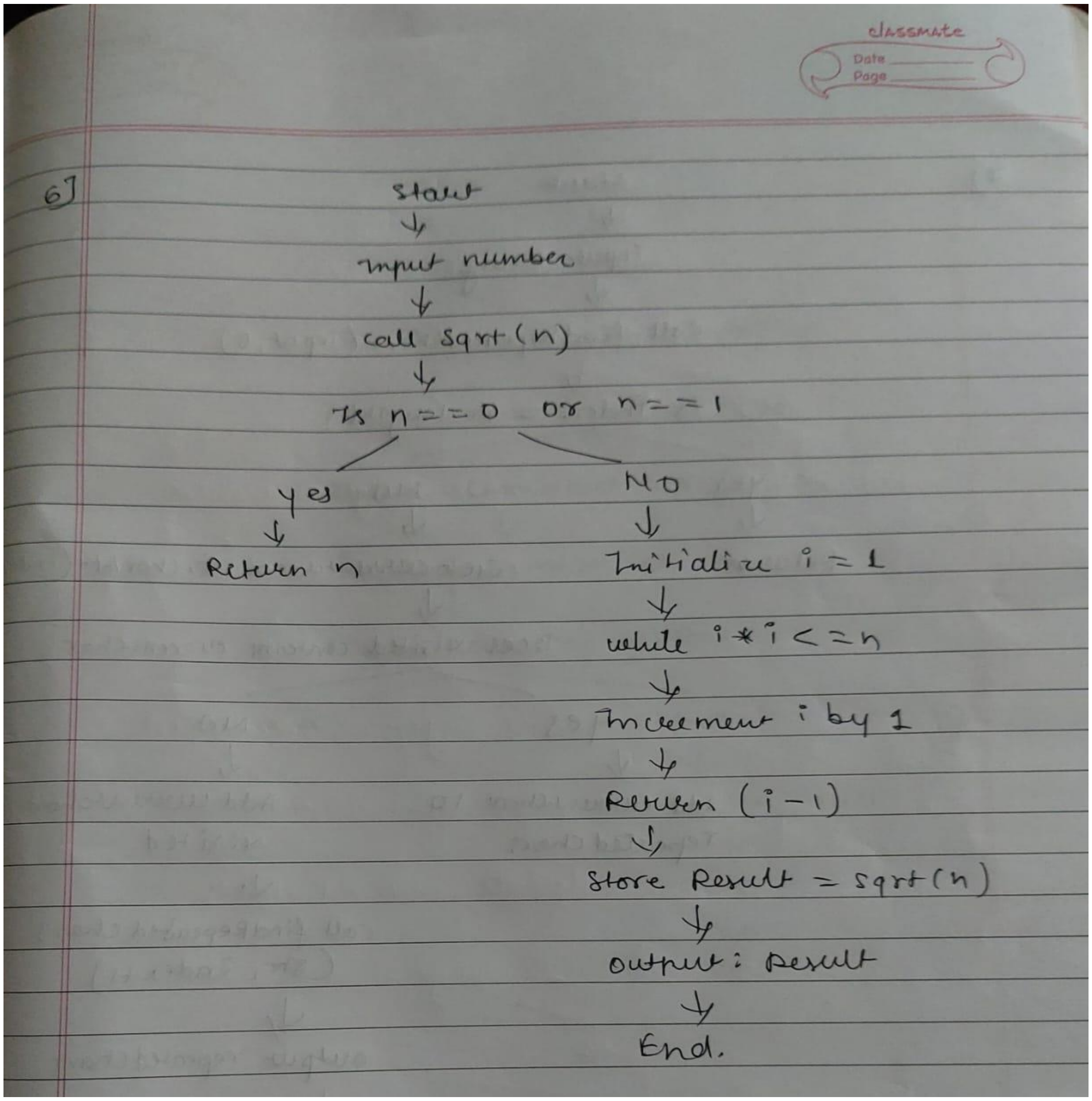
Time Complexity: O(n)
Space Complexity: O(1)

6. Find Square Root
Problem: Write a Java program to find the square root of a given number (using integer approximation).

Test Cases:

Input: x = 16
Output: 4
Input: x = 27
Output: 5

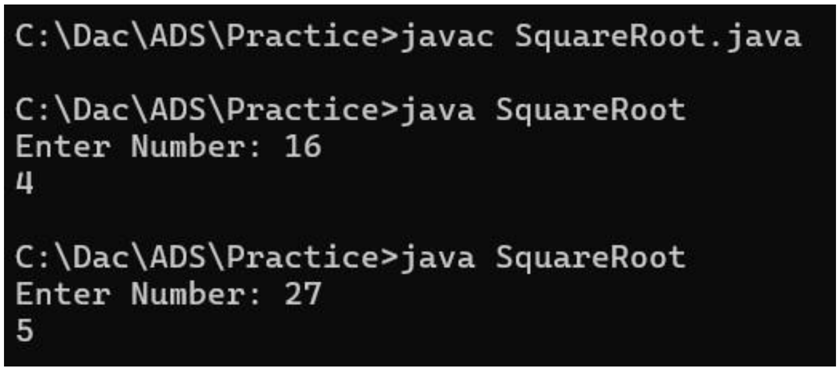
FLOWCHART:



PROGRAM:

```
import java.util.Scanner;

public class SquareRoot{
            static int sqrt(int n)
            {
                        if (n == 0 || n == 1) {
           return n;
        }
        int i = 1;
        while (i * i <= n) {
           i++;
        }
        return i - 1;
            }
            public static void main(String args[])
            {
                        Scanner sc = new Scanner(System.in);

                        System.out.print("Enter Number: ");
                        int n = sc.nextInt();

                        int result = sqrt(n);

                        System.out.println(result);
            }
}
```

OUTPUT:



```
C:\Dac\ADS\Practice>javac SquareRoot.java

C:\Dac\ADS\Practice>java SquareRoot
Enter Number: 16
4

C:\Dac\ADS\Practice>java SquareRoot
Enter Number: 27
5
```

EXPLANATION:

Base Case: If the input number n is either 0 or 1, the square root is the number itself.
Iterative Case:
You start from i = 1 and repeatedly check if i^2 is less than or equal to n.
Once i^2 exceeds n, you know the integer square root lies between i-1 and i, so you return i-1.

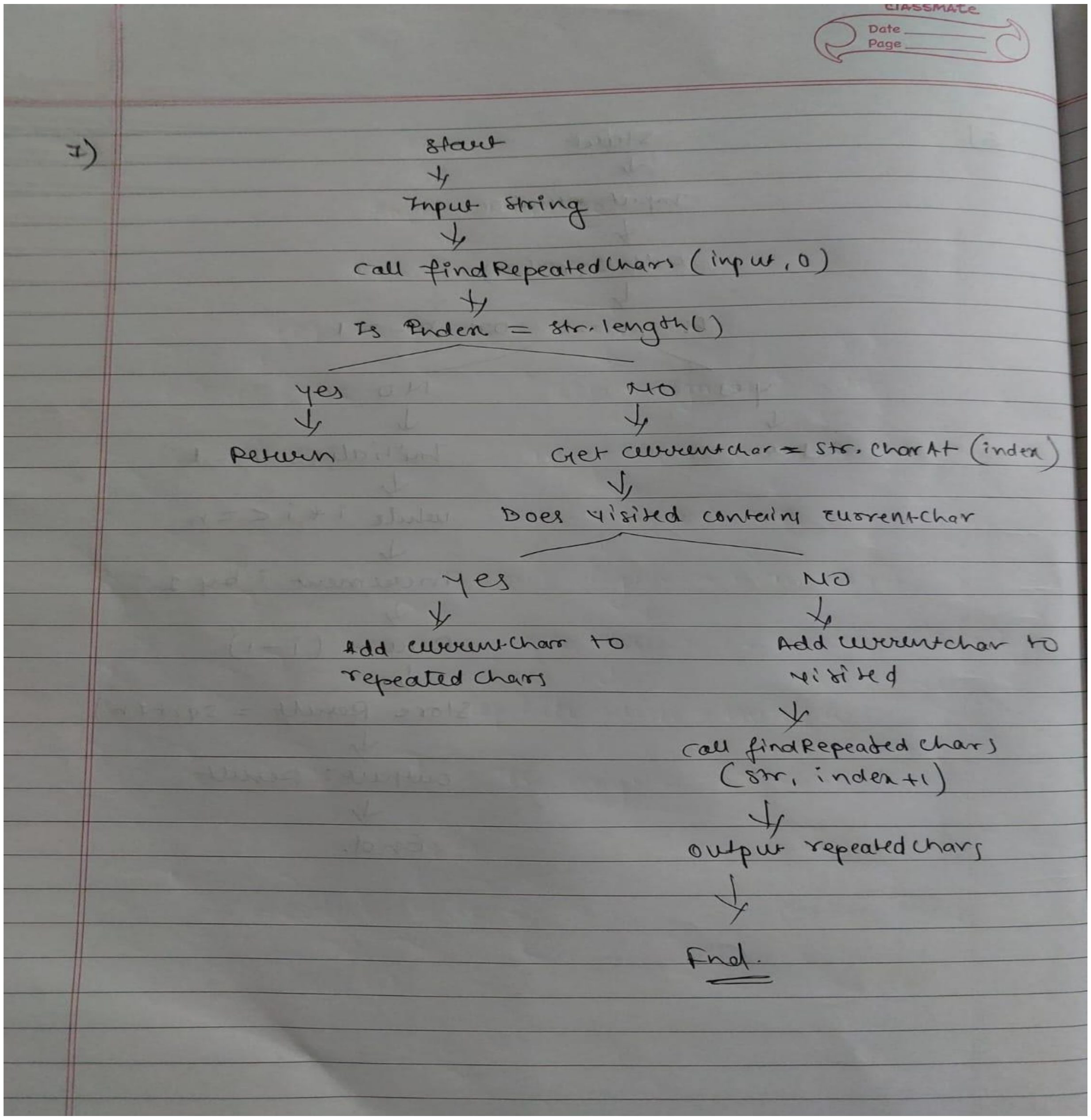
Time Complexity: O(√n)
Space Complexity: O(1)

7. Find Repeated Characters in a String
Problem: Write a Java program to find all repeated characters in a string.

Test Cases:

Input: "programming"
Output: ['r', 'g', 'm']
Input: "hello"
Output: ['l']

FLOWCHART:

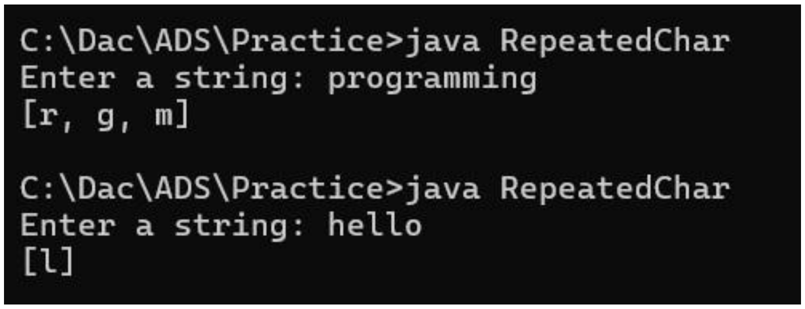


Classmate
Date
Page
7)
Start
Input String
call findRepeatedChars (input, 0)
Is index = str.length()
yes
NO
Return
Get currentchar = str.charAt (index)
Does visited contains currentChar
yes
NO
Add currentChar to repeated chars
Add currentchar to visited
call findRepeatedChars (str, index+1)
output repeated chars
End.

PROGRAM:

```
import java.util.*;

public class RepeatedChar {
    static Set<Character> visited = new HashSet<>();
    static Set<Character> repeatedChars = new HashSet<>();

    // Recursive function to find repeated characters
    static void findRepeatedChars(String str, int index) {
        // Base case: If index reaches the length of the string, return
        if (index == str.length()) {
            return;
        }

        char currentChar = str.charAt(index);

        // If character is already visited, add it to repeated characters
        if (visited.contains(currentChar)) {
            repeatedChars.add(currentChar);
        } else {
            visited.add(currentChar);
        }

        // Recursive call for the next character
        findRepeatedChars(str, index + 1);
    }

    public static void main(String[] args) {
                        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a string: ");
        String input = sc.nextLine();

        findRepeatedChars(input, 0); // Start the recursion

        System.out.println(repeatedChars);

        sc.close();
    }
}
```

OUTPUT:


```
C:\Dac\ADS\Practice>java RepeatedChar
Enter a string: programming
[r, g, m]

C:\Dac\ADS\Practice>java RepeatedChar
Enter a string: hello
[l]
```

EXPLANATION:
Static Sets:
We use two static Set<Character>:
visited: To keep track of characters that have been encountered.
repeatedChars: To store characters that are found to be repeated.

Recursive Function:
The function findRepeatedChars takes the input string and the current index as parameters.

Base Case: If the index reaches the length of the string, the recursion stops.

Checking Current Character:
If the current character has already been seen (exists in visited), it is added to repeatedChars.
Otherwise, it is added to visited.
The function then calls itself recursively, moving to the next character by incrementing the index.

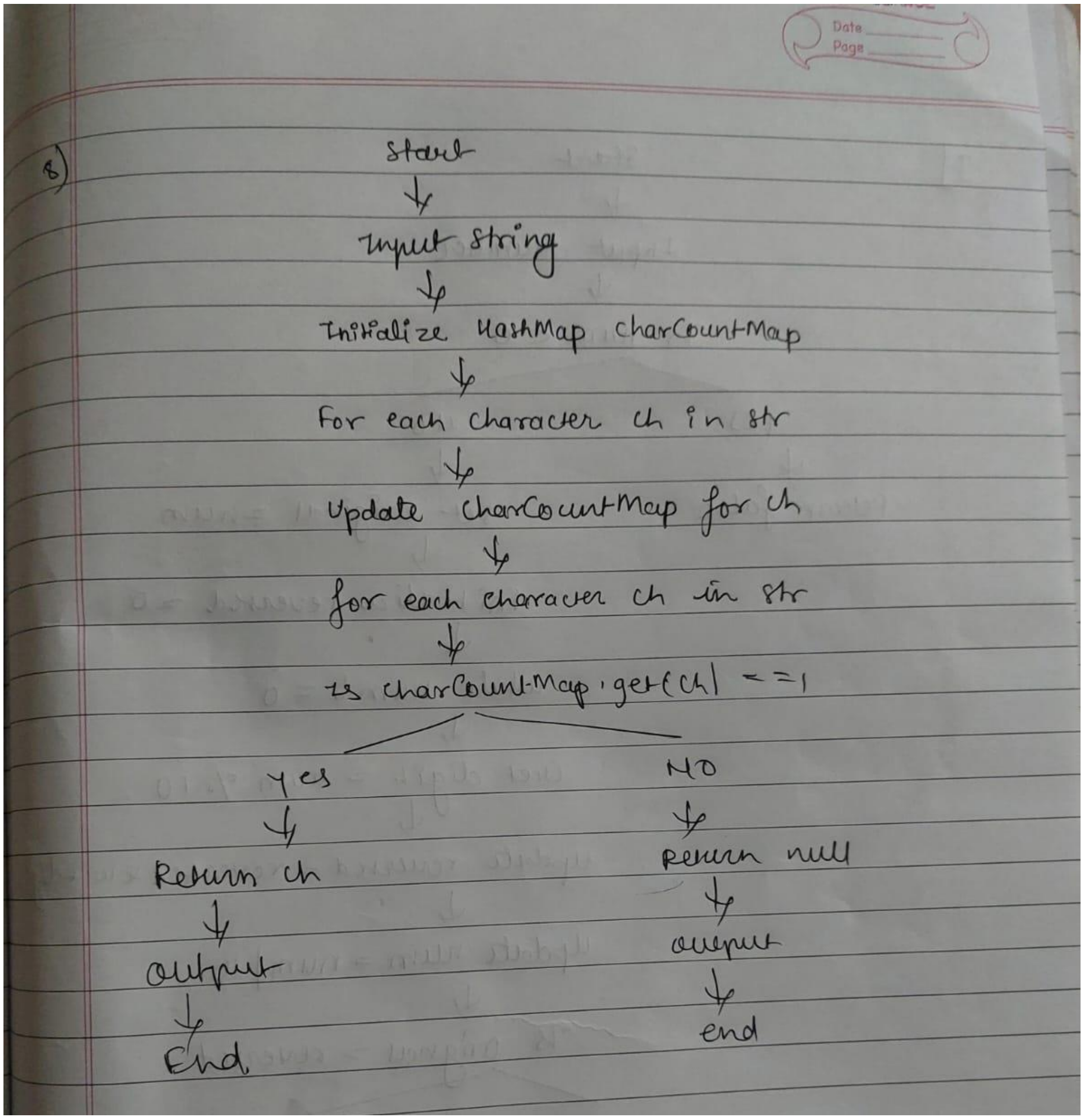
Time Complexity: O(n)
Space Complexity: O(n)

8. First Non-Repeated Character
Problem: Write a Java program to find the first non-repeated character in a string.

Test Cases:

Input: "stress"
Output: 't'
Input: "aabbcc"
Output: null

FLOWCHART:

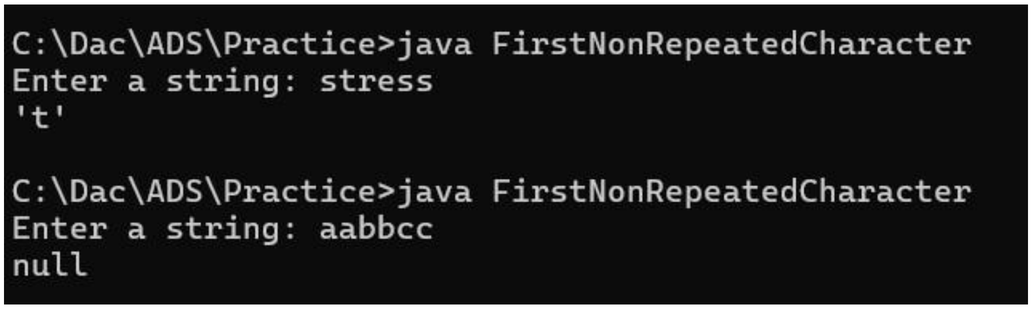


8)
Start
Input string
Initialize HashMap charCountMap
For each character ch in str
Update charCountMap for ch
for each character ch in str
is charCountMap.get(ch) ==1
Yes
NO
Return ch
output
End.
Return null
output
end
Date
Page

PROGRAM:

```
import java.util.*;

public class FirstNonRepeatedCharacter {
    public static Character findFirstNonRepeatedChar(String str) {
        HashMap<Character, Integer> charCountMap = new HashMap<>();

        for (char ch : str.toCharArray()) {
            charCountMap.put(ch, charCountMap.getOrDefault(ch, 0) + 1);
        }

        for (char ch : str.toCharArray()) {
            if (charCountMap.get(ch) == 1) {
                return ch;
            }
        }

        return null;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.print("Enter a string: ");
        String input = sc.nextLine();

        Character result = findFirstNonRepeatedChar(input);
        if (result != null) {
            System.out.println("'"+result+"'");
        } else {
            System.out.println("null");
        }

        sc.close();
    }
}
```

OUTPUT:



```
C:\Dac\ADS\Practice>java FirstNonRepeatedCharacter
Enter a string: stress
't'

C:\Dac\ADS\Practice>java FirstNonRepeatedCharacter
Enter a string: aabbcc
null
```

EXPLANATION:
HashMap<Character, Integer>:
This HashMap stores the frequency of each character in the string.

Counting Character Frequencies:
The first loop iterates through each character in the string and updates its count in the HashMap.

Finding the First Non-Repeated Character:
The second loop iterates through the string again and checks the frequency of each character.
The first character with a frequency of one is returned as the first non-repeated character.

Return Value:
If a non-repeated character is found, it is returned. If none exist, the method returns null.

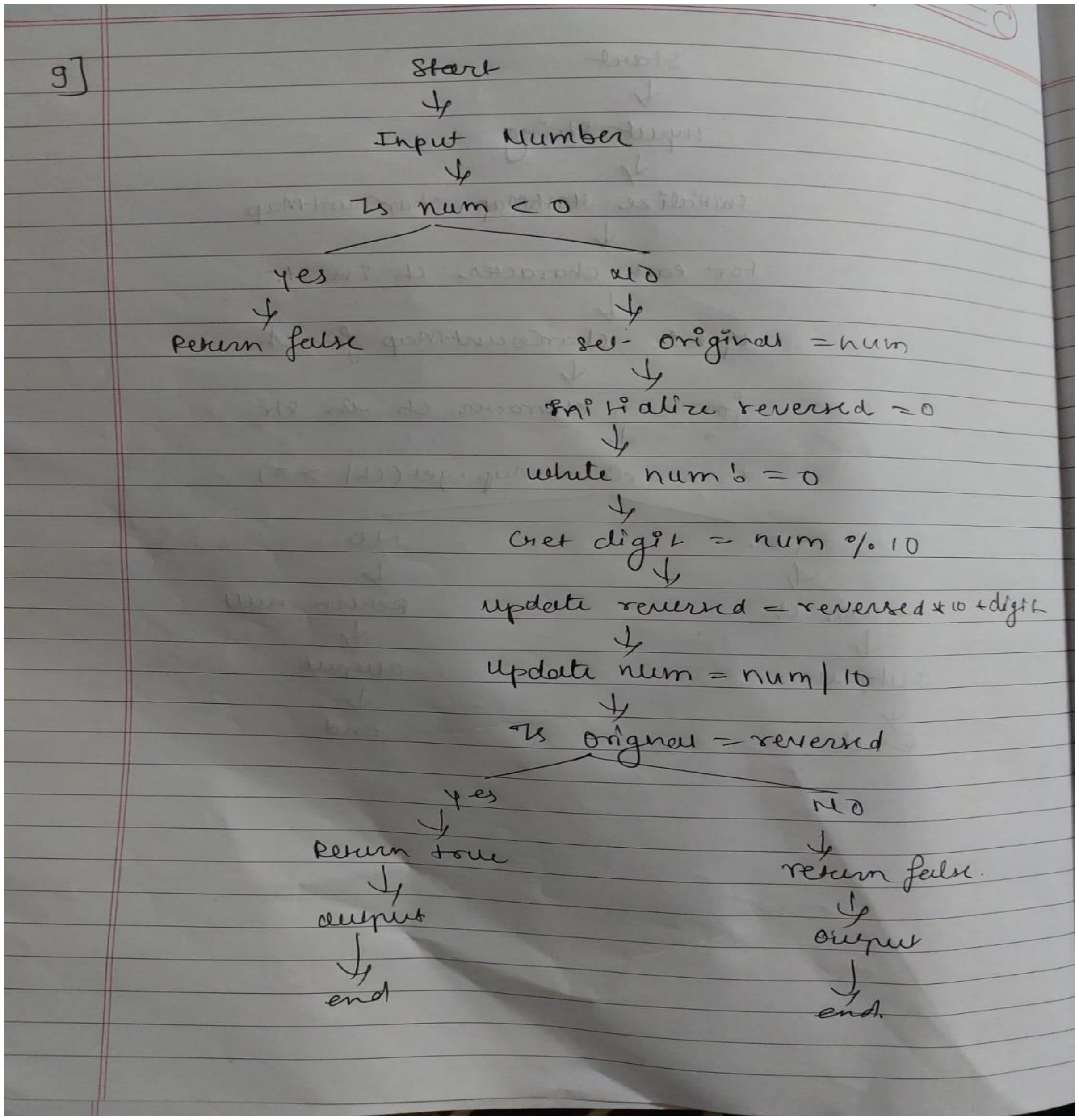
Time Complexity: O(n)
Space Complexity: O(n)

9. Integer Palindrome
Problem: Write a Java program to check if a given integer is a palindrome.

Test Cases:

Input: 121
Output: true
Input: -121
Output: false

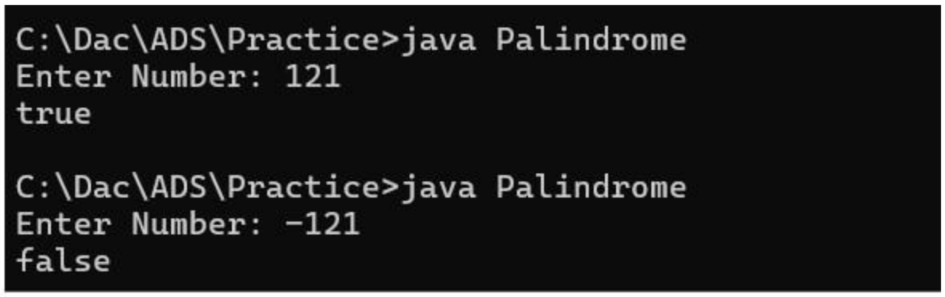
FLOWCHART:



PROGRAM:

```
import java.util.Scanner;

public class Palindrome {

    public static boolean isPalindrome(int num) {
        if (num < 0) {
            return false;
        }

        int original = num;
        int reversed = 0;

        while (num != 0) {
            int digit = num % 10;
            reversed = reversed * 10 + digit;
            num /= 10;
        }

        return original == reversed;
    }

    public static void main(String[] args) {
                        Scanner sc = new Scanner(System.in);
                        System.out.print("Enter Number: ");

                        int num = sc.nextInt();

        System.out.println(isPalindrome(num));
    }
}
```

OUTPUT:



```
C:\Dac\ADS\Practice>java Palindrome
Enter Number: 121
true

C:\Dac\ADS\Practice>java Palindrome
Enter Number: -121
false
```

EXPLANATION:
First the input will taken an integer num.

Then it will check for Negativity: If num is negative, it cannot be a palindrome, so the method returns false.

Reverse the Number:
The original value of num is stored in the variable original.
A while loop is used to reverse the digits of num.
In each iteration:
The last digit of num is obtained using num % 10.
This digit is added to the reversed number (multiplied by 10 to shift digits left).
The last digit is removed from num using integer division (num /= 10).

Comparison: After reversing, the method compares original and reversed. If they are equal, the number is a palindrome, so it returns true; otherwise, it returns false.

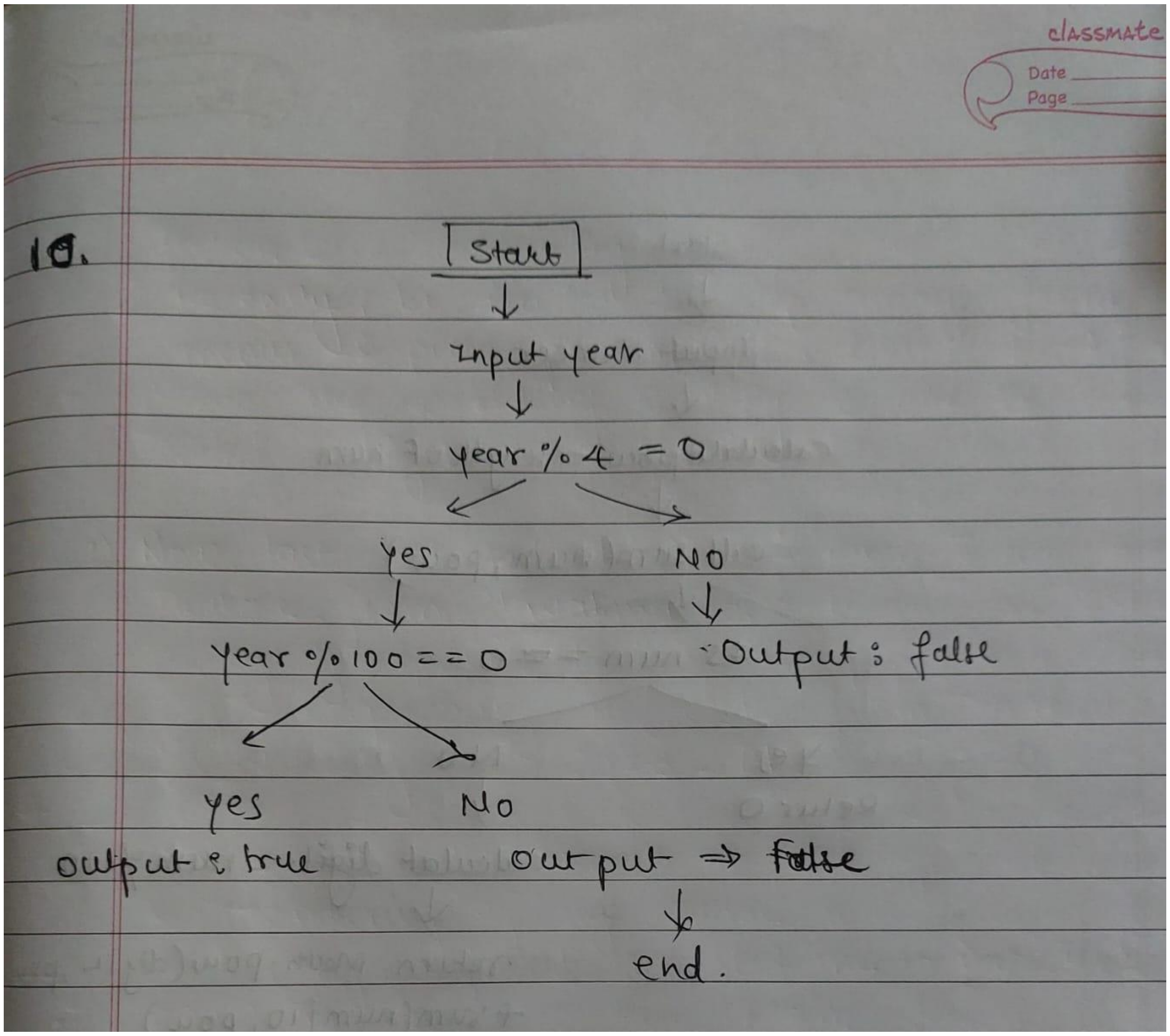
Time Complexity: O(n)
Space Complexity: O(1)

10. Leap Year
Problem: Write a Java program to check if a given year is a leap year.

Test Cases:

Input: 2020
Output: true
Input: 1900
Output: false

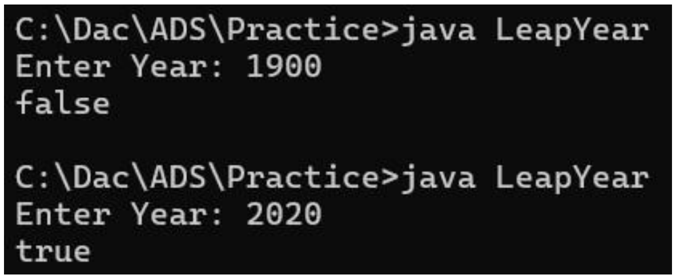
FLOWCHART:



PROGRAM:

```
import java.util.Scanner;

public class LeapYear{

            static boolean isLeap(int year)

            {

                        if (year % 4 == 0) {

          if (year % 100 == 0) {

             return year % 400 == 0;

          } else {

             return true;

          }

       } else {

          return false;

       }

            }


            public static void main(String args[])

            {

                        Scanner sc = new Scanner(System.in);

                        System.out.print("Enter Year: ");

                        int year = sc.nextInt();

                        System.out.println(isLeap(year));

            }

}
```

OUTPUT:



```
C:\Dac\ADS\Practice>java LeapYear
Enter Year: 1900
false

C:\Dac\ADS\Practice>java LeapYear
Enter Year: 2020
true
```

EXPLANATION:

A year is a leap year if:It is divisible by 4 **&** If it is divisible by 100, it must also be divisible by 400.

i.e.:

If year % 4 == 0:

If year % 100 == 0, return year % 400 == 0

Otherwise, return true

If year is not divisible by 4, return false (not a leap year).

Time Complexity: O(1)
Space Complexity: O(1)