**C-DAC Mumbai Date 25/09/2024**

**Subject: Algorithm and Data Structure**

**Assignment 1**

**Solve the assignment with following thing to be added in each question.**

-Program

-Flow chart

-Explanation

-Output

-Time and Space complexity

================================================================================

**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**

**Algorithm:**

Step 1: Input Number

Step 2: Calculate Cube of Each Digit

rem=n1%10; // Extract each digit from the number

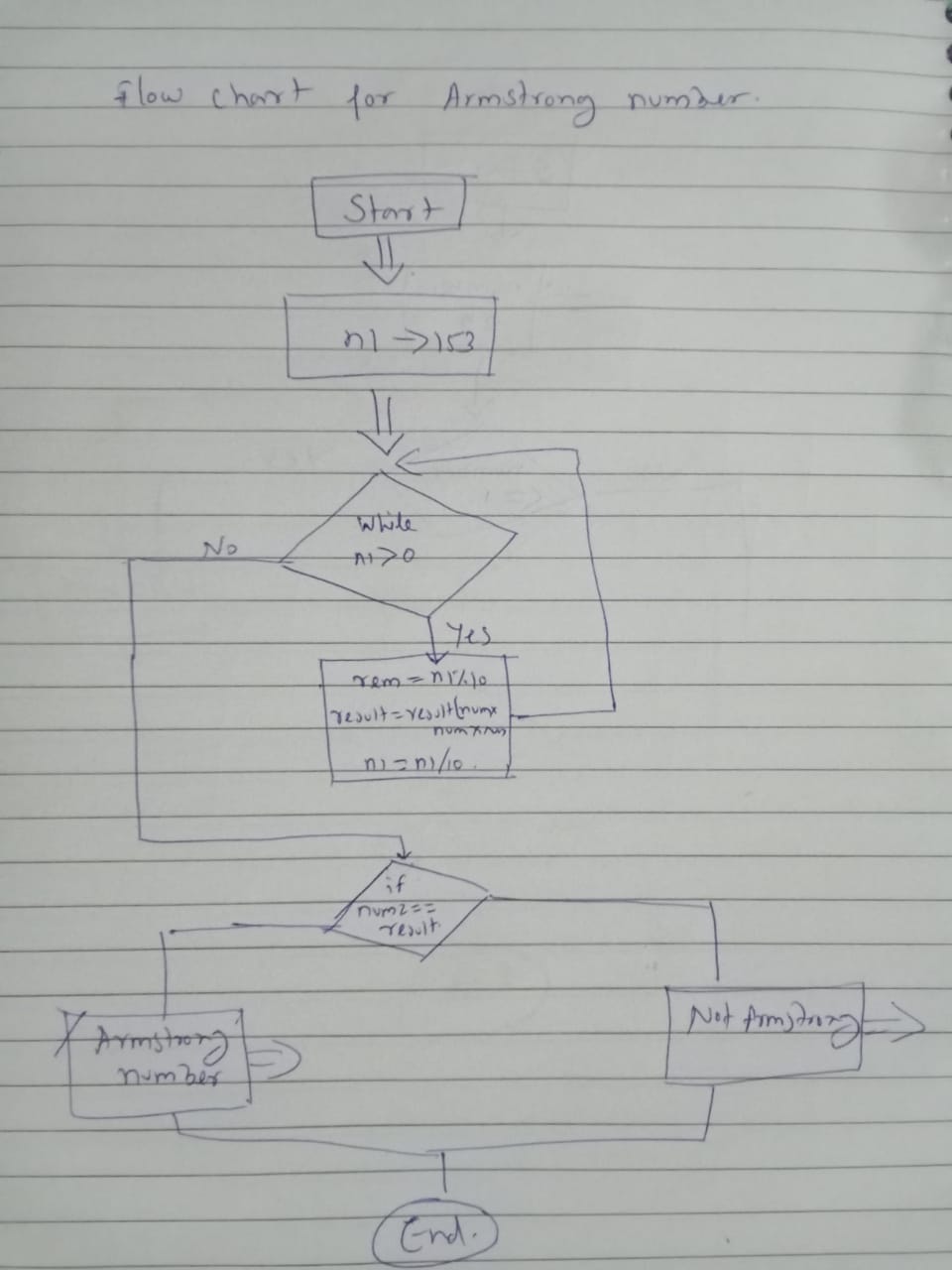
(rem\*rem\*rem);- Calculate the cube of each digit

result=result+- Add the cubes to a result

Step 3: Compare Total with Original Number

Step 4: If the total equals the original number, it's an Armstrong number else, it's not an Armstrong number.

**Flow chart**



**Program:**

public class ArmstrongNumber{

public static void main (String[] args){

int n1=153;

int result=0,rem;

int num2=n1;

while(n1>0)

{

rem=n1%10;

result=result+(rem\*rem\*rem);

n1=n1/10;

}

if(num2==result)

{

System.out.println(result+" Is Armstrong Number");

}

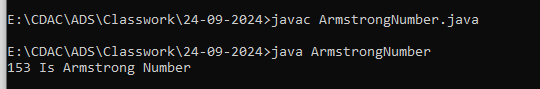
else

System.out.println(result+" Is not Armstrong Number");

}

}

**Output:**



**Time complexity: Olog(n)**

**Space complexity: O(1)**

===============================================================================

**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

**Algorithm**

Step 1: Input Number

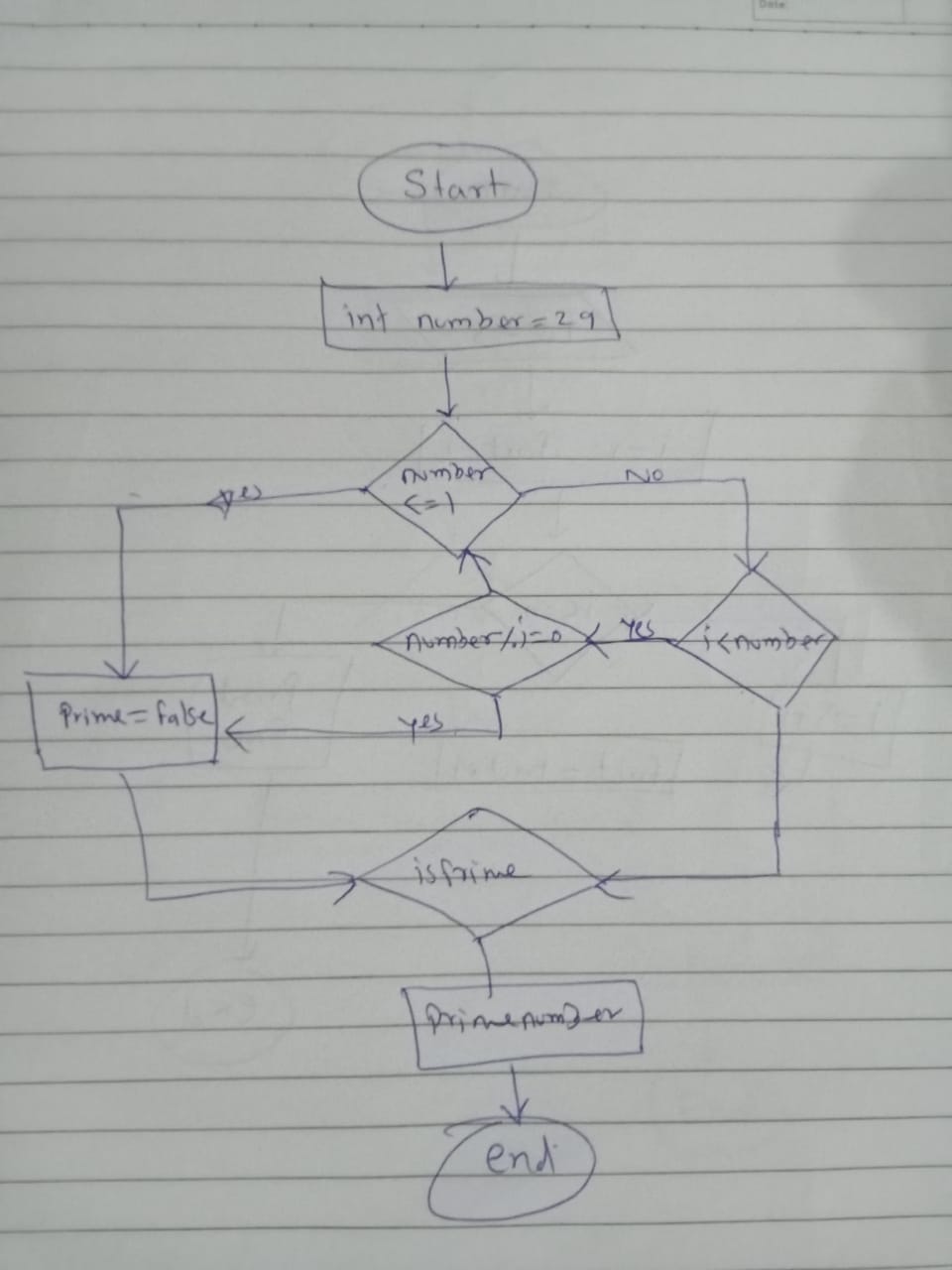
Step 2: If the number is less than or equal to 1, it's not prime

Step 3: Check Divisibility

Step 4: If no divisors found, the number is prime

Step 5: Print whether the number is prime or not

**Flowchart:**



**Program:**

public class PrimeNumber{

public static void main(String[] args){

int number = 29;

boolean isPrime = true;

if(number <= 1){

isPrime = false;

}

else{

for(int i = 2; i<number;i++){

if(number % i == 0){

isPrime = false;

break;

}

}

}

if(isPrime){

System.out.println( number +" is a prime number");

}

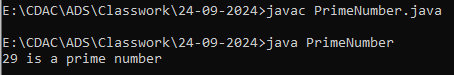
else{

System.out.println(number + " is not a prime number");

}

}

}



Time complexity:O(1)

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**

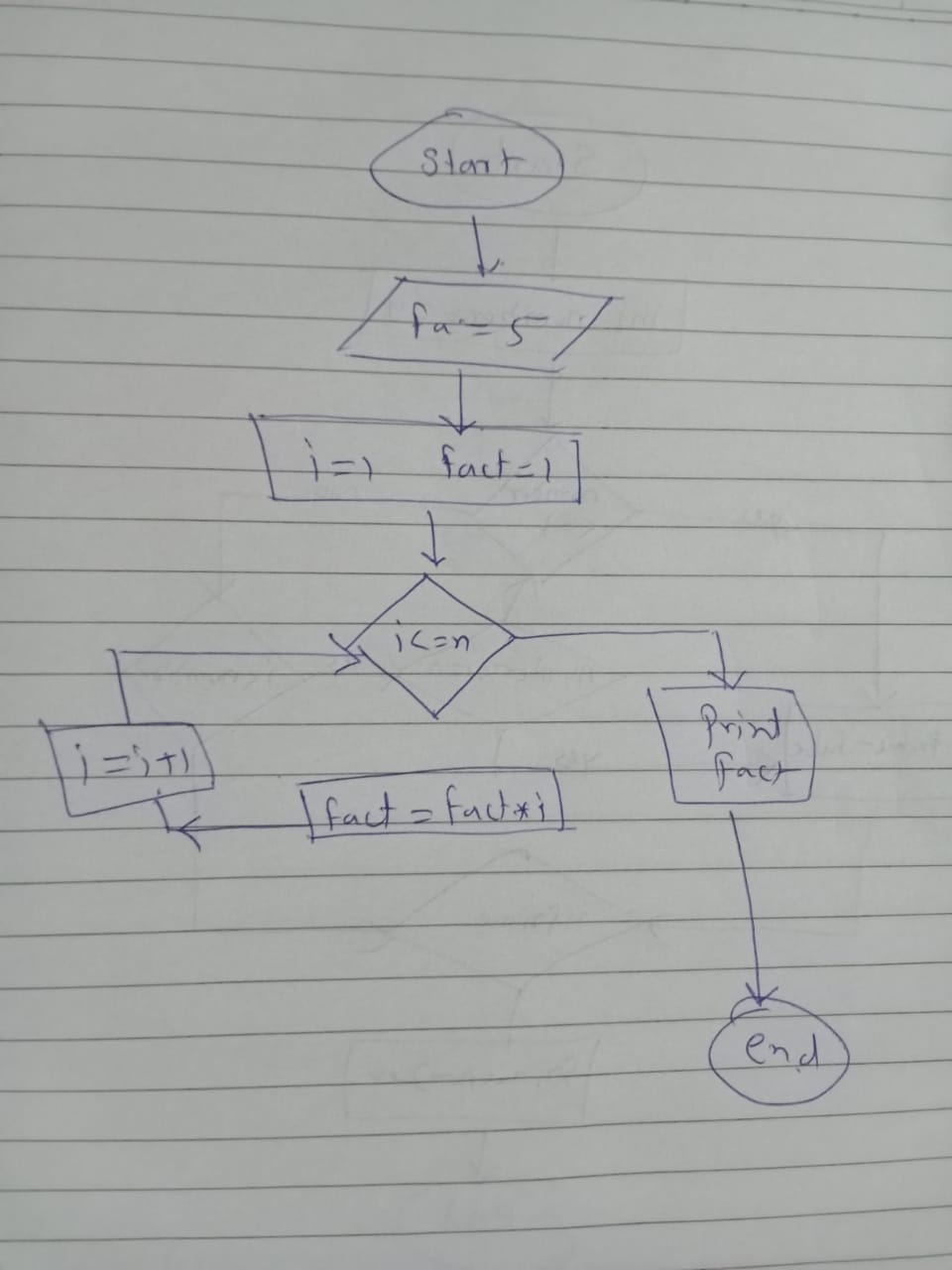
**Explaination:**

1. The function takes an integer n as input.

2. Base case: If n is 0 or 1, return 1 (since factorial of 0 and 1 is 1).

3. Recursive case: If n is greater than 1, return n multiplied by the factorial of n-1.

**Flow chart:**



**Program:**

class facto{

static int fact(int n)

{

if(n<=1)

return 1;

else

return n\*fact(n-1);

}

public static void main(String [] args)

{

System.out.println(fact(5));

}

}



**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]**

Step 1: Determine Number of Terms

Step 2: Calculate Fibonacci Numbers

- For each number (starting from 0):

- If number is 0 or 1, its value is the number

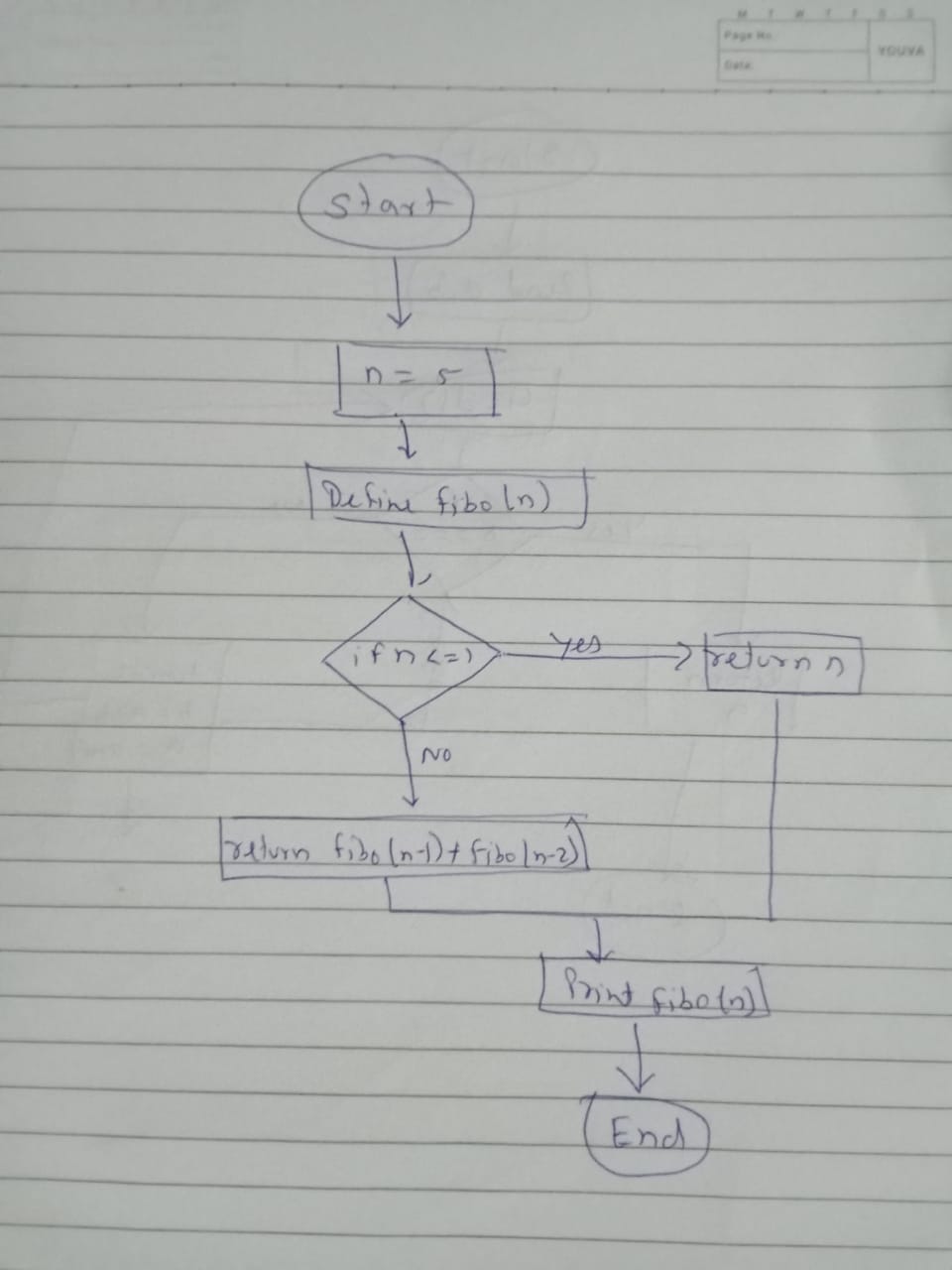
- Otherwise, calculate the number as the sum of the previous two number

Step 3: Generate Fibonacci Sequence

Step 4: Store Fibonacci Sequence

Step 5: Display Fibonacci Series

**Flowchart:**



**Program:**

public class FibonacciSeries {

public static void main(String[] args) {

int n = 5;

System.out.println("Fibonacci Series for n = " + n + " ");

printFibonacci(n);

}

public static void printFibonacci(int n) {

for (int i = 0; i < n; i++) {

System.out.print(fibonacci(i) );

}

}

public static int fibonacci(int n) {

if (n <= 1) {

return n;

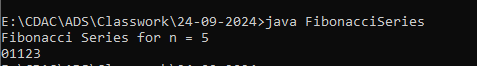
} else {

return fibonacci(n - 1) + fibonacci(n - 2);

}

}

}



**Time Complexity: O(2^n)**

**Space Complexity: O(n)**

**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

**Explaination:**

1: Initialize Numbers

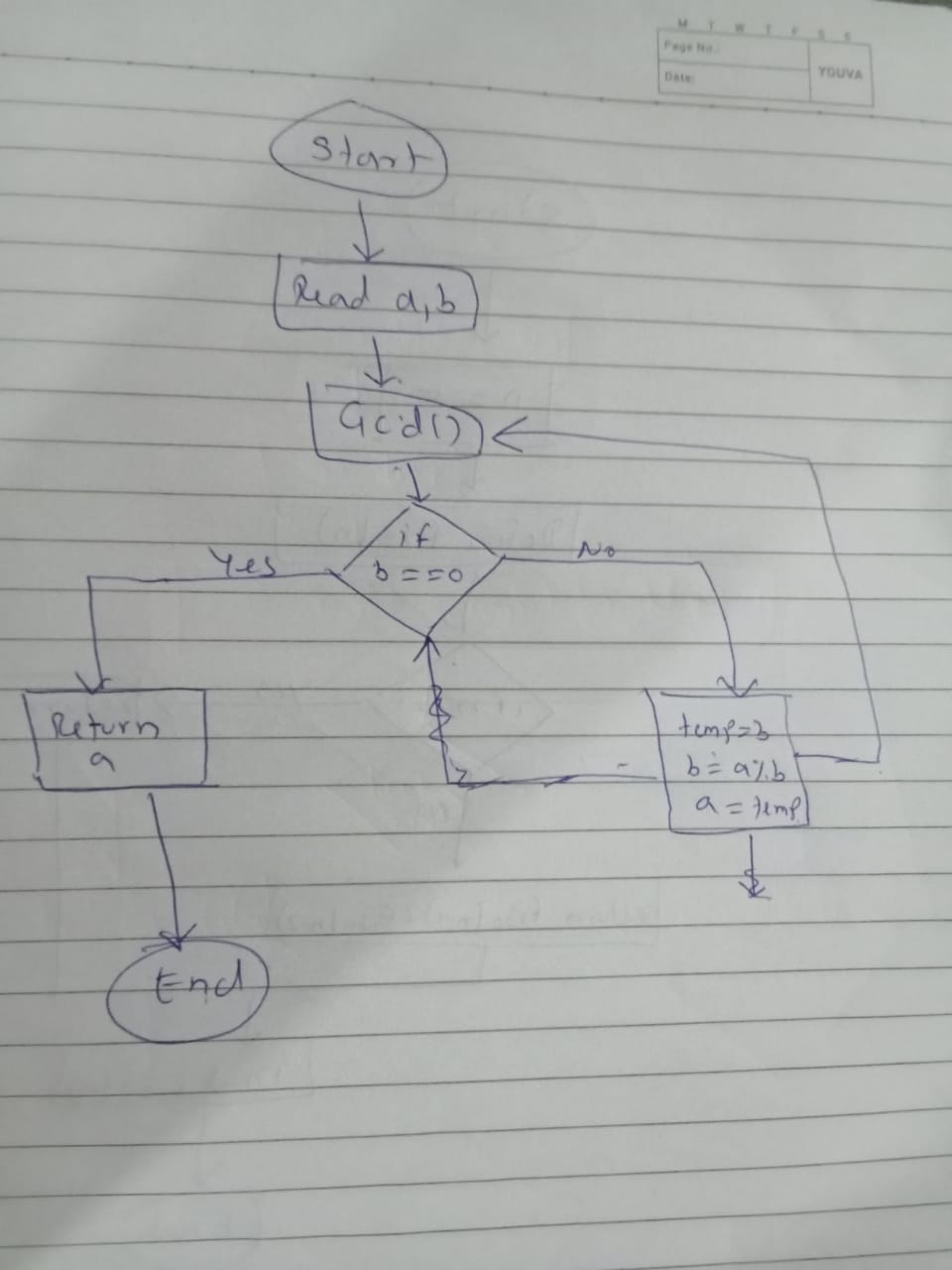
2: Check Base Case

3: Calculate Remainder

4: Recursive Replacement

5: return GCD

**Flowchart:**



**Program:**

public class GreatestCommonDivisor {

public static void main(String[] args) {

int a = 54;

int b=24;

int gcd = findGCD(a, b);

System.out.println("GCD of " + a + " and " + b + ": " + gcd);

}

public static int findGCD(int a, int b) {

if (b == 0) {

return a;

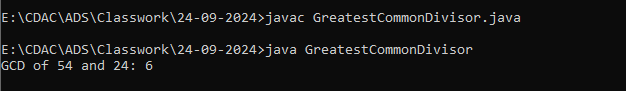
} else {

return findGCD(b, a % b);

}

}

}

**Output:**

**Time Complexity: O(log(n))**

**Space Complexity: O(log(n))**

**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

**Algorithm**

1: Input Number

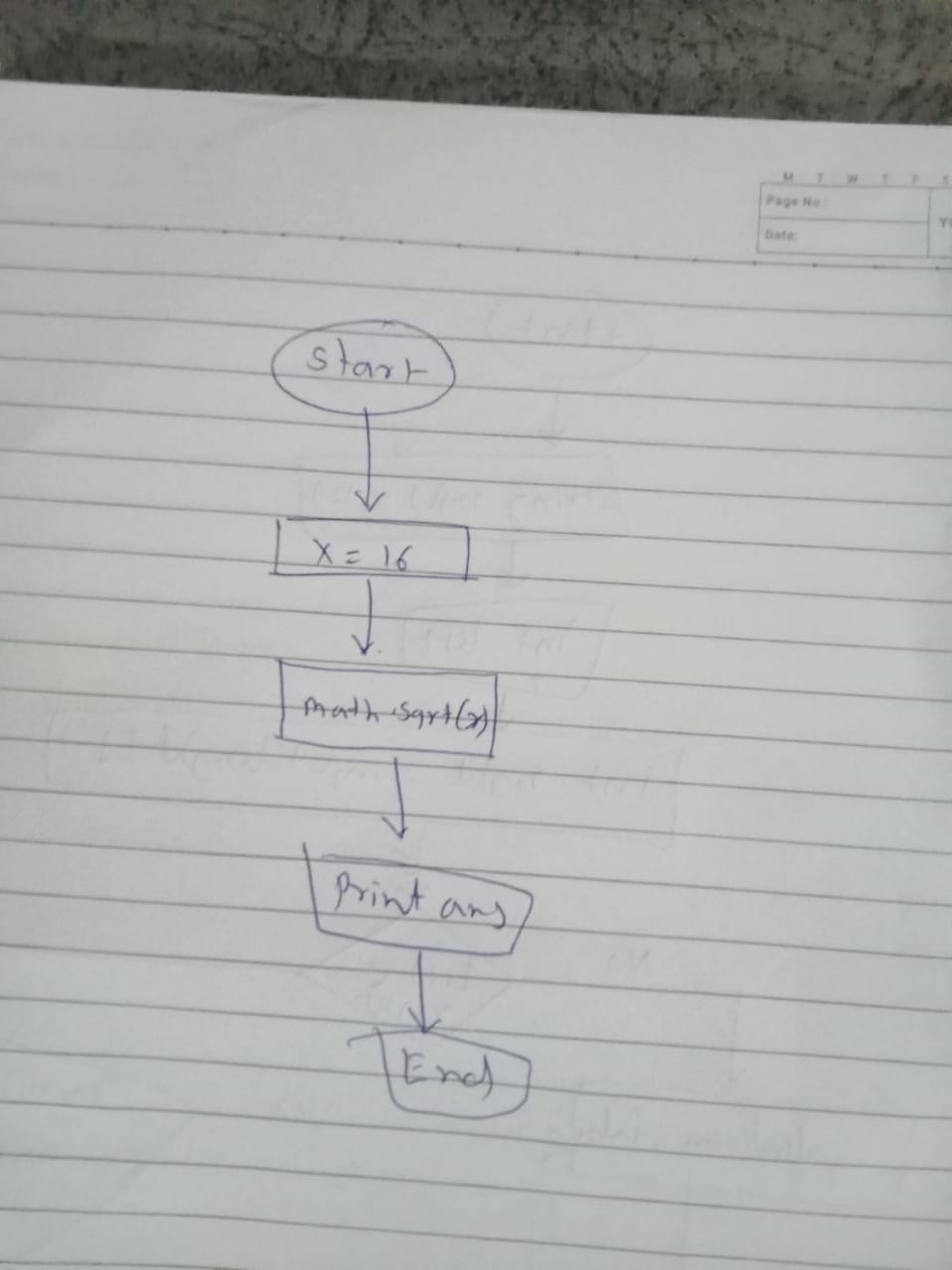
2: Calculate Square Root

3: Data Type Conversion (Narrowing)

4: Store Result

5: Display Result

**Flowchart:**



**Program:**

public class SquareRoot {

public static void main(String[] args) {

int x = 16;

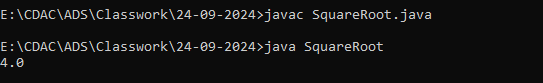
double ans = (int) Math.sqrt(x); //narrowing conversion - data type

System.out.println(ans);

}

}

**Output:**



**Time Complexity: O(1)**

**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']

**Explaination:**

1 Creates a String variable input with the value "programming".

2. Create an array

3. For each character, increment its index in the array

4. Identify characters with count greater than 1

5. Store repeated characters.

6.Display the char

**Program:**

public class RepeatedCharacters{

public static void main(String[] args) {

String input = "programming";

int[] charCount = new int[26];

for (int i = 0; i < input.length(); i++) {

charCount[input.charAt(i) - 'a']++;

}

System.out.print("Repeated characters: ");

for (int i = 0; i < charCount.length; i++) {

if (charCount[i] > 1) {

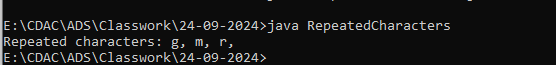
System.out.print((char) ('a' + i) + ", ");

}

}

}

}



Time Complexity: O(n)

Space Complexity: O(1)

**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

Program:

public class NonRepeatedCharacter

{

public static void main(String[] args)

{

String str = "stress";

char[] arr = str.toCharArray();

for(int i=0; i<arr.length; i++)

{

for(int j=i+1; j<arr.length; j++)

{

if(arr[i] != arr[j])

{

System.out.println(arr[j]);

System.exit(0);

}

else

{

System.out.println("null");

System.exit(0);

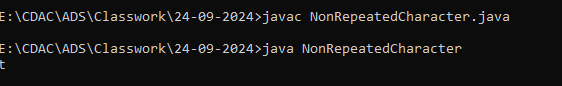
}

}

}

}

Output:



**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**

**Algorithm:**

1: Initialize Variables

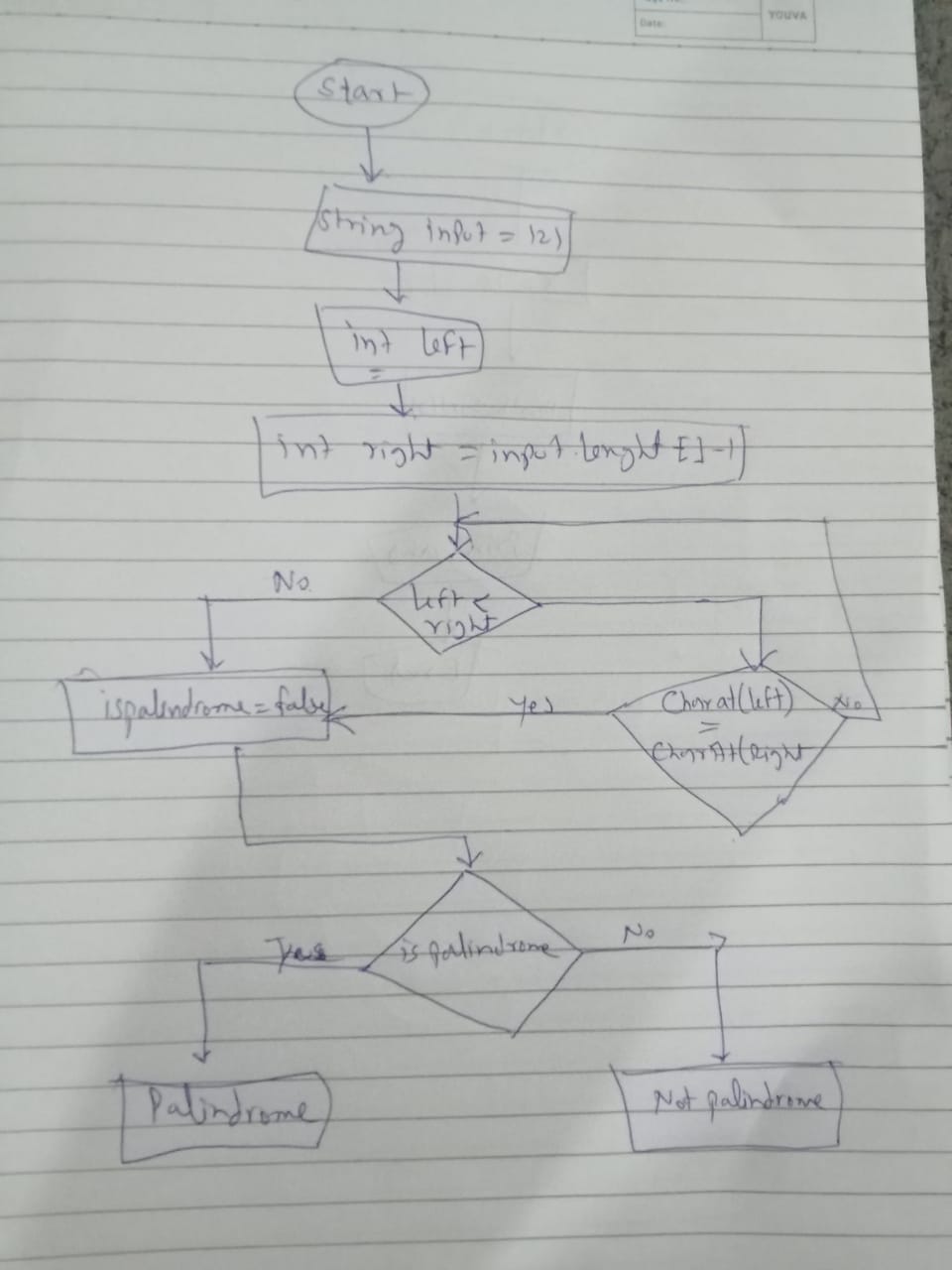
2: Compare Characters from Both Ends

3. Increment left and decrement right

4. Repeat steps 2-3 until left pointer meets or crosses right

5.print result.

**Flowchart:**



**Program:**

public class IntegerPalindrome {

public static void main(String[] args) {

String input ="121";

boolean isPalindrome = true;

int left = 0;

int right = input.length() - 1;

while (left < right) {

if (input.charAt(left) != input.charAt(right)) {

isPalindrome = false;

break;

}

left++;

right--;

}

if (isPalindrome) {

System.out.println(input + " is a palindrome.");

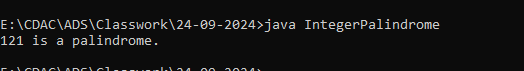
} else {

System.out.println(input + " is not a palindrome.");

}

}

}



**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

**Explaination:**

1: Get User Input

2: Check Divisibility by 4

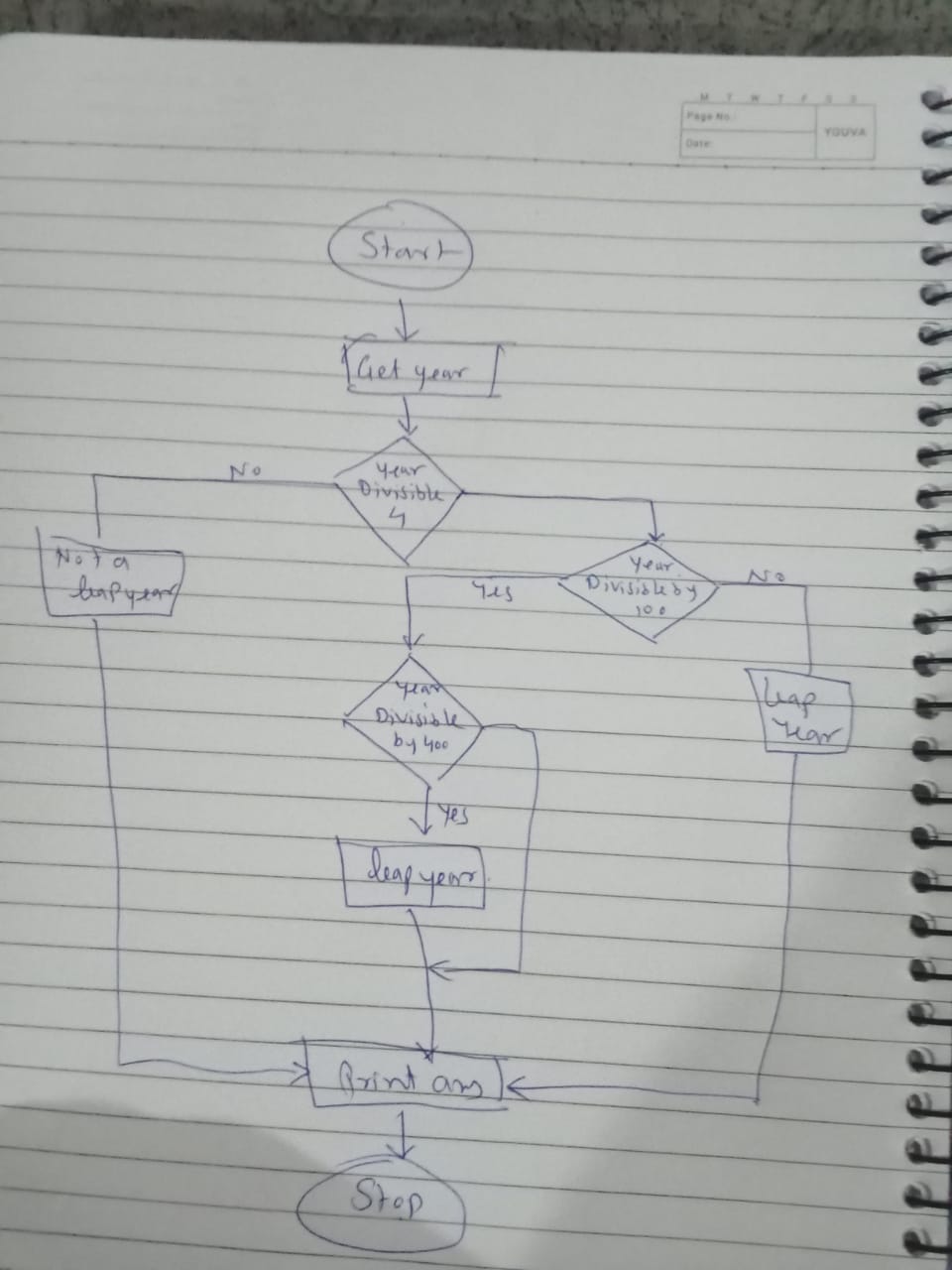
3: If year is divisible by 100, it must also be divisible by 400 to be a leap year

4:If year is divisible by 100 but not 400, it's not a leap year (goto Step 6)

5:If year passes Step 2 and Step 3 checks, it's a leap year

6:Display leap year or not

**Flowchart:**



**Program:**

import java.util.\*;

class LeapYear{

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the year you want to choose: ");

int year = sc.nextInt();

if (year % 4 == 0 && year %100 != 0 || year % 400 == 0)

{

System.out.println(true);

}

else

{

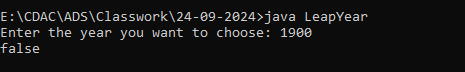
System.out.println(false);

}

}

}

**Output:**



**Time Complexity**: O(n)

**Space Complexity:** O(1)