C-DAC Mumbai

Date 25/09/2024

Subject: Algorithm and Data Structure
Assignment 1

Name: Yash Bandu Dhole
Center: Juhu

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.

Code:

```
package problem1;
import java.util.Scanner;

public class armstrongMain {

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

		int origNum = number;
		int result = 0;
		int digit = 0;

		while(origNum!=0) {	// counting the number of digit
			origNum /=10;
			digit++;
		}
		origNum = number;
		while (origNum!=0) {
			int remainder = origNum %10;
			result += Math.pow(remainder, digit);
			origNum /=10;
		}

		if(result == number) {
			System.out.println(number + " is an Armstrong number ");
		}else {
```

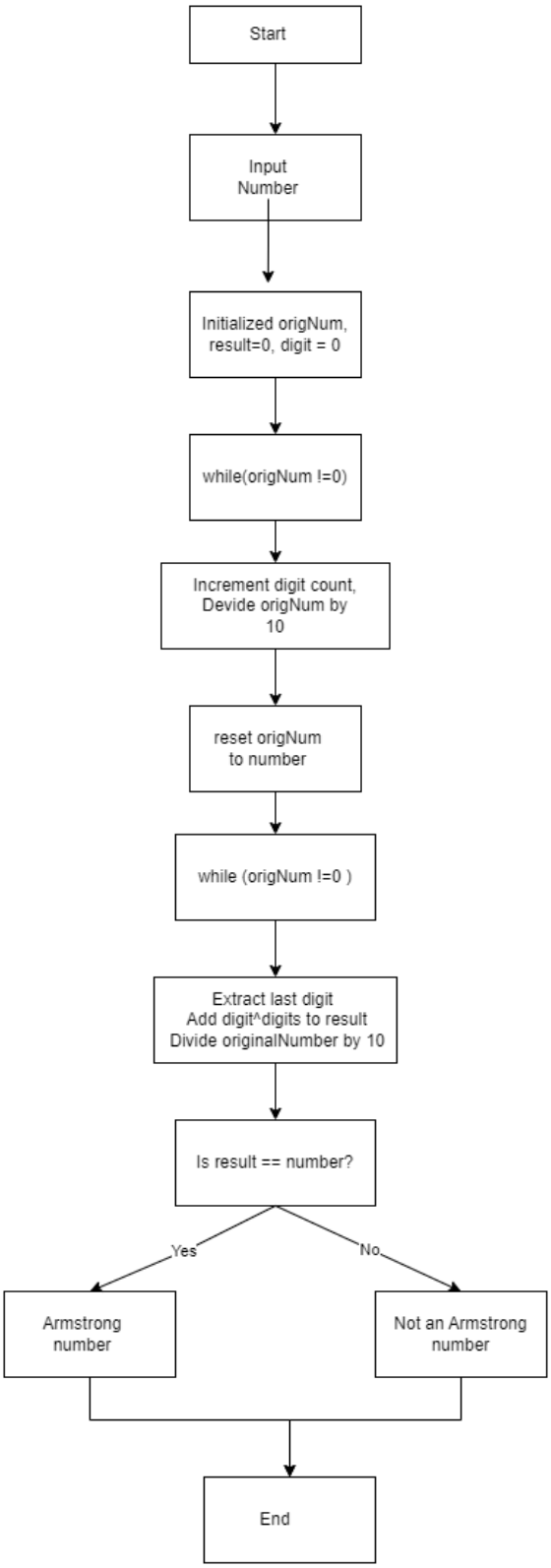
```
                    System.out.println(number + " is not an Armstrong number ");
                }

                sc.close();
        }

}
```

O/P:
enter a number
153
153 is an Armstrong number

Time Complexity: O(d) - d for number of digits.

Space Complexity: O(1)

Flowchart:


Start
Input
Number
Initialized origNum,
result=0, digit = 0
while(origNum !=0)
Increment digit count,
Devide origNum by
10
reset origNum
to number
while (origNum !=0)
Extract last digit
Add digit^digits to result
Divide originalNumber by 10
Is result == number?
Yes
No
Armstrong
number
Not an Armstrong
number
End

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

Code:

```
package problem2;
import java.util.Scanner;

public class primeCheck {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Enter number to check");
		int number = sc.nextInt();

		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");
		}
	}


}
```

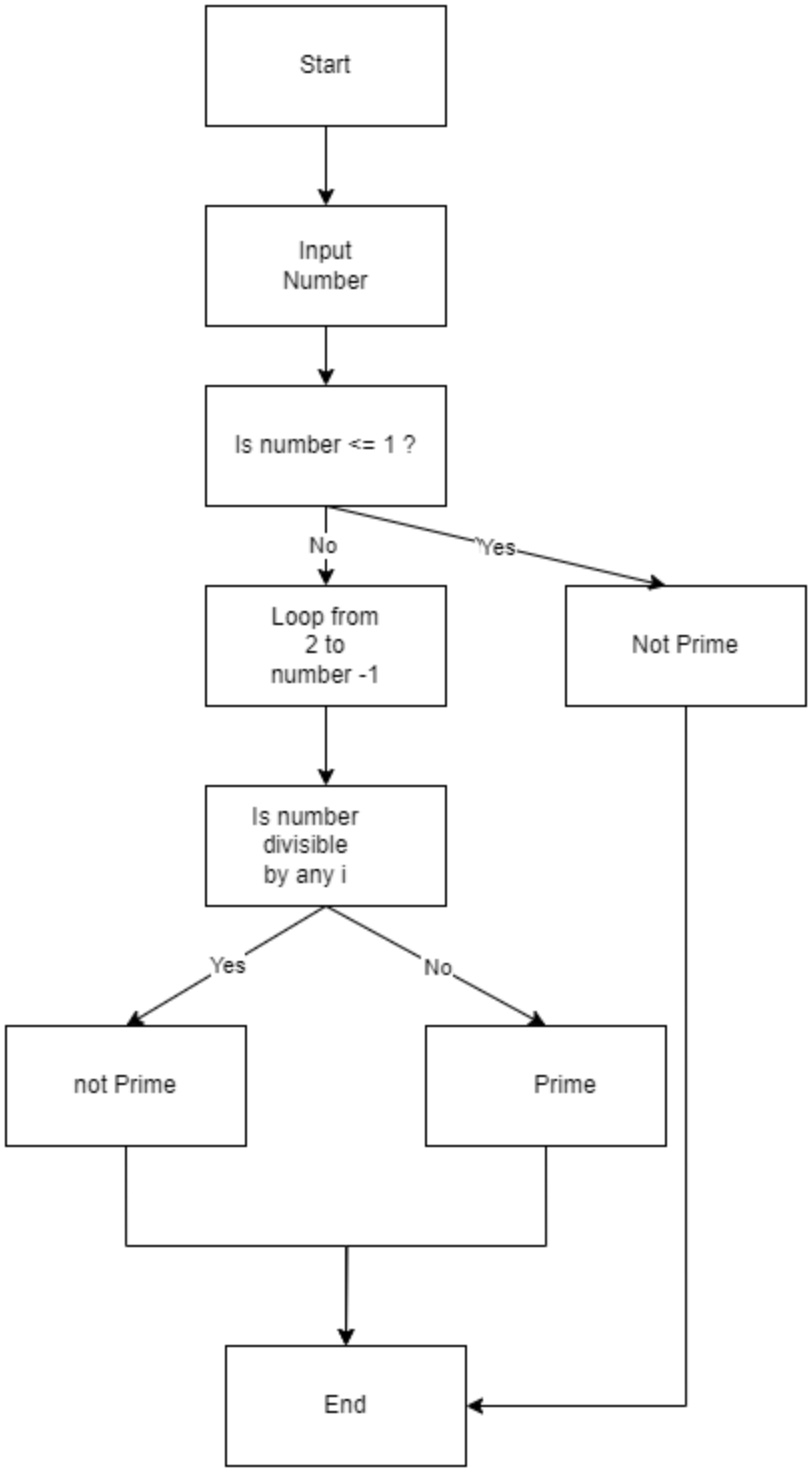
O/p:

```
Enter number to check
123
123 is not a prime number
```

Time Complexity: O(n)

Space Complexity: O(1)

Flowchart:


Start
Input
Number
Is number <= 1 ?
No
Yes
Loop from
2 to
number -1
Not Prime
Is number
divisible
by any i
Yes
No
not Prime
Prime
End

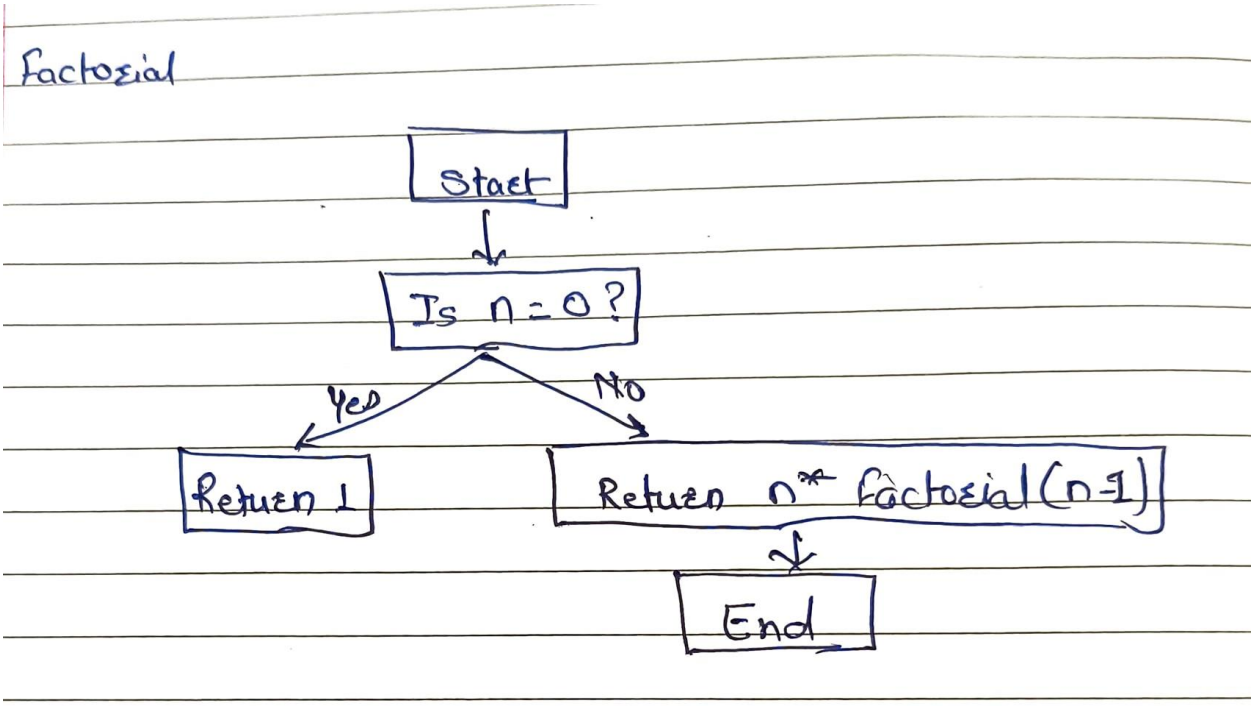
3. Factorial
Problem: Write a Java program to compute the factorial of a given number.
Code:

```
package problem3;

public class factorial {
	public static int fact (int n) {
		if (n==0)
			return 1;
		return n*fact(n-1);
	}
	public static void main(String[] args) {
		int num = 5;
		System.out.println("Factorial of " + num + " is: " + fact(num));

	}

}
```

O/p:
Factorial of 5 is: 120

Time complexity: O(n)
Space complexity: O(n)

Flowchart:



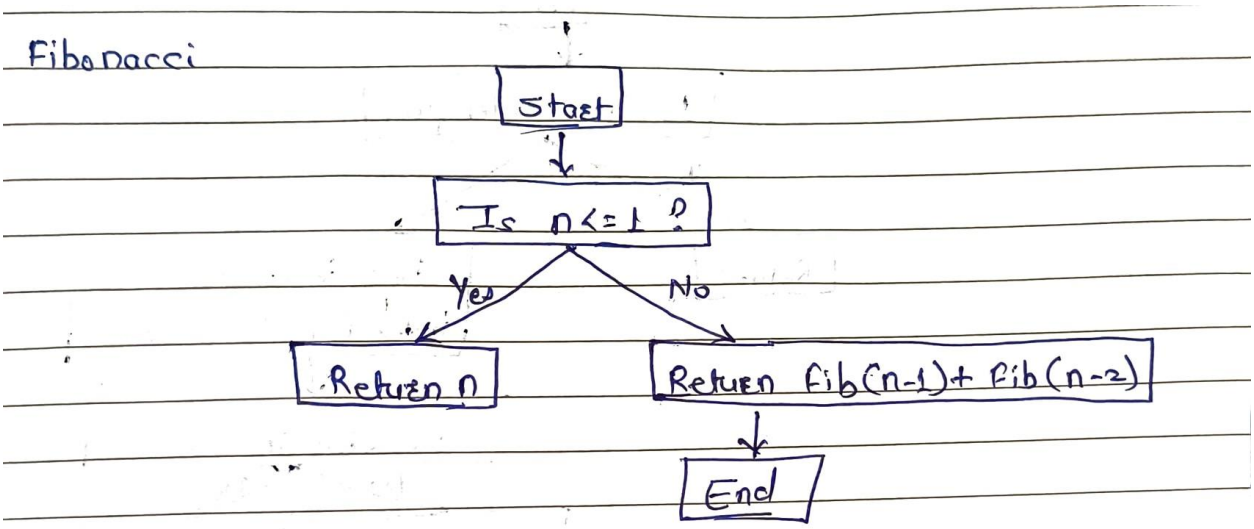
4. Fibonacci Series
Problem: Write a Java program to print the first n numbers in the Fibonacci series.
Code:

```
package problem4;

import java.util.ArrayList;
import java.util.List;

public class Fibonacci {
	public static int fib(int n ) {
		if(n==0) return 0;
		if(n==1) return 1;
		return fib(n-1) + fib(n-2);
	}

	public static List<Integer> fibonacciSeries(int n){
		List<Integer> series = new ArrayList<>();
		for(int i =0; i<n; i++) {
			series.add(fib(i));
		}
		return series;
	}
	public static void main(String[] args) {
		int n1 = 5;
		System.out.println("First " + n1 + "Fibonacci numbers: " + fibonacciSeries(n1));


		int n2 = 8;
		System.out.println("First " + n2 +" Fibonacci numbers: " + fibonacciSeries(n2));

	}

}
```

O/p:
First 5Fibonacci numbers: [0, 1, 1, 2, 3]
First 8 Fibonacci numbers: [0, 1, 1, 2, 3, 5, 8, 13]

Time complexity: $O(2^n)$
Space complexity: O(n)

Flowchart:



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

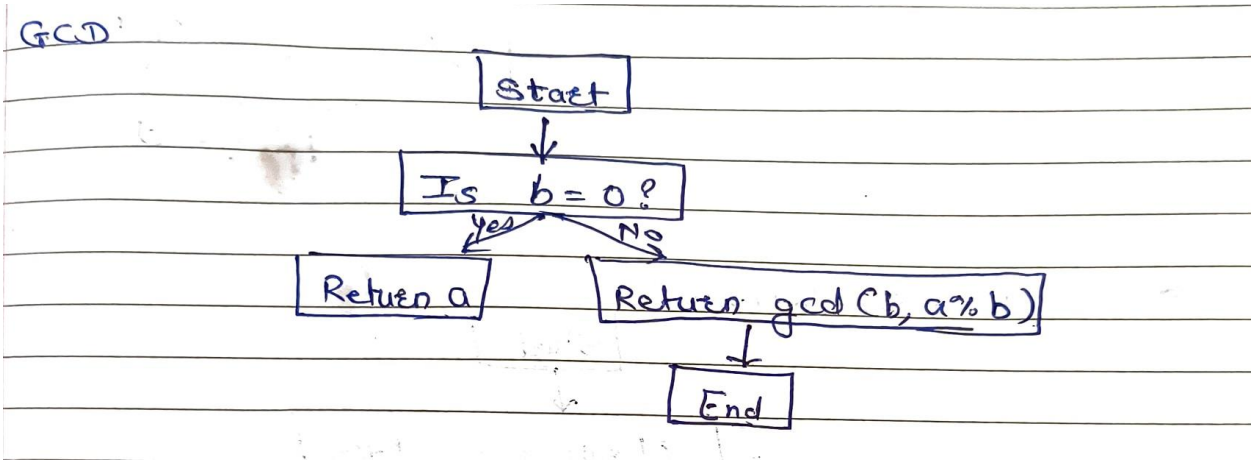
```
package problem5;

import java.util.Scanner;

public class GCD {

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

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

      System.out.print("Enter first number (a): ");
      int a = sc.nextInt();
      System.out.print("Enter second number (b): ");
      int b = sc.nextInt();
      if (a == 0 && b == 0) {
         System.out.println("GCD is undefined for both numbers being zero.");
      } else {
         System.out.println("GCD of " + a + " and " + b + " is: " + gcd(a, b));
      }
   }
}
```

O/p:

```
Enter first number (a): 17
Enter second number (b): 45
GCD of 17 and 45 is: 1
```

Time complexity: O(log(min(a,b)))
Space complexity: O(log(min(a,b)))

Flowchart:



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

```
package problem6;

public class SquareRoot {
	public static int sqrt (int x, int guess) {
		if (guess * guess >x) return guess -1;
	   return sqrt(x, guess +1);
	}

	public static int sqrt (int x) {
		return sqrt(x, 1);
	}
	public static void main(String[] args) {
		int x1 =16;
		System.out.println("Square root of " + x1 +" is: " + sqrt(x1));


		int x2 =27;
		System.out.println("Square root of " + x2 + " is: "+ sqrt(x2));

	}

}
```

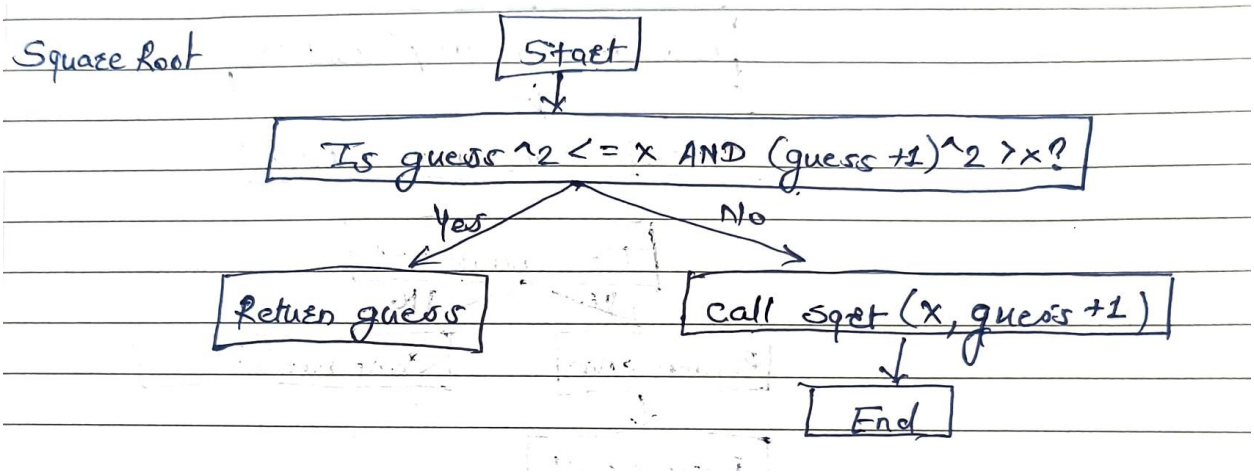
O/p:

```
Square root of 16 is: 4
Square root of 27 is: 5
```

Time complexity: $O(\sqrt{x})$

Space complexity: O($\sqrt{x}$)

Flowchart:



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

Code:

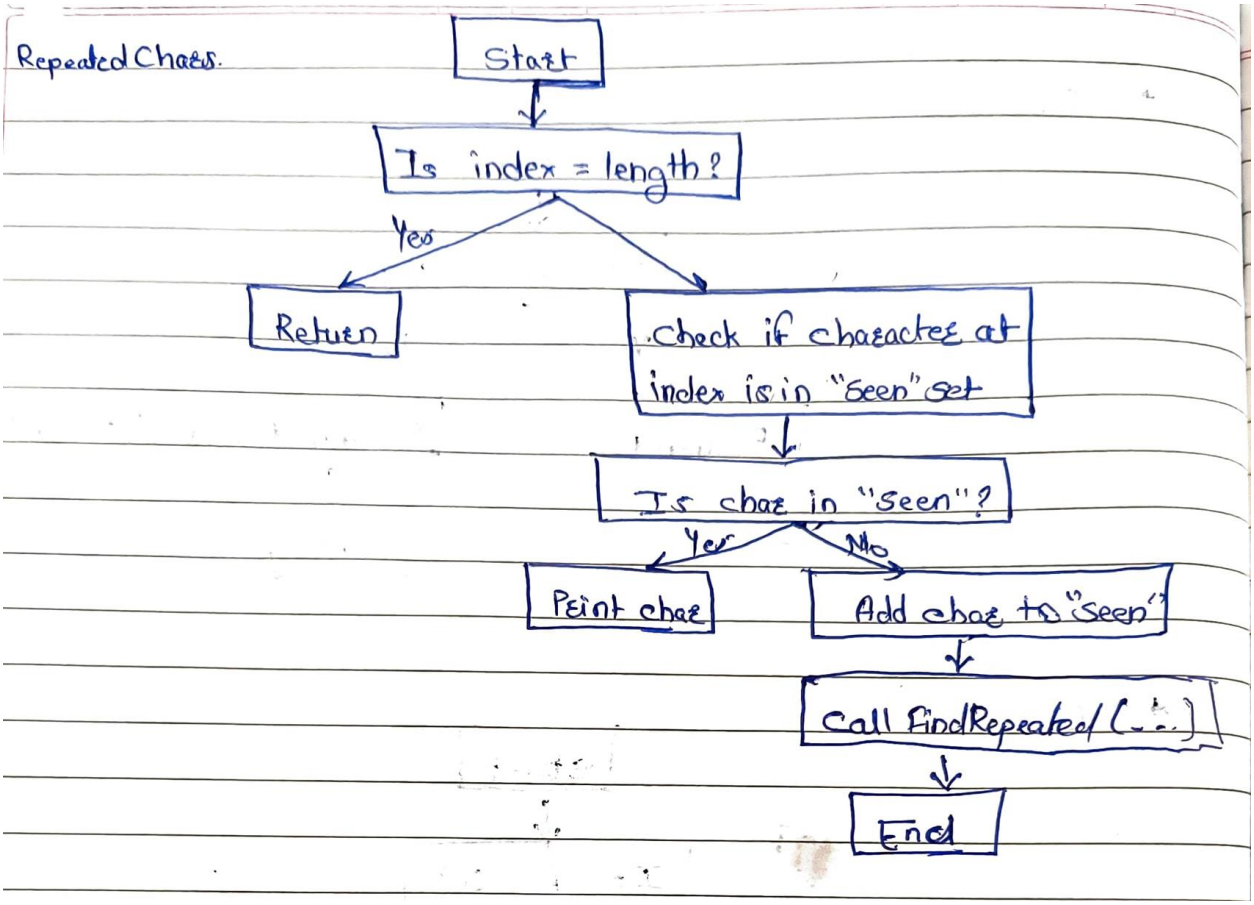
```
package problem7;
import java.util.HashSet;
public class repeatedChars {
	public static void findRepeated(String str, int index, HashSet<Character>seen) {
		if (index ==str.length()) return;

		char current = str.charAt(index);
		if(seen.contains(current)) {
			System.out.println(current + " ");
		}else {
			seen.add(current);
		}
		findRepeated(str, index + 1, seen);
	}
	public static void main(String[] args) {
		String input = "programming";

		HashSet<Character> seen =new HashSet<>();

		System.out.println("Repeated characters are: ");
		findRepeated(input, 0, seen);
	}

}
```

O/p:

```
Repeated characters are:
r
m
g
```

Time complexity: O(n)
Space complexity: O(n)

Flowchart:



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

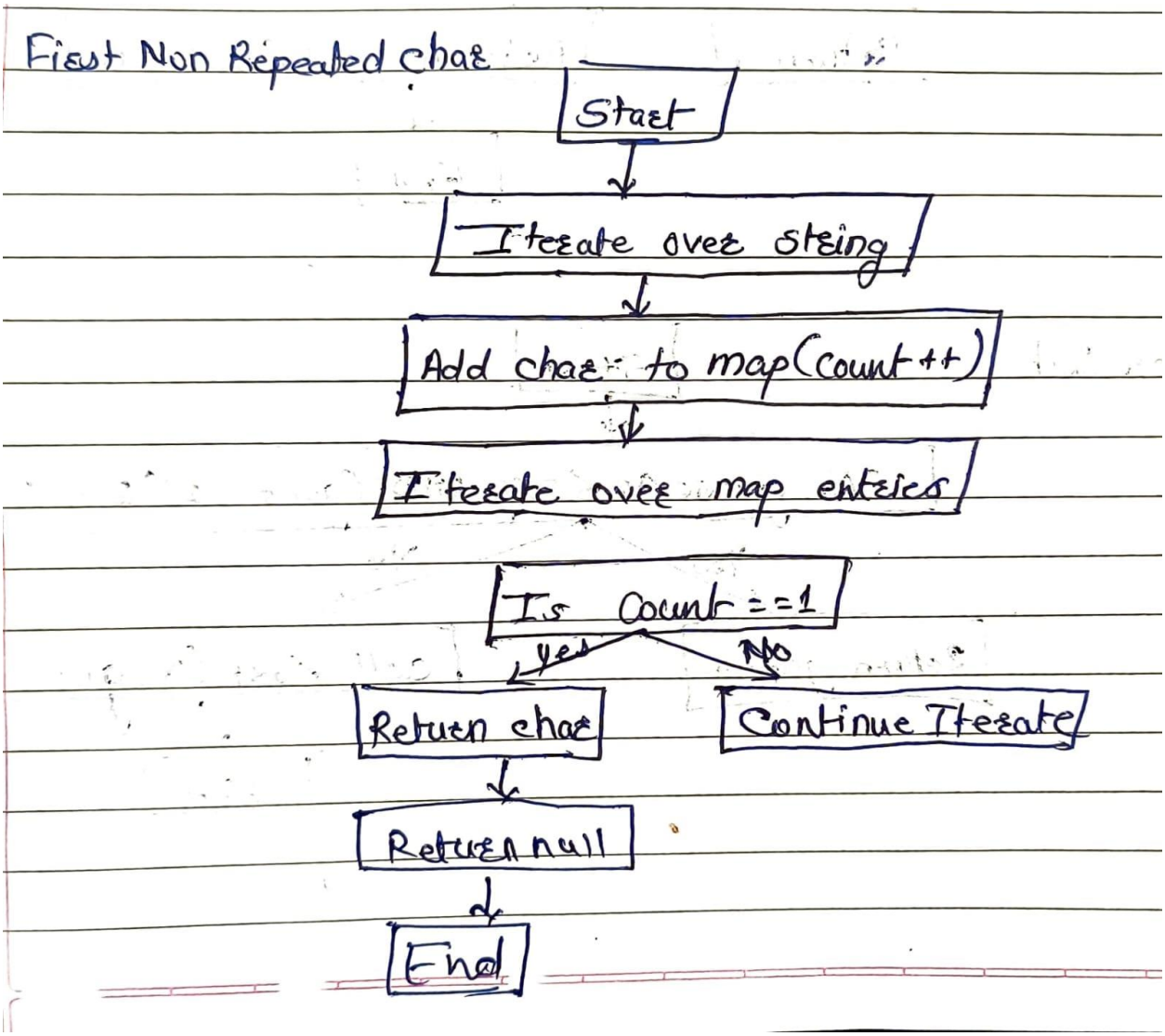
```
package problem8;
import java.util.LinkedHashMap;
import java.util.Map;

public class firstNonRepeatedChar {

	public static Character FirstNonRepeat(String str) {
		Map<Character, Integer> charCount = new LinkedHashMap<>();


		for(char c:str.toCharArray()) {
			charCount.put(c,charCount.getOrDefault(c, 0)+1);

		}
		for(Map.Entry<Character, Integer>entry : charCount.entrySet()) {
```

```
                        if(entry.getValue()==1) {
                                return entry.getKey();

                        }
                }
                return null;
        }

        public static void main(String[] args) {
                String input1 = "stress";
                String input2 = "aabbcc";

                System.out.println("First non-repeated character in '" + input1 + "' is: " +
FirstNonRepeat(input1));
                System.out.println("First non-repeated character in '" + input2 + "' is: " +
FirstNonRepeat(input2));
        }

}
```

O/p:
First non-repeated character in 'stress' is: t
First non-repeated character in 'aabbcc' is: null

Time complexity: O(n)
Space complexity: O(n)

Flowchart:



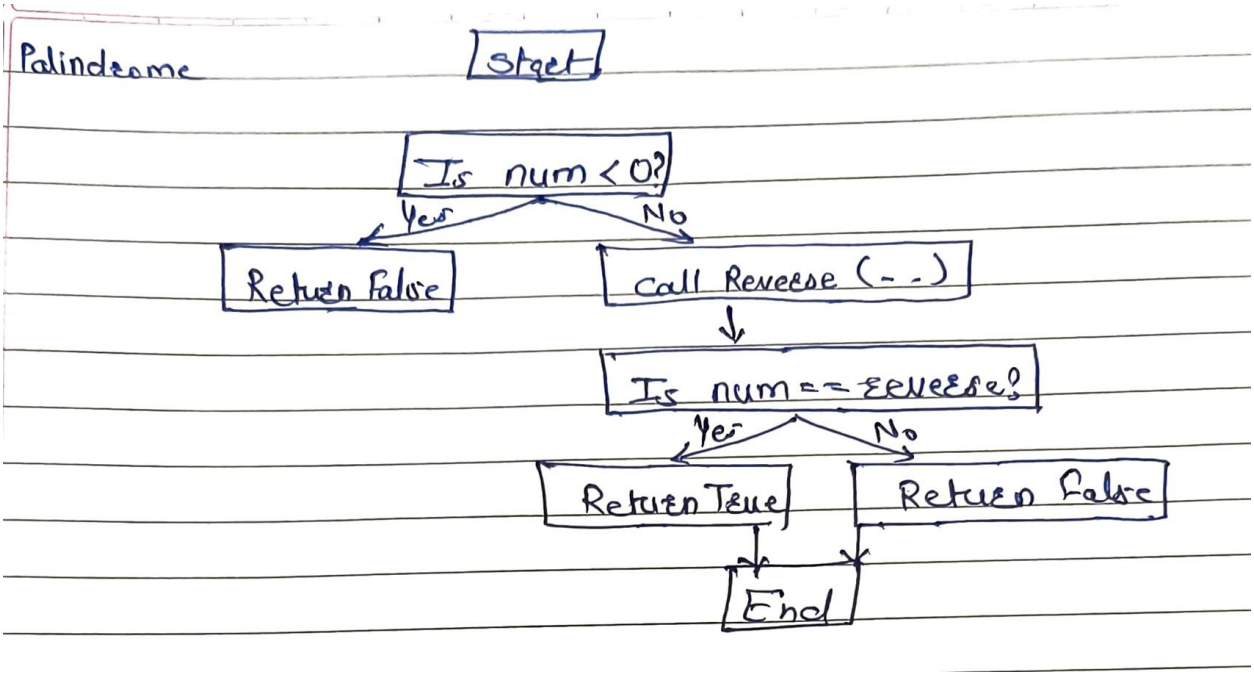
9. Integer Palindrome
Problem: Write a Java program to check if a given integer is a palindrome.
Code:

```
package problem9;

public class palindromeCheck {
        public static int reverse(int num, int rev) {
    if (num == 0) return rev;
    return reverse(num / 10, rev * 10 + num % 10);
  }


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

```
    public static void main(String[] args) {
        int num1 = 121;
        int num2 = -121;

        System.out.println(num1 + " is palindrome: " + isPalindrome(num1));
        System.out.println(num2 + " is palindrome: " + isPalindrome(num2));
    }

}
```

O/p:
121 is palindrome: true
-121 is palindrome: false
Time complexity: O(d)
Space complexity: O(d)

Flowchart:



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

```
package problem10;



public class leapYearCheck {

        public static boolean isLeapYear(int year) {
```

```
        if (year < 0) return false;
        return (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);
    }

    public static void main(String[] args) {
        int year1 = 2020;
        int year2 = 1900;

        System.out.println(year1 + " is a leap year: " + isLeapYear(year1));
        System.out.println(year2 + " is a leap year: " + isLeapYear(year2));
    }
}
```

O/p:

2020 is a leap year: true

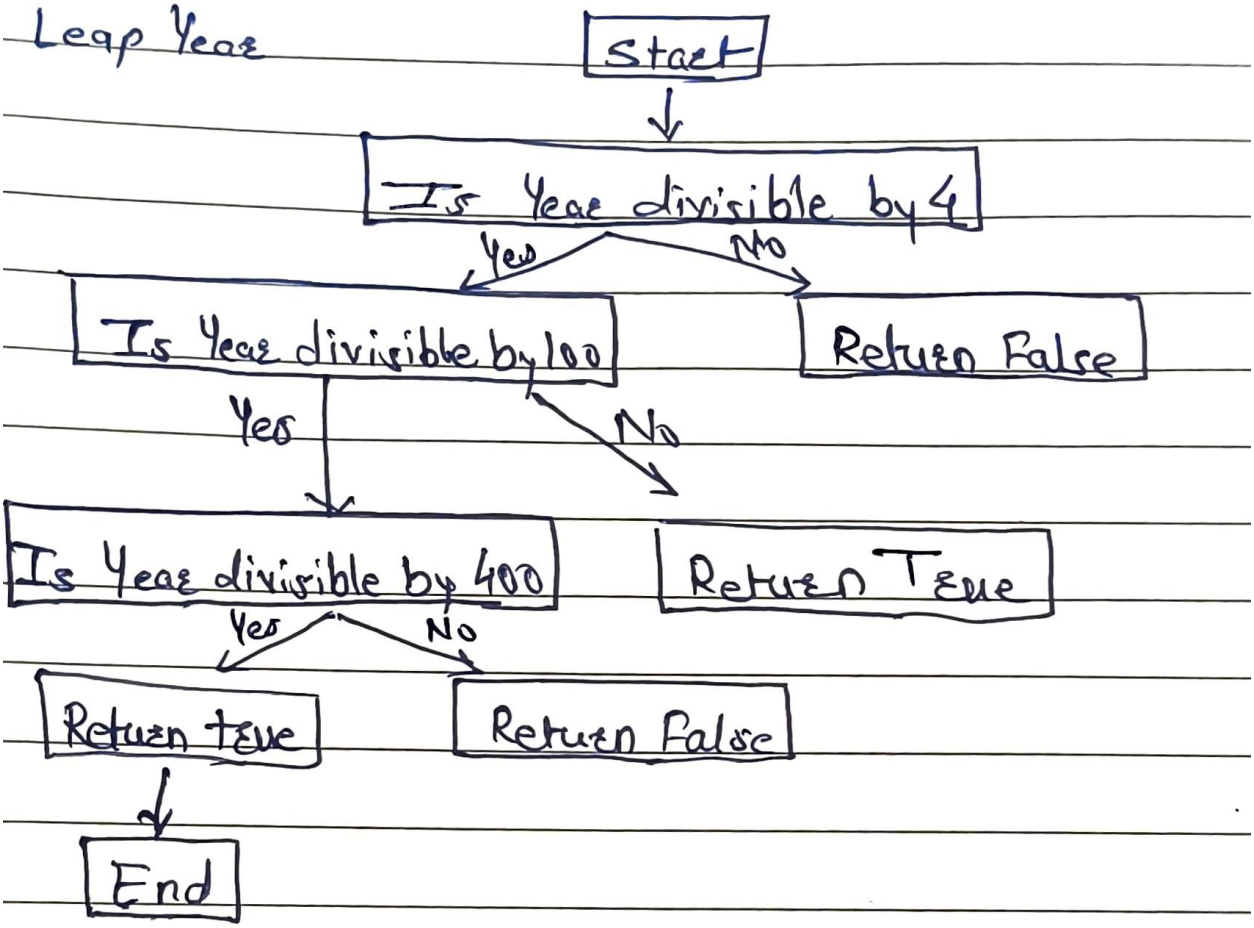
1900 is a leap year: false

Time complexity: O(1)

Space complexity: O(1)

Flowchart:

Leap Year


Start
Is Year divisible by 4
Yes
No
Is Year divisible by 100
Return False
Yes
No
Is Year divisible by 400
Return True
Yes
No
Return true
Return False
End