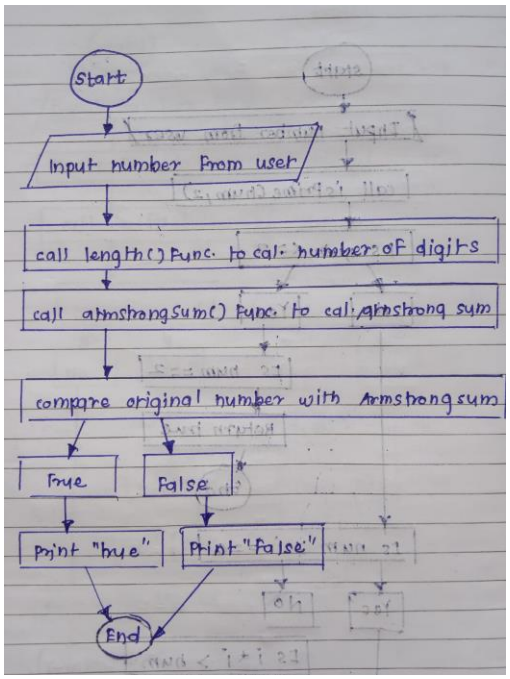
Assignment 1 Solution

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

Program -

```
public class ArmstrongNumRecursion {

   static int length(int num) {
      if (num == 0) {
         return 0;
      }
      return 1 + length(num / 10);
   }

   static int armstrongSum(int num, int len) {
      if (num == 0) {
         return 0;
      }
      int rem = num % 10;
      return (int) Math.pow(rem, len) + armstrongSum(num / 10, len);
   }

   public static void main(String[] args) {
      int num = 153;
      int len = length(num);
      int armstrongValue = armstrongSum(num, len);

      if (num == armstrongValue) {
         System.out.println("true");
      } else {
         System.out.println("false");
      }
   }
}
```

Flow chart -



Explanation -

1. Input: The program takes an integer input from the user.
2. Length Calculation (length() method): It recursively calculates the number of digits in the input number.
3. Armstrong Sum Calculation (armstrongSum() method): It recursively computes the Armstrong sum by raising each digit to the power of the number of digits.
4. Comparison: The program checks if the original number is equal to the calculated Armstrong sum.
5. Output: It prints true if the number is an Armstrong number, otherwise, it prints false

Time and Space complexity - O(n)

Output –

C:\Windows\system32\cmd.exe

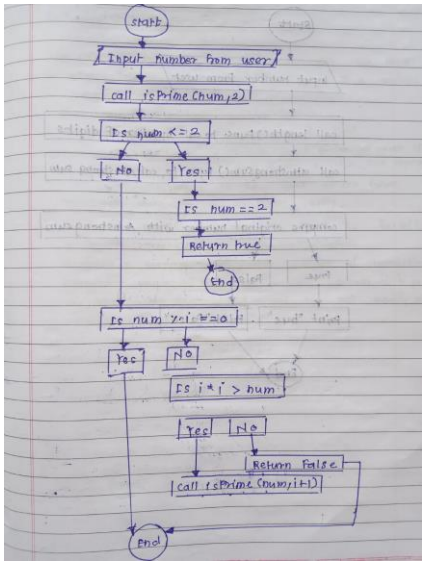
```
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>javac ArmstrongNumRecursion.java

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java ArmstrongNumRecursion
Enter teh number :
153
true

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java ArmstrongNumRecursion
Enter teh number :
135
false

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>
```

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

Program -

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

   static boolean isPrime(int num, int i) {
      if (num <= 2) {
         return num == 2;
      }
      if (num % i == 0) {
         return false;
      }
      if (i * i > num) {
         return true;
      }
      return isPrime(num, i + 1);
   }

   public static void main(String args[]) {

      Scanner sc = new Scanner(System.in);
                  System.out.println("Enter the number : ");
      int num = sc.nextInt();

      if (isPrime(num, 2)) {
         System.out.println("true");
      } else {
         System.out.println("false");
      }
   }
}
```

Flow chart -



Explanation -

1. Input: The program prompts the user to enter an integer.
2. Prime Check (isPrime() method):
3. It first checks if the number is less than or equal to 2. If it is 2, it returns true (2 is prime); otherwise, it returns false for numbers less than 2.
4. It checks if the number is divisible by the current divisor i. If it is, the number is not prime, and it returns false.
5. It checks if i * i is greater than num. If it is, the function concludes that the number is prime and returns true.
6. If the number is not yet determined to be prime, the function calls itself recursively, incrementing i by 1 to check the next potential divisor.
7. Output: The program prints true if the number is prime and false if it is not.

Time and Space complexity - O(√n)

Output –

C:\Windows\system32\cmd.exe

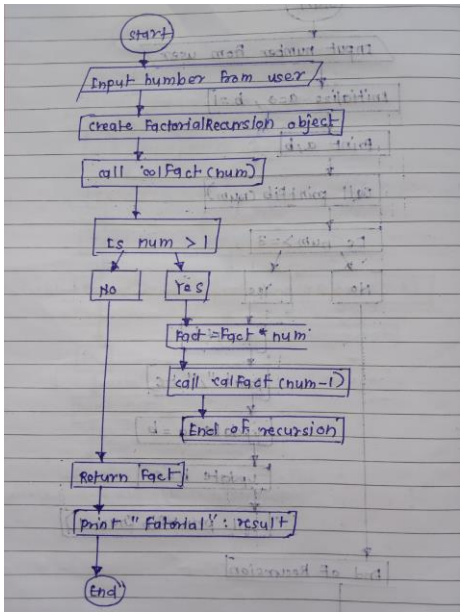
```
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>javac PrimeNumRecursion.java

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java PrimeNumRecursion
Enter the number :
7
true

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java PrimeNumRecursion
Enter the number :
6
false

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>
```

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

Program -

```
import java.util.Scanner;

public class FactorialRecursion{
        int fact = 1;

        public int calFact(int num){
                if(num > 1){
                        fact = fact * num;
                        calFact( num - 1 );
                }
                return fact;
        }

        public static void main(String args[]){

                Scanner sc = new Scanner(System.in);

                System.out.println("Enter the number for calculating factorial : ");
                int num = sc.nextInt();
                int result;
                FactorialRecursion f = new FactorialRecursion();
                result = f.calFact(num);
                System.out.println("Factorial of " + num + " : " + result );
        }
}
```

Flow chart –

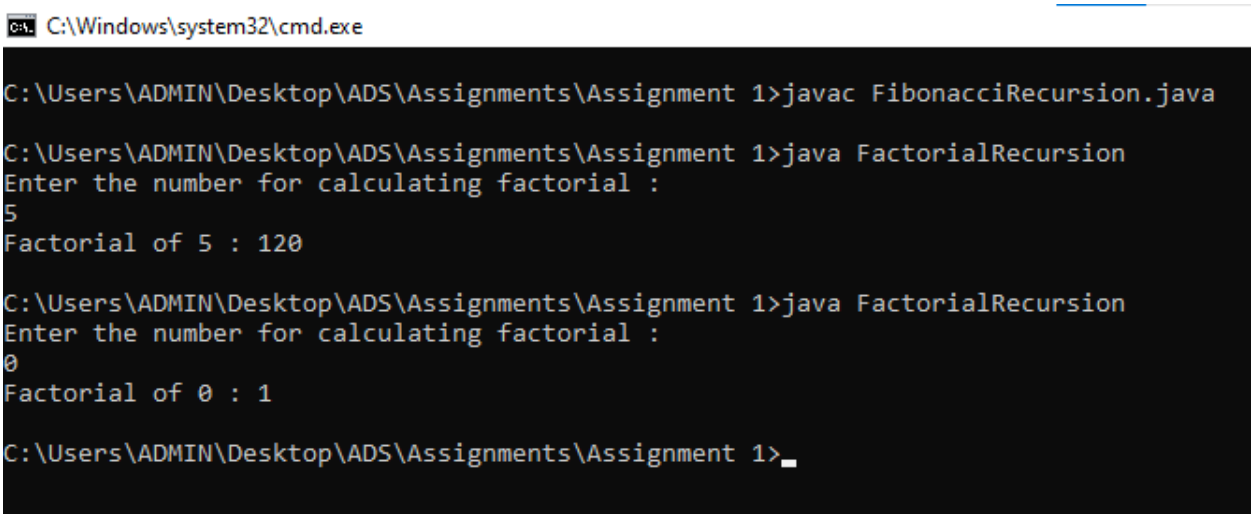


Explanation -

1. Input: The program prompts the user to enter an integer for which the factorial needs to be calculated.
2. Factorial Calculation (calFact() method):
3. The method checks if the input number is greater than 1. If so, it multiplies the current value of fact by num.
4. It then calls itself recursively with the value of num - 1 to continue calculating the factorial.
5. The recursion continues until num is no longer greater than 1.
6. Output: Once the recursion completes, the method returns the calculated factorial, which is then printed to the console.

Time and Space complexity - O(n)

Output –



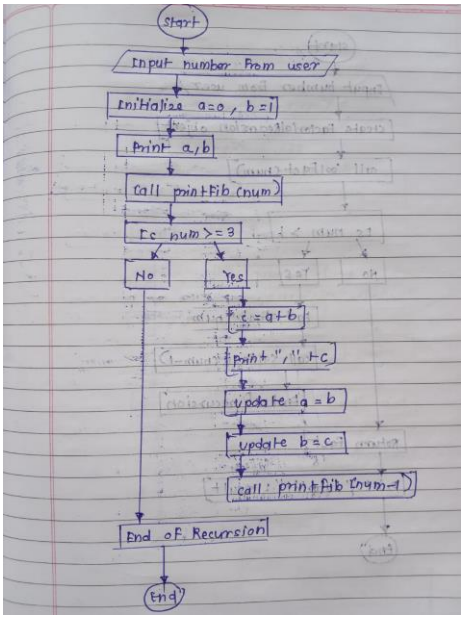
4. Fibonacci Series
Problem: Write a Java program to print the first n numbers in the Fibonacci series.

Program -

```
import java.util.Scanner;

public class FibonacciRecursion{

	static int a = 0, b=1;

	public void printFib(int num){
		int c;
		if(num >= 3){
			c = a + b;
			System.out.print("," + c);
```

```
                        a = b;
                        b = c;
                        printFib(num - 1);
                }
        }

        public static void main(String args[]){

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number for calculating fibonacci series : ");
        int num = sc.nextInt();

        System.out.print(a + "," + b);

        FibonacciRecursion obj = new FibonacciRecursion();
        obj.printFib(num);

        }
}
```

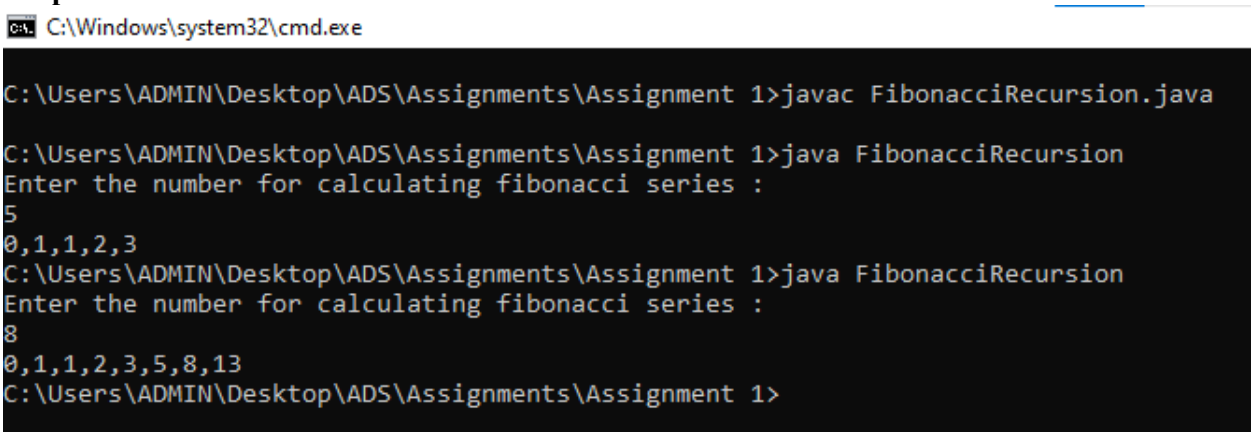
Flow chart -



Explanation -

1. Input: The program prompts the user to enter a number, representing how many terms of the Fibonacci sequence should be printed.
2. Initialization: The first two Fibonacci numbers (a = 0 and b = 1) are initialized and printed.
3. Recursive Fibonacci Calculation (printFib() method):
4. If num is greater than or equal to 3, it calculates the next Fibonacci number c as the sum of a and b.
5. It prints the new number, updates a and b to shift to the next two terms, and then recursively calls printFib() with num - 1 until the desired sequence length is reached.
6. Output: The Fibonacci series is printed, starting with 0 and 1, followed by the recursive computation of the remaining numbers.

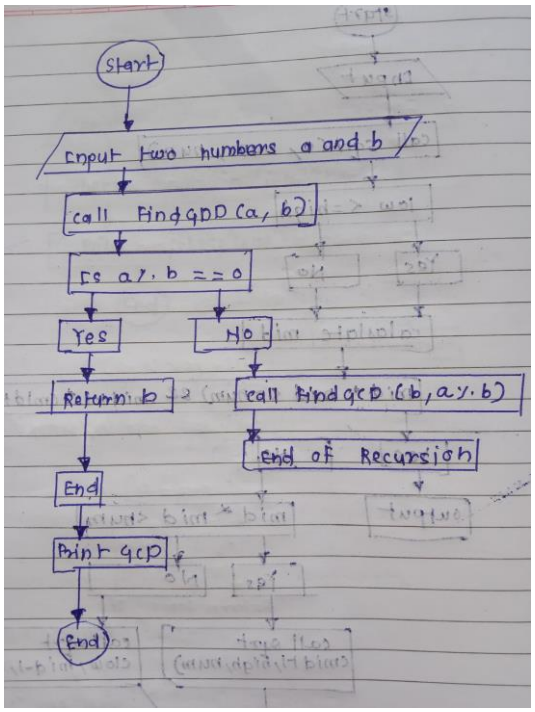
Time and Space complexity - O(n)
Output –



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

```
import java.util.Scanner;

public class GCDRecursion{

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

		if(a % b == 0){
			return b;
		}else{
			return findGCD(b, a%b);
		}

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

		Scanner sc = new Scanner(System.in);
		System.out.println("Enter two number : ");
		int a = sc.nextInt();
		int b = sc.nextInt();

		int result = findGCD(a, b);
		System.out.println("GCD : " + result);
	}

}
```

Flow chart -



Explanation -

1. Input: The program prompts the user to enter two integers.
2. GCD Calculation (findGCD() method):
3. This method implements the Euclidean algorithm to calculate the GCD.
4. It checks if a is divisible by b. If true, it returns b as the GCD.
5. If not, it recursively calls itself with b and the remainder of a divided by b (a % b).
6. Output: The GCD of the two numbers is calculated and printed to the console.

Time and Space complexity - O(n)

Output –

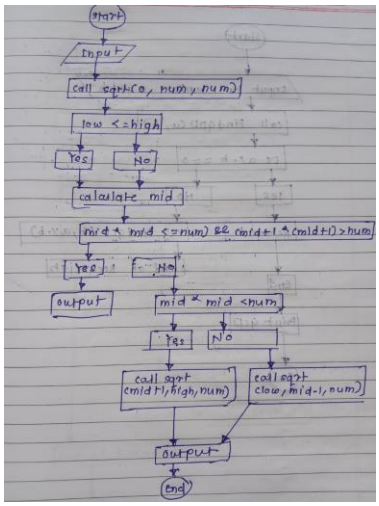
```
C:\Windows\system32\cmd.exe

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>javac GCDRecursion.java

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java GCDRecursion
Enter two number :
54
24
GCD : 6

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java GCDRecursion
Enter two number :
17
13
GCD : 1

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>
```

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

Program -

```
import java.util.Scanner;

public class SquareRootRecursion{

	static int sqrt(int low, int high, int num){

		if(low <= high){
			int mid = (int) (low + high) / 2;

			if((mid * mid <= num) && ((mid + 1) * (mid + 1) > num)){
				return mid;
			}
			else if(mid * mid < num){
				return sqrt(mid + 1, high, num);
			}
			else{
				return sqrt(low, mid-1, num);
			}
		}
		return low;

	}

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

		System.out.println("Enter the number : ");
		int num = sc.nextInt();
		System.out.println("Square root of " + num + " : " + sqrt(0, num, num));

	}


}
```

Flow chart -



Explanation -

1. Class Definition: The class SquareRootRecursion contains methods to calculate the square root of a number using recursion.
   - Recursive Method sqrt(int low, int high, int num):
   - Parameters:
   - low: The lower bound of the search range.
   - high: The upper bound of the search range.
   - num: The number for which the square root is being calculated.
2. Base Case: If low exceeds high, it returns low.
   - Mid Calculation: It calculates the midpoint (mid) of the current range.
3. Check Conditions:
   - If mid * mid is less than or equal to num and (mid + 1) * (mid + 1) is greater than num, it returns mid as the integer square root.
   - If mid * mid is less than num, it calls sqrt with the upper half of the range (mid + 1).
   - Otherwise, it calls sqrt with the lower half of the range (low to mid - 1).
4. Main Method:
   - Uses a Scanner to take user input for the number.
   - Calls the sqrt method with initial bounds 0 and num and prints the calculated square

Time and Space complexity - O(logn)

Output –

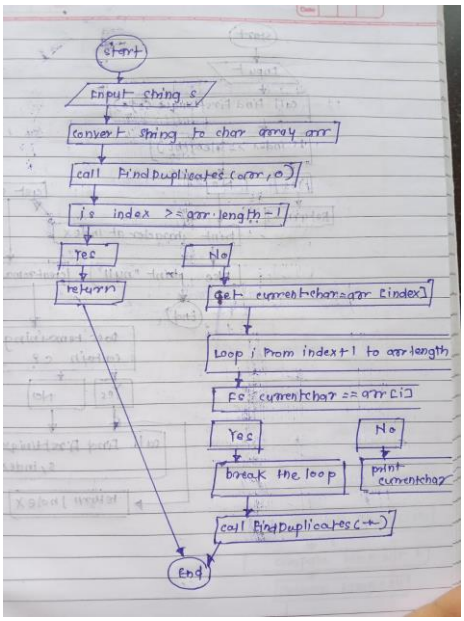
```
C:\Windows\system32\cmd.exe

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>javac SquareRootRecursion.java

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java SquareRootRecursion
Enter the number :
16
Square root of 16 : 4

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java SquareRootRecursion
Enter the number :
27
Square root of 27 : 5

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>
```

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

Program -

```
import java.util.Scanner;

public class DuplicateCharRecursion {

   public static void findDuplicates(char[] arr, int index) {
      if (index >= arr.length - 1) {
         return;
      }

      char currentChar = arr[index];

      for (int i = index + 1; i < arr.length; i++) {
         if (currentChar == arr[i]) {
            System.out.print(currentChar + " ");
            break;
         }
      }


      findDuplicates(arr, index + 1);
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
                  System.out.println("Enter String : ")
      String s = sc.next();
      char[] arr = s.toCharArray();

      findDuplicates(arr, 0);
   }
}
```

Flow chart -



Explanation -

1. Class Definition: The class DuplicateCharRecursion contains methods to find and print duplicate characters in a string using recursion.
   - Method findDuplicates(char[] arr, int index):
   - Parameters:
   - arr: The character array containing the characters from the input string.
   - index: The current index being checked for duplicates.
2. Base Case: If index is greater than or equal to arr.length - 1, the method returns, ending the recursion.
   - Current Character: It assigns the character at the current index (currentChar).
   - Inner Loop: It iterates through the characters starting from index + 1:
   - If a character matches currentChar, it prints currentChar and breaks out of the loop.
3. Main Method:
   - Uses a Scanner to take user input for the string.
   - Converts the input string to a character array and calls findDuplicates(arr, 0) to initiate the duplicate finding process.
   -

Time Complexity - O(n)
Space complexity – $O(n^2)$
Output –

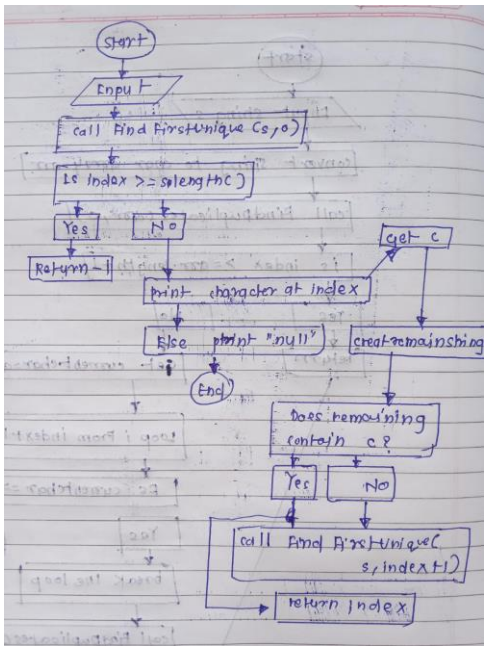
```
C:\Windows\system32\cmd.exe

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>javac DuplicateCharRecursion.java

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java DuplicateCharRecursion
Enter String :
programming
r g m
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java DuplicateCharRecursion
Enter String :
hello
l
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>
```

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

Program -

```
import java.util.Scanner;

class NonRepeatedCharRecursion {

   public static int findFirstUnique(String s, int index) {
      if (index >= s.length()) {
         return -1;
      }

      char c = s.charAt(index);
      String remainingString = s.substring(0, index) + s.substring(index + 1);

      if (!remainingString.contains(String.valueOf(c))) {
         return index;
      } else {
         return findFirstUnique(s, index + 1);
      }
   }

   public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
                     System.out.println("Enter string : ");
      String s = sc.next();

      int index = findFirstUnique(s, 0);

      if (index != -1) {
         System.out.println(s.charAt(index));
      } else {
         System.out.println("null");
      }
   }
}
```

Flow chart -



Explanation -

1. Purpose: The code finds the first non-repeated character in a string using recursion.
2. Method findFirstUnique(String s, int index):
3. Base Case: Returns -1 if index exceeds the string length.
4. Logic: Retrieves the character at index, removes it from the string, and checks if it exists in the remaining string. If not, returns the index; otherwise, recursively calls itself with the next index.
5. Main Method: Takes user input, calls findFirstUnique, and prints the first non-repeated character or "null" if none exists.

Time Complexity - $O(n^2)$
Space complexity – $O(n)$
Output –

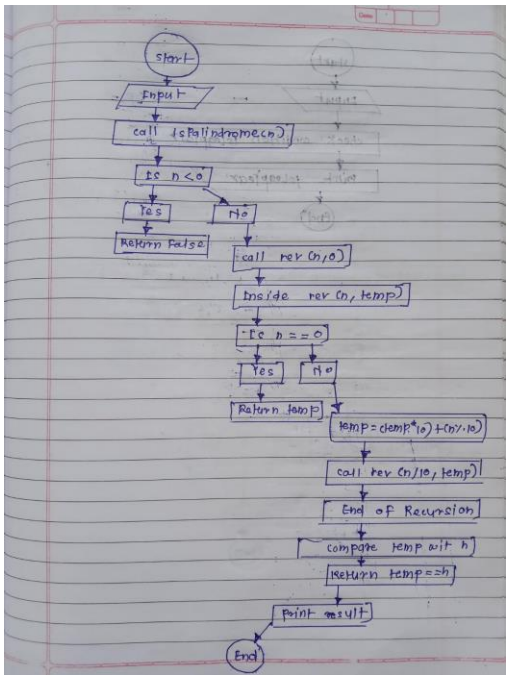
```
C:\Windows\system32\cmd.exe

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>javac NonRepeatedCharRecursion.java

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java NonRepeatedCharRecursion
Enter string :
stress
t

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java NonRepeatedCharRecursion
Enter string :
aabbcc
null

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>
```

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

Program -

```
import java.util.Scanner;

public class IntPalindromeRecursion
{
   static int rev(int n, int temp)
   {
      if (n == 0)
         return temp;

      temp = (temp * 10) + (n % 10);
      return rev(n / 10, temp);
   }

   static boolean isPalindrome(int n)
   {
      if (n < 0)
         return false;

      int temp = rev(n, 0);

      return temp == n;
   }

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

      boolean result = isPalindrome(n);

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

Flow chart -



Explanation -

1. Purpose: The code checks if a given integer is a palindrome using recursion.
2. Method rev(int n, int temp):
   - Reverses the integer n recursively. If n becomes 0, it returns the reversed number (temp).
   - Constructs the reversed number by taking the last digit of n and appending it to temp.
3. Method isPalindrome(int n):
   - Checks if the integer is negative; if so, it returns false.
   - Calls rev(n, 0) to get the reversed number and compares it with the original number.
4. Main Method:
   - Takes user input for an integer and calls isPalindrome to check if it's a palindrome.
   - Prints the result (true or false).

Output -

C:\Windows\system32\cmd.exe

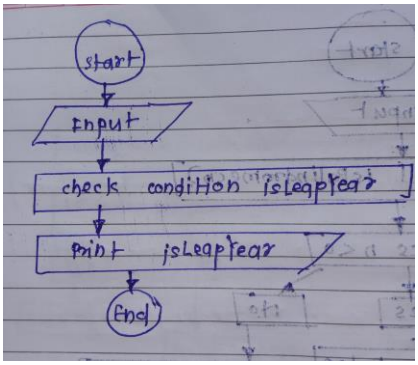
```
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>javac IntPalindromeRecursion.java

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java IntPalindromeRecursion
Enter the number :
121
true

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java IntPalindromeRecursion
Enter the number :
-121
false

C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>
```

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

Program -

```
import java.util.Scanner;

public class LeapYear {

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

        System.out.println("Enter the year to check whether it is a leap year or not: ");
        int year = sc.nextInt();

        // Corrected leap year condition
        boolean isLeapYear = (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

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

Flow chart -

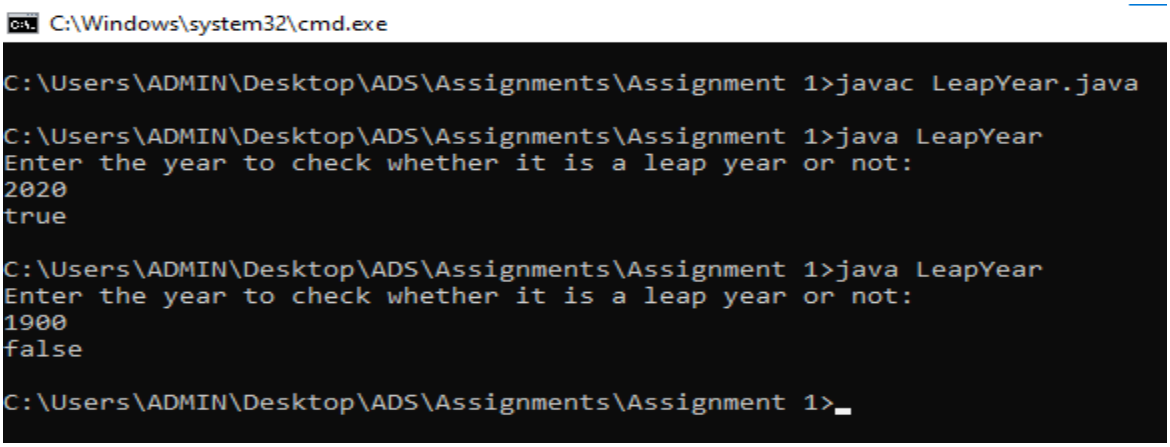


Explanation -

1. Purpose: The code checks if a given year is a leap year.
2. Main Method:
    - Uses a Scanner to take user input for the year.
    - The leap year condition is evaluated using the formula:
    - A year is a leap year if it is divisible by 4 but not divisible by 100, or it is divisible by 400.
    - The result (true or false) indicating whether the year is a leap year is printed to the console.

Time and Space complexity - O(1)

Output –


C:\Windows\system32\cmd.exe
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>javac LeapYear.java
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java LeapYear
Enter the year to check whether it is a leap year or not:
2020
true
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>java LeapYear
Enter the year to check whether it is a leap year or not:
1900
false
C:\Users\ADMIN\Desktop\ADS\Assignments\Assignment 1>