**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.

**import** java.util.Scanner;

**public** **class** ArmstrongNo {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("Enter Number:");

**int** n= sc.nextInt();

**int** result=0,rem;

**int** n1=n;

**while**(n>0)

{

rem=n%10;

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

n=n/10;

}

**if**(n1==result) {

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

}

**else** {

System.***out***.println(result+" is Not a Armstrong Number");

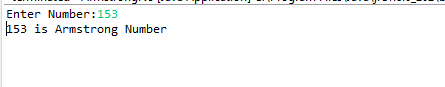
}

sc.close();

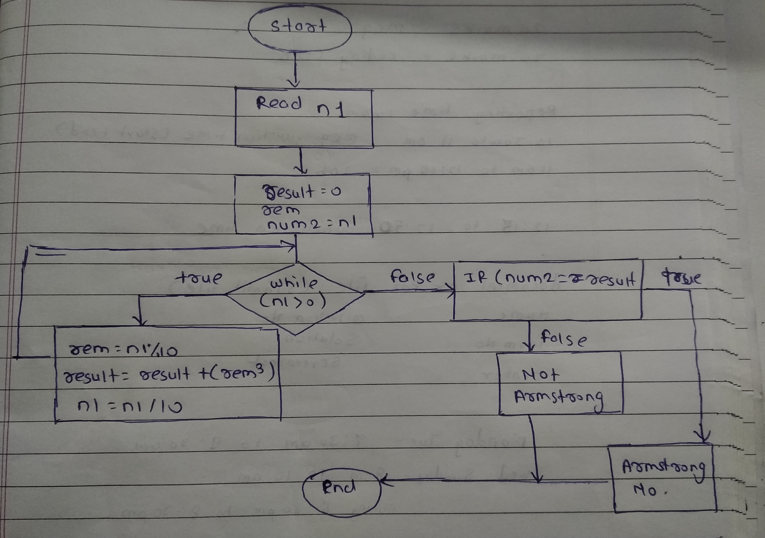
}

}

**Output:-**



**Flowchart:-**

****

**Explanation:-**

1. **Start**
2. User inputs a number.
3. The program extracts digits from the number, cubes each digit, and sums them.
4. The computed sum is compared to the original number.
5. If they match, the input is an Armstrong number; otherwise, it is not.
6. The program ends.

 **Time Complexity**: O(log10n)

 **Space Complexity**: O(1)

2. Prime Number

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

**import** java.util.Scanner;

**public** **class** PrimeNo {

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

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

System.***out***.println("Enter a number:");

**int** n=sc.nextInt();

**boolean** flag= **false**;

**if**(n==0 || n==1) {

flag=**true**;

}

**for** (**int** i = 2; i <= n / 2; ++i) {

**if** (n % i == 0) {

flag = **true**;

**break**;

}

}

**if** (!flag)

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

**else**

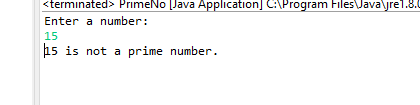
System.***out***.println(n + " is not a prime number.");

sc.close();

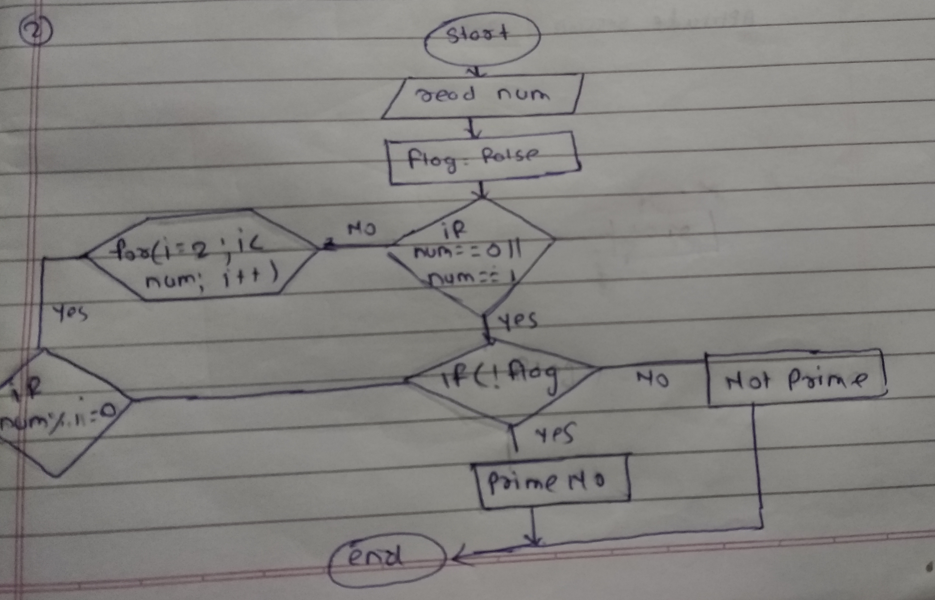
}

}

**Output:-**



**Flowchart:-**

****

**Explanation:-**

1. **Start**
2. Enter number n
3. The program checks if n is 0 or 1
4. For numbers greater than 1, it checks if the number has any divisors from 2 to n / 2.
5. If no divisors are found, the number is prime; otherwise, it’s not.
6. The program displays the result and terminates.

 **Time Complexity:**

* Worst Case: O(n)
* Best Case: O(1)

 **Space Complexity:** O(1)

3. Factorial

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

**import** java.util.Scanner;

**public** **class** FactorialNo {

**public** **static** **int** fact(**int** n) {

**if**(n<=1) {

**return** n;

}

**return** n\**fact*(n-1);

}

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

// **TODO** Auto-generated method stub

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

System.***out***.print("Enter number:");

**int** n = sc.nextInt();

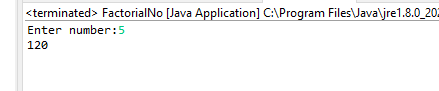
System.***out***.println(*fact*(n)+ " ");

sc.close();

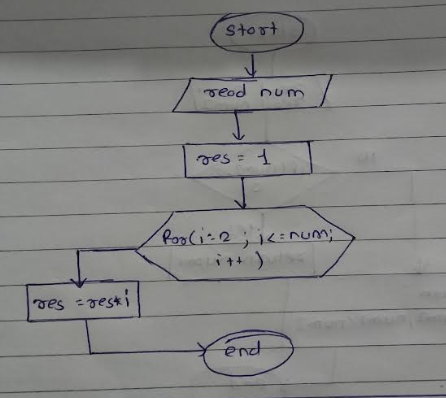
}

}

**Output:-**



**Flowchart:-**

****

**Explanation:-**

1. **Start**
2. Input integer n
3. **Call Factorial Method (**fact(n)**)**

* **If** n <= 1

Return n (base case)

* **Else**

Return n \* fact(n - 1) (recursive case)

1. **Print the result:** "Factorial of n is: fact(n)"
2. End

 **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.

**import** java.util.Scanner;

**public** **class** Fibonacci {

**public** **static** **int** fibo(**int** n) {

**if**(n<=1) {

**return** n;

}

**return** *fibo*(n-1)+*fibo*(n-2);

}

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

// **TODO** Auto-generated method stub

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

System.***out***.println("Enter a Number:");

**int** num=sc.nextInt();

**for**(**int** i=0;i<=num;i++) {

System.***out***.print(*fibo*(i)+" ");

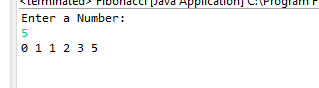
}

sc.close();

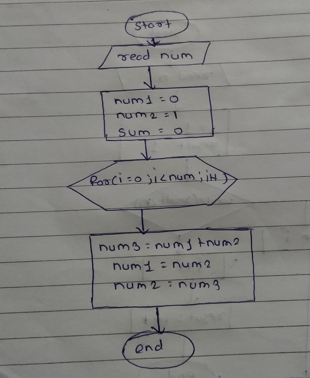
}

}

**Output:-**



**Flowchart:-**



**Explanation :-**

1. **Start**
2. Enter a number which det
3. The program loops from 0 to num, calling the recursive fibo() function for each term.
4. The Fibonacci sequence is generated and printed recursively.
5. The program ends after displaying the sequence.

 **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.

**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) {

// **TODO** Auto-generated method stub

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

System.***out***.println("Enter two numbers:");

**int** a = sc.nextInt();

**int** b = sc.nextInt();

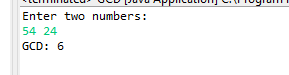
System.***out***.println("GCD: " + *gcd*(a, b));

sc.close();

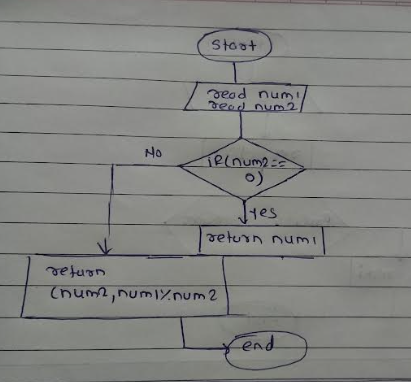
}

}

**Output:-**



**Flowchart:-**

****

**Explanation:-**

1. **Start**

2. Input two integers a and b

3. **Call GCD method (**gcd(a, b)**):**

- **If** b == 0

Return a as the GCD

* **Else:**

Call gcd(b, a % b)

* Continue recursively until b == 0

4. **Print the result:** "GCD: " + gcd(a, b)

5. End

 **Time Complexity**: O(log(min(a,b)))

 **Space Complexity**: O(log(min(a,b)))

6. Find Square Root

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

**import** java.util.Scanner;

**public** **class** SquareRoot {

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

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

System.***out***.println("Enter Number:");

**int** i = sc.nextInt();

**int** squareroot= (**int**) Math.*sqrt*(i);

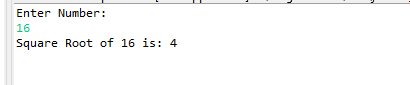
System.***out***.println("Square Root of "+i+" is: " +squareroot);

sc.close();

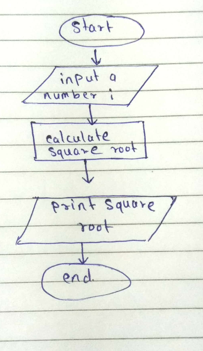
}

}

**Output:-**

****

**Flowchart:-**



**Explanation:-**

1. **Start**
2. User inputs a number (i).
3. The program calculates the square root of i using Math.sqrt(i) and converts it to an integer.
4. The square root is displayed in the output.
5. The program ends.

 **Time Complexity:** O(log⁡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.

**import** java.util.Scanner;

**public** **class** RepeateChar {

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

// **TODO** Auto-generated method stub

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

System.***out***.print("Enter a String: ");

String str=sc.nextLine();

**char** [] char1=str.toCharArray();

System.***out***.print("Duplicate Character in Above String are: ");

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

{

**for**(**int** j=i+1;j<str.length();j++) {

**if**(char1[i]==char1[j]) {

System.***out***.print(char1[j]+ " ");

**break**;

}

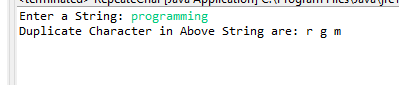
}

}

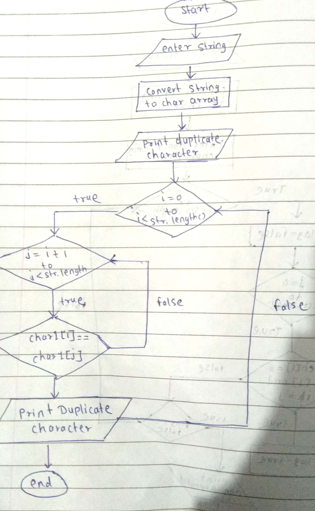
sc.close();

}}

**Output:-**

****

**Flowchart:-**

**  **

**Explanation:-**

1. **Start**
2. Enter string str
3. Convert string into character array char1[]
4. Print the message: "Duplicate Character in Above String are: "
5. **Outer loop (for each character** char1[i]**):**
6. **Inner loop (compare** char1[i] **with** char1[j]**, where** j = i + 1**):**

* If char1[i] == char1[j], print char1[j] and break

1. End

8. First Non-Repeated Character

Problem: Write a Java program to find the first non-repeated character in a string.

**import** java.util.Scanner;

**public** **class** NoRepeateChar {

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

// **TODO** Auto-generated method stub

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

System.***out***.println("Enter a String:");

String str= sc.nextLine();

**char** []ch=str.toCharArray();

System.***out***.print("Enter Non Repeated Character in above String:");

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

{

**boolean** flag=**false**;

**for**(**int** j=0;j<str.length();j++)

{

**if**(i !=j && ch[i]==ch[j])

{

flag=**true**;

**break**;

}

}

**if**(!flag)

{

System.***out***.print(ch[i]+" ");

}

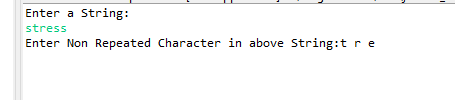
}

sc.close();

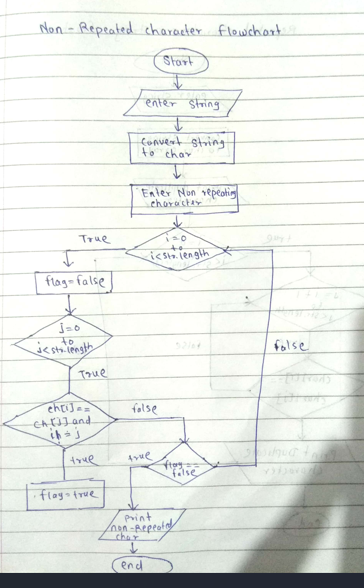
}

}

**Output:-**



Flowchart:-

****

**Explanation:-**

1. **Start**
2. Enter string str
3. Convert the string into a character array ch[]
4. Print the message: "Enter Non Repeated Character in above String"
5. **Outer loop (for each character ch[i])**

* Initialize flag = false

1. **Inner loop (compare ch[i] with ch[j])**

* If ch[i] == ch[j] and i != j, set flag = true and break

1. **Check for non-repetition:**

* If flag == false,
* print the character ch[i]

1. End

9. Integer Palindrome

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

**import** java.util.Scanner;

**public** **class** Palindrome {

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

// **TODO** Auto-generated method stub

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

System.***out***.println("Enter Number:");

**int** n= sc.nextInt();

**int** rem,sum=0,temp;

temp=n;

**while**(n>0) {

rem=n%10;

sum=(sum\*10)+rem;

n=n/10;

}

**if**(temp==sum) {

System.***out***.println("Palindrome");

}

**else** {

System.***out***.println("Not Palindrome");

}

sc.close();

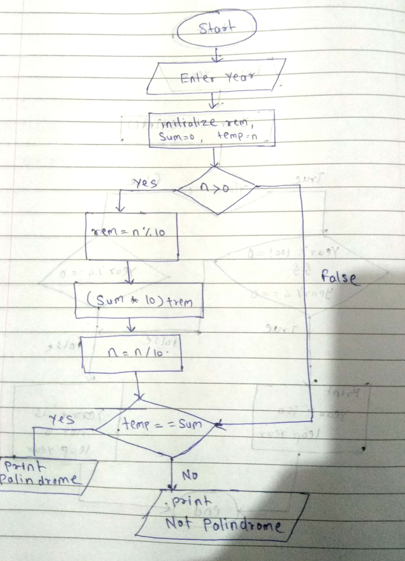
}

}

**Output:-**



**Flowchart:-**

****

**Explanation:-**

1. Start
2. Enter integer n
3. Store original number temp = n
4. Initialize sum = 0
5. **By using loop Reverse the digits**

* While n > 0 then Calculate rem = n % 10, update sum = (sum \* 10) + rem, reduce n = n / 10
* If condition false then Exit the loop

1. **Compare** temp **and** sum
2. If temp == sum then Print "Palindrome" otherwise if condition false then Print "Not Palindrome"
3. End

 **Time Complexity**: O(log⁡10(n))

 **Space Complexity**: O(1)

10. Leap Year

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

**import** java.util.Scanner;

**public** **class** LeapYear {

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

// **TODO** Auto-generated method stub

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

System.***out***.println("Enter Year:");

**int** year= sc.nextInt();

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

{

System.***out***.println(year + " is a Leap Year");

}

**else**

{

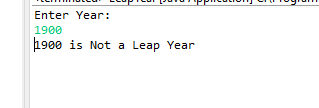
System.***out***.println(year + " is Not a Leap Year");

}

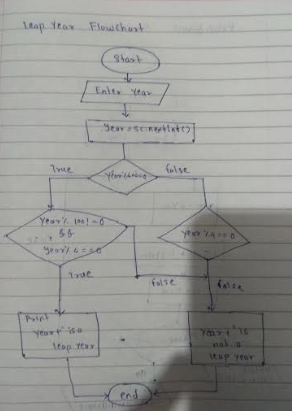
sc.close();

}

**Output:-**



**FlowChart:-**

****

**Explanation:**

1. Start
2. Enter year
3. Check the condition

If year divisible by 400 or divisible by 4 and not divisible by 100 ,then Print "Leap

Year" otherwise if condition is false then Print "Not a Leap Year"

1. End Program

 **Time Complexity:** O(1)

 **Space Complexity:** O(1)