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

**Subject: Algorithm and Data Structure**

**Assignment 1**

**Name :- Avdhoot Giirish Pol**

**Student PRN :- 240840520018**

**Centre:-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.

Program:

**package** Com.in;

**import** java.util.Scanner;

**public** **class** AmstrongNum {

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

**int** Number;

**int** digit;

**int** cubesum = 0;

**int** num;

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

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

num = sc.nextInt();

Number = num;

**while** (num!=0) {

digit=num%10;

cubesum += Math.*pow*(digit, 3);

num /= 10;

}

**if** (cubesum==Number)

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

**else**

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

sc.close();

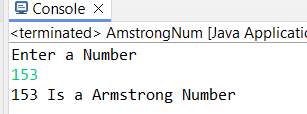
}

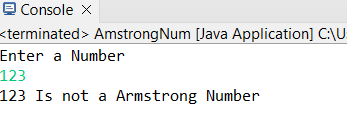
}

Flowchart:-

Explanation:-In the Armstrong number the given number sum of cubes of individual number must be equal to the original number. For this I have taken four integer in which the Number stores original number and digit will contain individual single number .Cubesum will take value of sum of cubes of three numbers. In given program there is while loop which starts if the condition is true then digit will seprate each separately and there cubes will be taken using math.pow function. In the If Condition if cubesum becomes equal to original number then it is Armstrong number.Else it will be not an Armstrong Number.

Output:-





Time Complexity:-O(n)

Test Cases:

Input: 153

Output: true

Input: 123

Output: false

2. Prime Number

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

Program:-

**package** Com.in;

**import** java.util.Scanner;

**public** **class** PrimeNumber {

**public** **static** **boolean** isPrime(**int** num) {

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

**return** **false**;

}

**for** (**int** i = 2; i <= Math.*sqrt*(num); i++) {

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

**return** **false**;

}

}

**return** **true**;

}

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

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

System.***out***.print("Enter a number to check if it is prime: ");

**int** number = scanner.nextInt();

**if** (*isPrime*(number)) {

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

} **else** {

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

}

scanner.close();

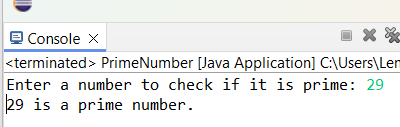
}

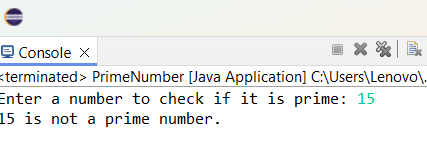
}

Flowchart:-

Explanation:-To Check whether the number is prime or not I have used first if condition to check whether the number is smaller than i it will return false because number less than 1 is not prime number If it is greater then it will enter for loop where I iterates from 2 to the square root of number. If condition check if the number is divisible by I then it is not a prime number. If it does not return remainder then it is prime number. In the main method I have call Isprime function to check I f the number is prime then it will print Number is prime or else it is not prime.

Output:-





Time Complexity:- O(√N)

Test Cases:

Input: 29

Output: true

Input: 15

Output: false

3. Factorial

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

Program:-

package Com.in;

import java.util.Iterator;

import java.util.Scanner;

public class Factorial {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

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

int Number = sc.nextInt();

int factorial=1;

for (int i = 1; i <=Number; i++) {

factorial\*=i;

}

System.out.println("Factorial of "+Number+" is : "+factorial);

sc.close();

}

}

Flowchart:-

Explanation:-In the program scanner takes input from the user. Factorial I sintialize with 1 to store the value calculated. For loop will start from 1 upto the value given and for each value of I will be multiplied with factorial and stored it in factorial.

Output:-

Time Complexity:-

Test Cases:

Input: 5

Output: 120

Input: 0

Output: 1

4. Fibonacci Series

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

Program:-

**package** Com.in;

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

**static** **int** fib(**int** n) {

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

**return** n;

}

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

}

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

**int** num = 5;

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

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

}

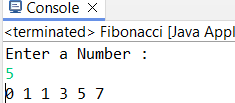
}

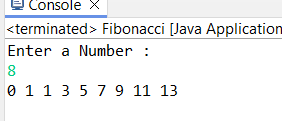
}

Flowchart:-

Explanation:-For Fibonacci series If contains the base condition n must not be less than 1 or it will return the value again or number is greater than 1 then it will recursively calucate fib(n) until it satisfies the base condition hence we will get the Fibonacci series upto the number provided.

Output:-





Time Complexity:- O(2^n)

Test Cases:

Input: n = 5

Output: [0, 1, 1, 2, 3]

Input: n = 8

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

5. Find GCD

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

Program:-

**package** Com.in;

**import** java.util.Scanner;

**public** **class** FindGCD {

**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 the first number: ");

**int** num1 = sc.nextInt();

System.***out***.print("Enter the second number: ")

**int** num2 = sc.nextInt();

**int** gcd = *GCD*(num1, num2);

System.***out***.println("The GCD of " + num1 + " and " + num2 + " is: " + gcd);

sc.close();

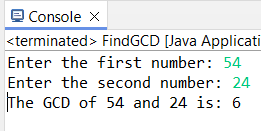
}

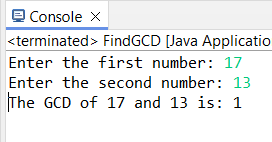
}

Flowchart:-

Explanation:-In the GCD program we are taking two numbers to find there GCD if b is equal to zero then a will be the gcd. If no then it will return GCD for a and b.

Output:-





Time Complexity:-O(log n)

Test Cases:

Input: a = 54, b = 24

Output: 6

Input: a = 17, b = 13

Output: 1

6. Find Square Root

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

Program:- **package** Com.in;

**import** java.util.Scanner;

**public** **class** Squareroot {

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

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

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

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

**double** squareroot = Math.*pow*(n, 0.5);

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

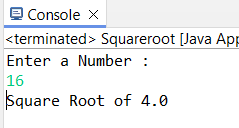
}

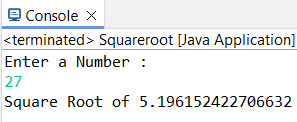
}

Flowchart:-

Explanation:-In the program we have to calculate squareroot of the given number. I have taken input from user using scanner class and stored it in n.For calculating squareroot used Math.pow function where n raise ot 0.5 will give root of the number.

Output:-





Time Complexity:- O(√X)

Test Cases:

Input: x = 16

Output: 4

Input: x = 27

Output: 5

7. Find Repeated Characters in a String

Problem: Write a Java program to find all repeated characters in a string.

Flowchart:-

Explanation:-

Output:-

Time Complexity:-

Test Cases:

Input: "programming"

Output: ['r', 'g', 'm']

Input: "hello"

Output: ['l']

8. First Non-Repeated Character

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

Program:-

**package** Com.in;

**import** java.util.LinkedHashMap;

**import** java.util.Map;

**import** java.util.Scanner;

**public** **class** RepeatedChar {

**public** **static** **char** findNonRepeatedChar(String str) {

Map<Character, Integer> charCountMap = **new** LinkedHashMap<>();

**for** (**char** ch : str.toCharArray()) {

charCountMap.put(ch, charCountMap.getOrDefault(ch, 0) + 1);

}

**for** (Map.Entry<Character, Integer> entry : charCountMap.entrySet()) {

**if** (entry.getValue() == 1) {

**return** entry.getKey();

}

}

**return** '\0';

}

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

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

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

String input = sc.nextLine();

**char** result = *findNonRepeatedChar*(input);

**if** (result != '\0') {

System.***out***.println("The first non-repeated character is: " + result);

} **else** {

System.***out***.println("No non-repeated character found.");

}

sc.close();

}

}

Flowchart:-

Explanation:-

Output:-

Time Complexity:-

Test Cases:

Input: "stress"

Output: 't'

Input: "aabbcc"

Output: null

9. Integer Palindrome

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

Program:- **package** Com.in;

**import** java.util.Scanner;

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

**public** **static** **boolean** isPalindrome(**int** num) {

**int** originalNum = num;

**int** reversedNum = 0;

**int** remainder;

**while** (num != 0) {

remainder = num % 10;

reversedNum = reversedNum \* 10 + remainder;

num = num / 10;

}

**return** originalNum == reversedNum;

}

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

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

System.***out***.print("Enter an integer: ");

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

**if** (*isPalindrome*(input)) {

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

} **else** {

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

}

sc.close();

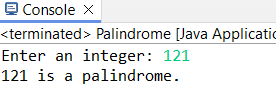
}

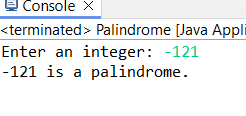
}

Flowchart:-

Explanation:-To check palindrome the number is taken as integer and reveersenumber will store the palindrome of number. In while base condition checks if the number is not equal to zero . then num is divided by 10 to seprate each digit at the end and stored in remainder. In next statement it will stored reversed number by adding remainder to it and num will be divide again to remove last digit . The loop will iterate until num becomes zero. After while loop it will check if the number is equal to reversenumber then it is palindrome.

Output:-





Time Complexity:-O(log n)

Test Cases:

Input: 121

Output: true

Input: -121

Output: false

10. Leap Year

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

Program:-

**package** Com.in;

**import** java.util.Scanner;

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

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

**if** (year % 4 == 0) {

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

**return** **true**;

}

}

**return** **false**;

}

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

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

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

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

**if** (*isLeapYear*(year)) {

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

} **else** {

System.***out***.println(year + " is not a leap year.");

}

sc.close();

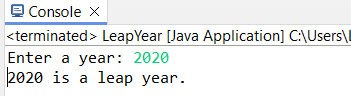
}

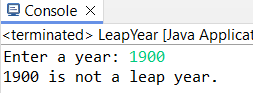
}

Flowchart:-

Explanation:-To check leap year first take the input of user of year, First if block contains that if year is divisible by 4 then it is leap year. Then if the year is not divisible by 100 but divisible by 4 then year is leap year Or year is divisible by both 100 and 400 then it is leap year.

Output:-





Time Complexity:-O(1)

Test Cases:

Input: 2020

Output: true

Input: 1900

Output: false