

**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

```
import java.util.Scanner;
class ArmstrongNumber {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println(" ");
        int num = sc.nextInt();
        int n = num;
        int sum = 0;

        while (num > 0) {
            int digit = num % 10;
            sum += digit * digit * digit;
            num /= 10;
        }

        if(sum == n) {
            System.out.println("True");
        } else {
            System.out.println("False");
        }

        sc.close();
    }
}
```

```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac ArmstrongNumber.java
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java ArmstrongNumber
153
True
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java ArmstrongNumber
129
False
```

Explanation:

The program checks if input number is equal to the sum of the cubes of its digits. It extracts each digit using modulus and division operations cubes it, and adds the result to a sum. Finally, it compares this sum with the original number to determine if it's an Armstrong number.

Time Complexity: $O(\log_{10}(n))$

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

```
import java.util.Scanner;
public class PrimeNumbers {
    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)) {
            System.out.println(" True");
        } else {
            System.out.println(" False");
        }

        sc.close();
    }
    public static boolean isPrime(int num) {

        if (num < 2) {           // If the number is less than 2, it's not a prime number
            return false;
        }
    }
}
```

```

    for (int i = 2; i <= Math.sqrt(num); i++) {
        if (num % i == 0) {
            return false; // If divisible, it's not prime
        }
    }

    return true; // If no divisors, it's prime
}
}

```

```

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac PrimeNumbers.java

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java PrimeNumbers
Enter the Number:
29
True

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java PrimeNumbers
Enter the Number:
15
False

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>|

```

Explanation:

The user enters a number.

Then, it will check the condition that is if number is less than 2 then it will return false. And if it's greater then it will move to the next for loop and here it will check that that the number is divisible up to the square root of the input which is given, if it divisible then it will return false.

Else it will print True

Time Complexity: $O(\text{square root } n)$

Space Complexity: $O(1)$

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

```

import java.util.Scanner;
class Factorial {
    public static void main(String args[]) {

```

```

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

int fact = 1;
for(int i = 1; i <= num; i++) {
    fact *= i;
}

System.out.println(+fact);
sc.close();

```

```

}
}

```

```

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac Factorial.java

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Factorial

5
120

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Factorial

7
5040

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>|

```

Explanation:

The user will give the input.

Then, for loop will execute and it will multiply from the number to 1.

Finally, the result will be print which will be factorial of the given number.

Time Complexity: $O(n)$

Space Complexity: $O(1)$

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]

```
import java.util.Scanner;
```

```

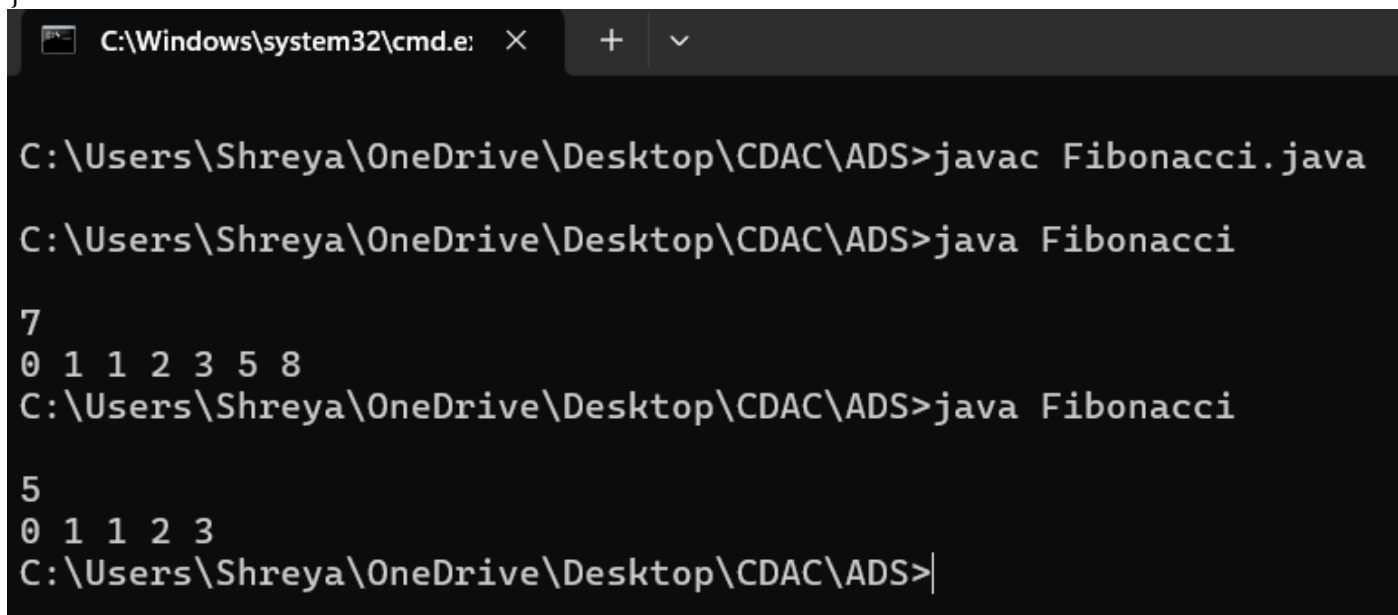
class Fibonacci {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.println(" ");
        int n = sc.nextInt();
        int a = 0, b = 1;

        for(int i=0; i < n; i++) {
            System.out.print(a+ " ");
            int next = a + b;

            a = b;
            b = next;
        }

        sc.close();
    }
}

```



The screenshot shows a Windows command prompt window with the title bar 'C:\Windows\system32\cmd.e:'. The user has navigated to the directory 'C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS'. They have compiled the 'Fibonacci.java' file using 'javac Fibonacci.java'. Then, they have run the program twice using 'java Fibonacci'. The first run, with input '7', produced the output '0 1 1 2 3 5 8'. The second run, with input '5', produced the output '0 1 1 2 3'.

```

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac Fibonacci.java
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Fibonacci
7
0 1 1 2 3 5 8
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Fibonacci
5
0 1 1 2 3
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>

```

Explanation:

The program prints the Fibonacci sequence up to the nth term, where each term is the sum of the previous two. It uses two variables to store the previous terms, updating them iteratively to generate the sequence.

Time Complexity: $O(n)$

Space Complexity: $O(1)$

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

```
import java.util.Scanner;
class GCD {
    public static int CaluculateGCD(int a, int b){
        while(b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
        }
        return a;
    }
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print("a= ");
        int a = sc.nextInt();
        System.out.print("b= ");
        int b = sc.nextInt();
        System.out.println(CaluculateGCD(a,b));
        sc.close();
    }
}
```

```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac GCD.java
```

```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java GCD
```

```
a= 5
b= 25
5
```

```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java GCD
```

```
a= 44
b= 24
4
```

```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java GCD
```

```
a= 54
b= 24
6
```

Explanation:

The program calculates the greatest common divisor (GCD) of two numbers.

It recursively finds the GCD by replacing the larger number with its remainder when divided by the smaller number until the remainder is zero.

Time Complexity: $O(\log(\min(a,b)))$

Space Complexity: $O(1)$

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

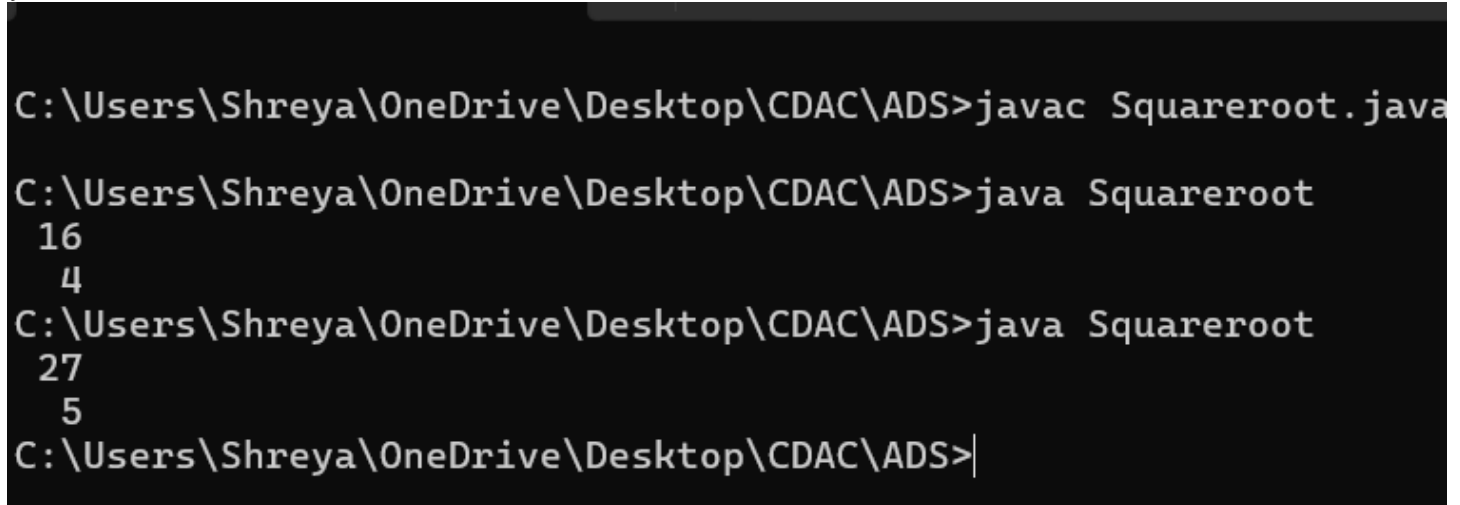
Input: x = 27

Output: 5

```
import java.util.Scanner;
class Squarerooot {

    public static int findroot(int n) {
        int i = 1;
        while( i * i <= n) {
            i++;
        }
        return i - 1;
    }
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print(" ");
        int n = sc.nextInt();

        System.out.print(" " + findroot(n));
        sc.close();
    }
}
```



```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac Squarerooot.java
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Squarerooot
16
4
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Squarerooot
27
5
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>
```

Explanation:

The program computes the integer square root of a given number by incrementing a value until its square exceeds the input number.

The largest value whose square is less than or equal to the input is returned as the integer square root.

Time Complexity: $O(\text{square root } n)$

Space Complexity: $O(1)$

7. Find Repeated Characters in a String

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

Test Cases:

Input: "programming"

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

Input: "hello"

Output: ['l']

```
import java.util.Scanner;
class Repeatedcharacter {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        System.out.print(" ");
        String ip = sc.nextLine();

        for (int i = 0; i < ip.length(); i++) {
            char c = ip.charAt(i);
            // Check if the character is repeated and if it's the first occurrence
            if (ip.indexOf(c) != ip.lastIndexOf(c) && ip.indexOf(c) == i) {
                System.out.print(c + " ");
            }
        }
        sc.close();
    }
}
```

```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac Repeatedcharacter.java
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Repeatedcharacter
programming
r g m
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Repeatedcharacter
hello
l
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>
```

Explanation:

We will take the input from the user.

Then, the for loop will start and it will iterate over each character in the input string.

If the conditions stated, then it will print the repeated character in the string.

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$

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

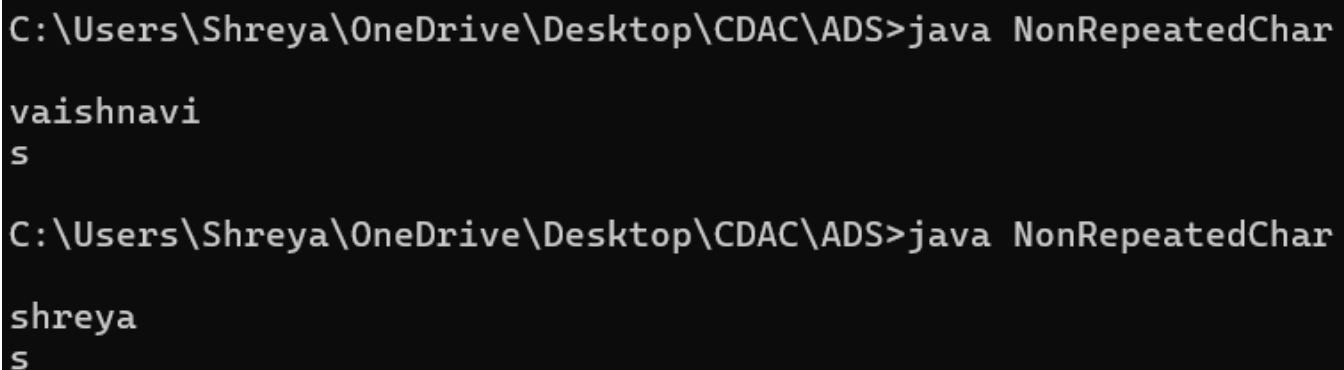
```
import java.util.Scanner;
public class NonRepeatedChar {
    public static Character firstNonRepeatedChar(String ip) {
        for (char c: ip.toCharArray()) {
            if(ip.indexOf(c) == ip.lastIndexOf(c)) {
                return c;
            }
        }
        return null;
    }

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);
        System.out.println(" ");
        String ip = sc.nextLine();

        Character res = firstNonRepeatedChar(ip);

        System.out.println(res != null ? res:"null");
        sc.close();
    }
}
```



```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java NonRepeatedChar
vaishnavi
s
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java NonRepeatedChar
shreya
s
```

Explanation:

It will take input of the string.

Then, in the for loop it will convert String into character array. After that it will check first occurrence index character and the last occurrence index of character if it's equal that means it appears only once in string. If non repeated character is found, it will return and if not by the end of loop null will return.

Time Complexity: $O(n^2)$

Space Complexity: $O(n)$

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

```
import java.util.Scanner;

public class Palindrome{
    public static void main (String a[]){
        Scanner scanner = new Scanner (System.in);
        System.out.println(" ");
        int num = scanner.nextInt();

        if(isPalindrome(num)){
            System.out.println("True.");
        }else{
            System.out.println("False");
        }
        scanner.close();
    }

    public static boolean isPalindrome(int num){
        int reversed = 0 , original = num;
        while (num!=0) {
            reversed = reversed * 10 + num % 10 ;
            num /= 10;
        }
        return original== reversed;
    }
}
```

```
C:\Windows\system32\cmd.e: X + v

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac Palindrome.java

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Palindrome

121
True.

C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Palindrome

15
False
```

Explanation:

First, takes the input from the user.

It will calculate the number with mod 10 and will store the result in reversed

And it will divide num with 10

Then, method will enter in loop that will continue until num becomes 0

Then, check whether original is equal to the reversed or not if it's equal then it will print True else it will print False

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.

Test Cases:

Input: 2020

Output: true

Input: 1900

Output: false

```
import java.util.*;
class Leapyear {
    public static void main(String args[]) {
        System.out.println(" ");
        Scanner sc = new Scanner(System.in);
        int year = sc.nextInt();
        if((year % 4 == 0 && year % 100 != 0) || year % 400 == 0 ) {
            System.out.println("True");
        }
    }
}
```

```
        else {  
            System.out.println("False");  
        }  
    }  
}
```

```
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>javac Leapyear.java  
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Leapyear  
2024  
True  
  
C:\Users\Shreya\OneDrive\Desktop\CDAC\ADS>java Leapyear  
2023  
False
```

Explanation:

First takes the year as input using Scanner class.

Then, it will check the condition that if the year is divisible by 4 & divisible by 400 and not by 100 then it will print True.

Else it will print False.

Time Complexity: $O(1)$

Space Complexity: $O(1)$