C-DAC Mumbai

Date 26/09/2024

Subject: Algorithm and Data Structure
Assignment 2

Solve the assignment with the following thing to be added in each question.

-Program
-Flow chart
-Explanation
-Output
-Time and Space complexity

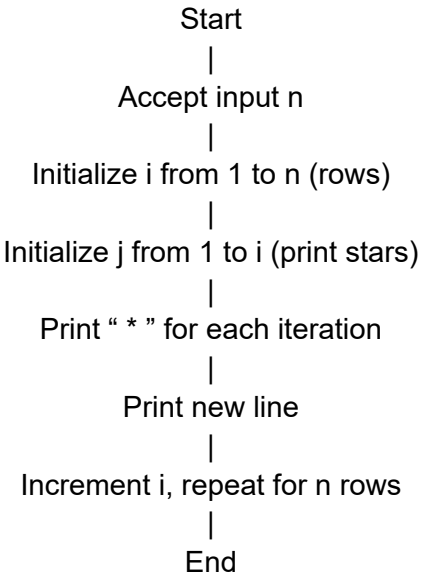
1. Printing Patterns
Problem: Write a Java program to print patterns such as a right triangle of stars.

Test Cases:
Input: n = 3
Output: *
**
***
Input: n = 5
Output: *
**
***
****
*****

-Program

```
import java.util.Scanner;
public class StarPattern {
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		System.out.println("Enter the number of rows: ");
		int n = scanner.nextInt();
		for(int i = 1; i <= n; i++){
			for(int j = 1; j <= i; j++){
				System.out.print("*");
			}
			System.out.println();
		}
		scanner.close();
	}
}
```

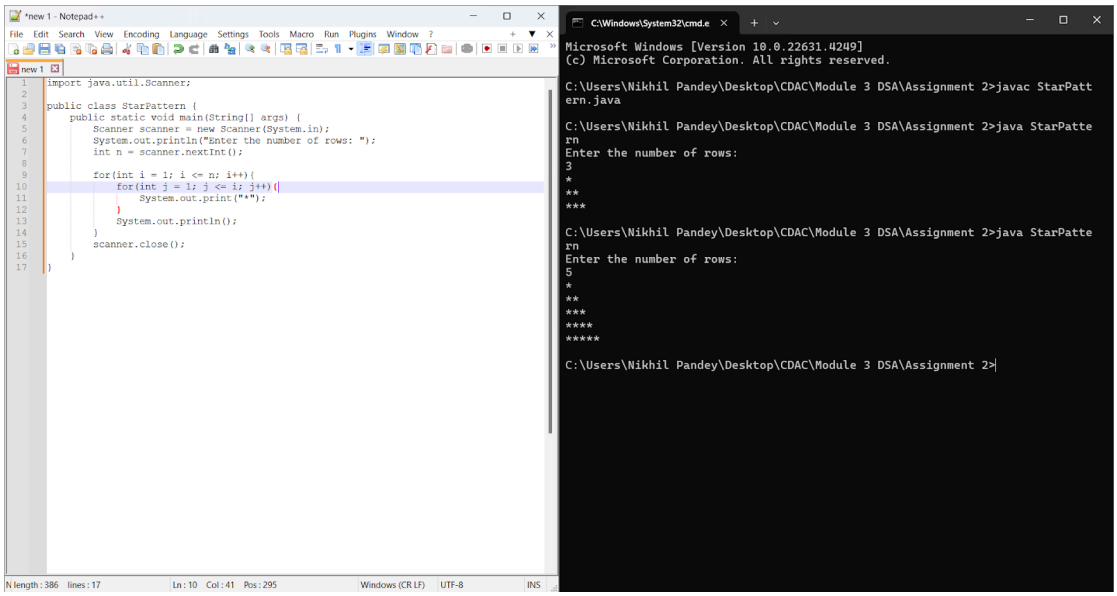
-Flow chart



-Explanation

Input: The program takes an integer n as input, representing the number of rows for the triangle pattern.
Logic: A nested loop is used. The outer loop runs from 1 to n (for each row), and the inner loop runs from 1 to the current row number to print stars (*).
Output: After printing the stars for a row, a newline is printed to move to the next line.

-Output



```
import java.util.Scanner;

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

        for(int i = 1; i <= n; i++){
            for(int j = 1; j <= i; j++){
                System.out.print("*");
            }
            System.out.println();
        }
        scanner.close();
    }
}
```

```
Microsoft Windows [Version 10.0.22631.4249]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 2>javac StarPatt
ern.java

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 2>java StarPatte
rn
Enter the number of rows:
3
*
**
***

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 2>java StarPatte
rn
Enter the number of rows:
5
*
**
***
****
*****

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 2>
```

-Time and Space complexity

Time and Space Complexity:
Time Complexity: **$O(n^2)$**
The outer loop runs n times, and the inner loop runs up to i times for each iteration of the outer loop, leading to a total of $1 + 2 + 3 + ... + n = O(n^2)$ operations.

Space Complexity: **O(1)**
The space used is constant, as only a few variables (i, j, and n) are required, and no additional space is used that grows with the input size.

2. Remove Array Duplicates
Problem: Write a Java program to remove duplicates from a sorted array and return the new length of the array.

Test Cases:
Input: arr = [1, 1, 2]
Output: 2
Input: arr = [0, 0, 1, 1, 2, 2, 3, 3]
Output: 4

-Program

```
import java.util.Scanner;

public class RemoveDuplicates {
   public static int removeDuplicates(int[] arr){
      if (arr.length == 0) return 0 ;

         int uniqueLength = 1;
         for(int i = 1; i<arr.length; i++){
            if (arr[i] != arr[i-1]) {
               arr[uniqueLength] = arr[i];
               uniqueLength++;
            }
         }
         return uniqueLength;
      }
      public static void main(String[] args) {
         Scanner scanner = new Scanner(System.in);
         System.out.println("Enter the number of elements in the array: ");
         int n = scanner.nextInt();
         int[] arr = new int[n];
         System.out.println("Enter the elements (Sorted Array): ");
         for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
         }
         int newLength = removeDuplicates(arr);
         System.out.println("The new length of the array after removing duplicates is: " +
newLength);
         scanner.close();

      }
   }
```

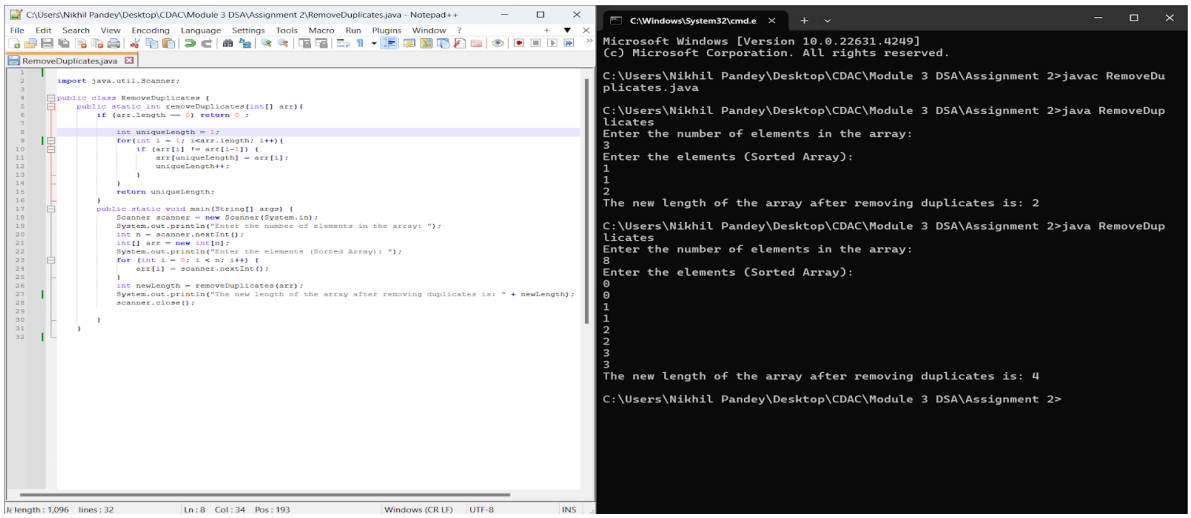
-Flow chart

Start
|
Accept input array size n
|
Accept n sorted array elements
|
Initialize uniqueLength = 1 (First element is unique)
|
Loop from i = 1 to n - 1
|
If arr[i] is not equal to arr[i - 1], copy to arr[uniqueLength] and increment uniqueLength
|
Return uniqueLength
|
End

-Explanation

Input: The program accepts an integer n for the size of the array and then accepts n sorted integers into the array.
Logic: The program starts with the first element and assumes it's unique.
It then iterates over the rest of the array, checking if the current element is different from the previous one.
If it's different, it copies the current element to the next position in the modified array.
The length of the unique elements is tracked by uniqueLength.
Output: The program returns the new length of the array after removing duplicates.

-Output



-Time and Space complexity

Time Complexity: **O(n)**
The program traverses the array once, making comparisons and moving elements, so it takes linear time relative to the size of the array.

Space Complexity: **O(1)**
The program modifies the array in place without using any additional space proportional to the input size.

3. Remove White Spaces from String
Problem: Write a Java program to remove all white spaces from a given string.

Test Cases:
Input: "Hello World"
Output: "HelloWorld"
Input: " Java Programming "
Output: "JavaProgramming"

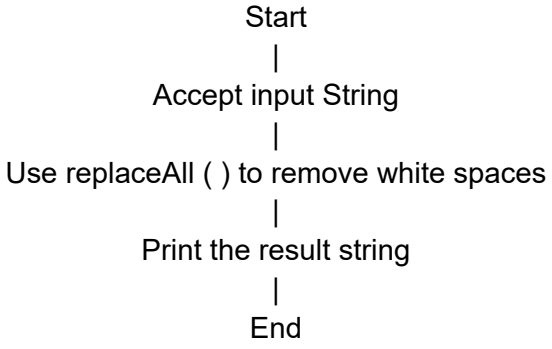
-Program

```
import java.util.Scanner;

public class RemoveWhiteSpaces {
	public static String removeWhiteSpaces(String str){
		return str.replaceAll("\\s", "");
	}
	public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		System.out.println ("Enter a string: ");
		String input = scanner.nextLine();

		String result = removeWhiteSpaces(input);
		System.out.println("Sorting after removing space: " + result);
	}
}
```

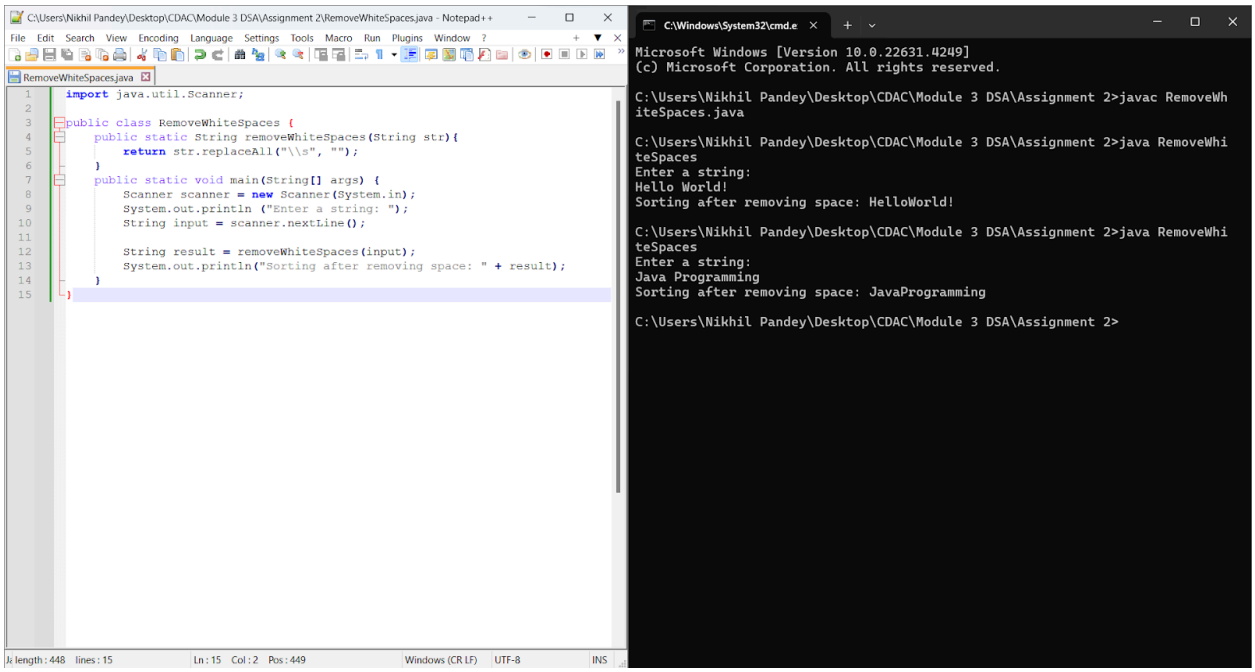
-Flow chart



-Explanation

Input: The program accepts a string from the user.
Logic: The program uses the replaceAll() method with the regex pattern \\s to remove all whitespace characters (including spaces, tabs, and newlines) from the input string.
Output: The program prints the string after removing all white spaces.

-Output



-Time and Space complexity

Time Complexity: **O(n)**
The program processes each character of the input string once, so the time complexity is linear in the size of the string.

Space Complexity: **O(n)**
A new string is created without the white spaces, which requires space proportional to the size of the input string.

4. Reverse a String
Problem: Write a Java program to reverse a given string.

Test Cases:
Input: "hello"
Output: "olleh"
Input: "Java"
Output: "avaJ"

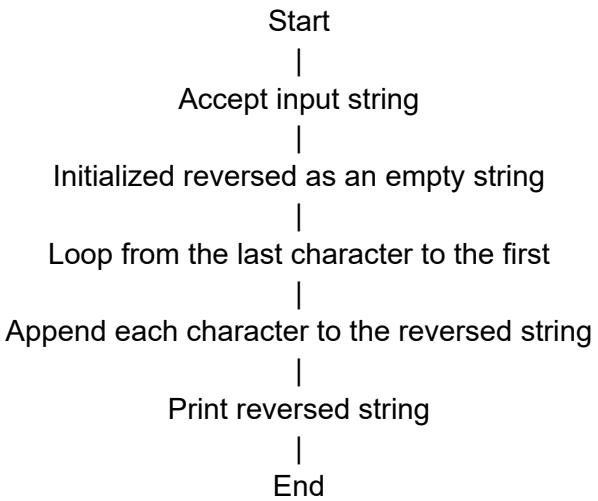
-Program

```
import java.util.Scanner;

public class ReverseString {
    public static String reverseString(String str) {
        String reversed = "";
        for (int i = str.length() - 1; i >= 0; i--) {
            reversed += str.charAt(i);
        }
        return reversed;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a string: ");
        String input = scanner.nextLine();

        String result = reverseString(input);System.out.println("Reversed string: " + result);
    }
}
```

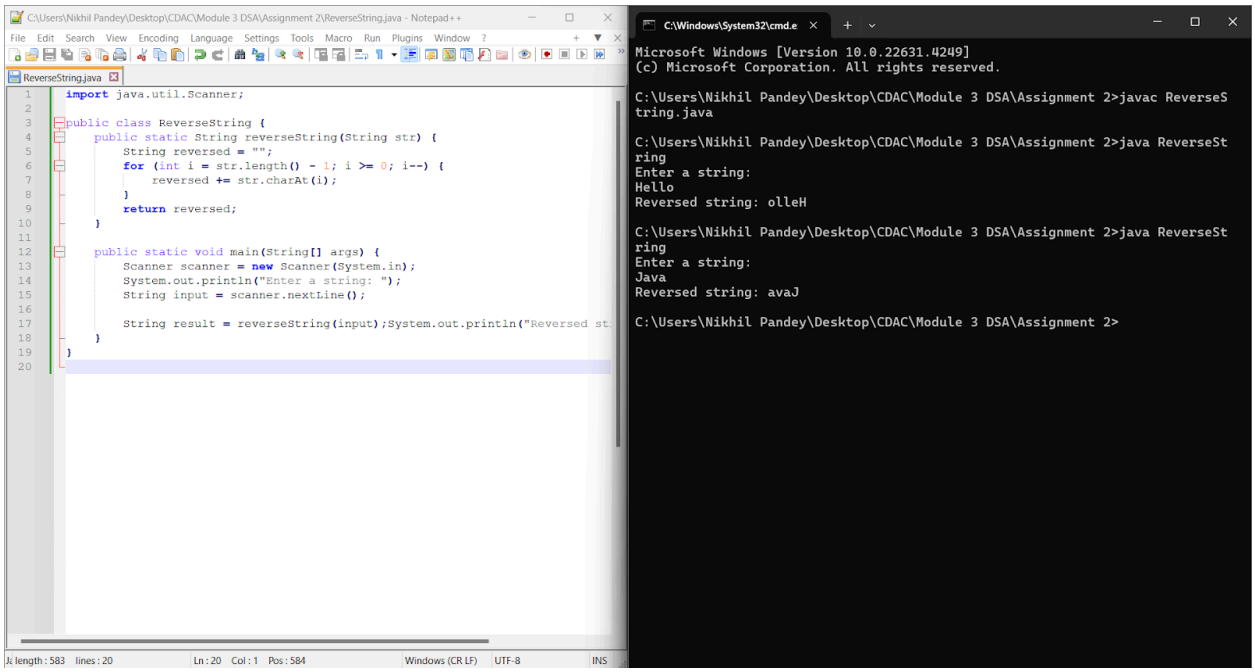
-Flow chart



-Explanation

Input: The program accepts a string from the user.
Logic: The program iterates through the input string starting from the last character and appends each character to a new string reversed.
Once the loop completes, the reversed string will contain the input string in reverse order.
Output: The program prints the reversed string.

-Output



-Time and Space complexity

Time Complexity: **O(n)**
The program processes each character of the input string once in a loop, so the time complexity is linear in the size of the string.

Space Complexity: **O(n)**
A new string is created to store the reversed string, so the space complexity is proportional to the input string's size.

5. Reverse Array in Place
Problem: Write a Java program to reverse an array in place.

Test Cases: Input: arr = [1, 2, 3, 4]
Output: [4, 3, 2, 1]
Input: arr = [7, 8, 9]
Output: [9, 8, 7]

-Program

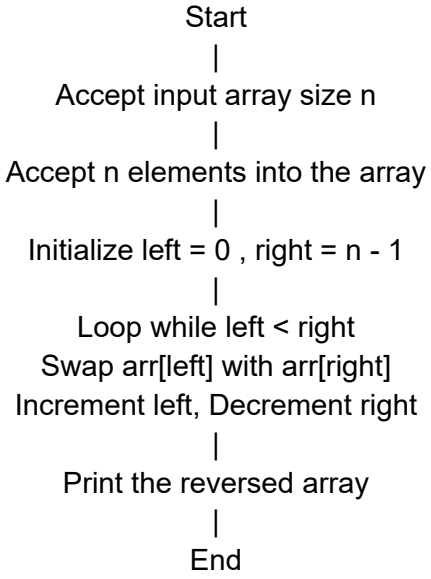
```
import java.util.Arrays;
import java.util.Scanner;

public class ReverseArray {
    public static void reverseArrayInPlace(int[] arr) {
        int left = 0;
        int right = arr.length - 1;

        while (left < right) {
            int temp = arr[left];
            arr[left] = arr[right];
            arr[right] = temp;
            left++;
            right--;
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] arr = new int[n];

        System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        reverseArrayInPlace(arr);
        System.out.println("Reversed array: " + Arrays.toString(arr));
    }
}
```

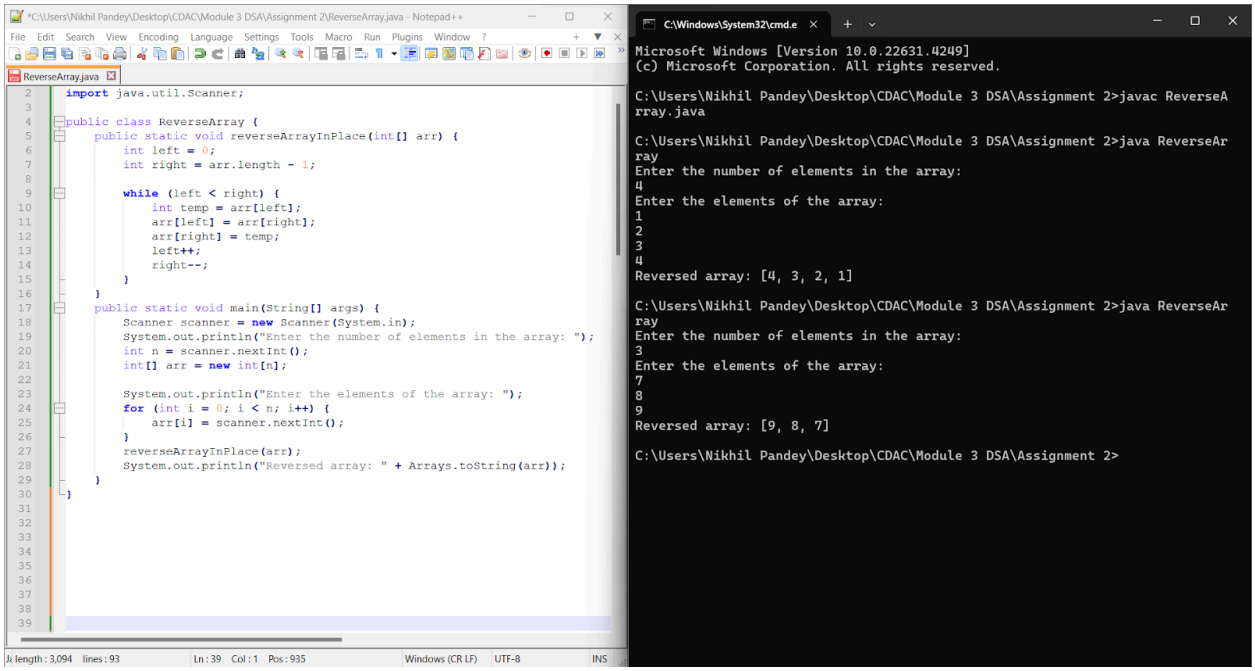
-Flow chart



-Explanation

Input: The program accepts an integer n for the size of the array, followed by n integers as the array elements.
Logic: The program uses two pointers, left starting from the beginning of the array and right starting from the end.
In each iteration, the values at the left and right pointers are swapped, and both pointers are moved towards the center.
This process continues until left is no longer less than right, ensuring that all elements have been reversed in place.
Output: The program prints the array after reversing it in place.

-Output



-Time and Space complexity

Time Complexity: **O(n)**
The program swaps elements from both ends of the array, and it does this for half of the elements, resulting in a linear time complexity.

Space Complexity: **O(1)**
The program reverses the array in place, meaning it uses constant extra space.

6. Reverse Words in a String
Problem: Write a Java program to reverse the words in a given sentence.

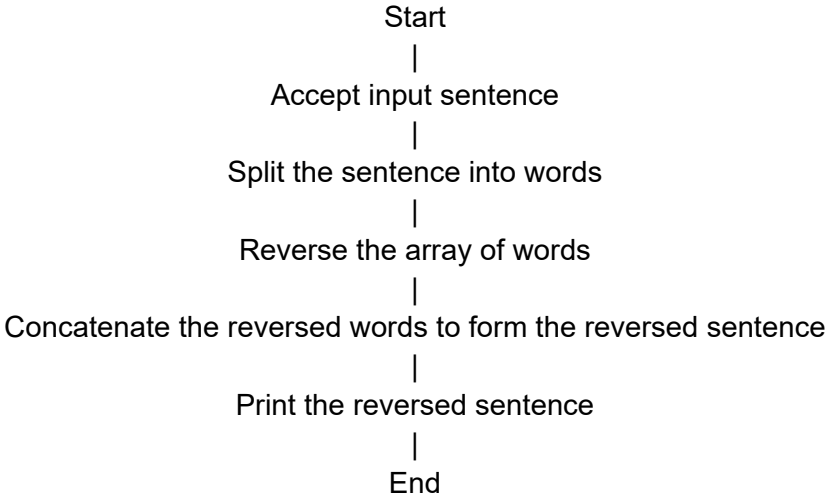
Test Cases:
Input: "Hello World"
Output: "World Hello"
Input: "Java Programming"
Output: "Programming Java"

-Program

```
import java.util.Scanner;

public class ReverseWordsIntoString {
    public static String reverseWords(String sentence) {
        String[] words = sentence.trim().split("\\s+");
        StringBuilder reversedSentence  = new StringBuilder();
        for (int i = words.length - 1; i >= 0; i--) {
            reversedSentence.append(words[i]).append(" ");
        }
        return reversedSentence.toString().trim();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a sentence: ");
        String sentence = scanner.nextLine();
        String result = reverseWords(sentence);
        System.out.println("Reversed sentence: " + result);
    }
}
```

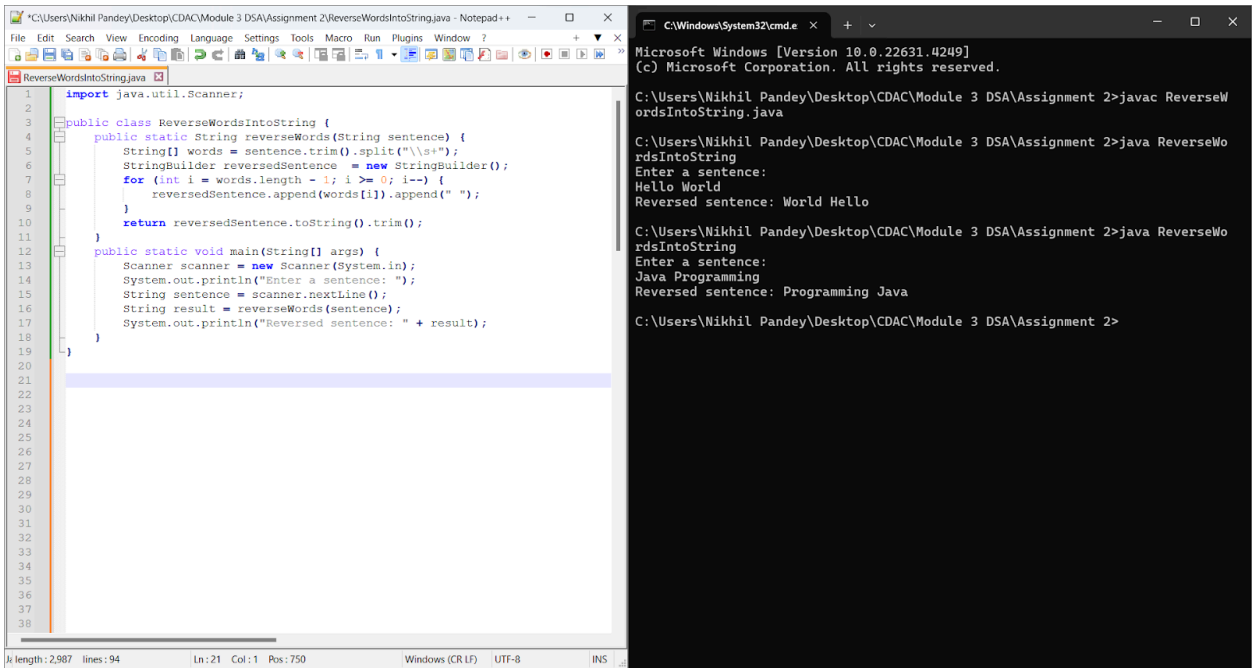
-Flow chart



-Explanation

Input: The program accepts a sentence from the user.
Logic: The input sentence is first split into individual words using the split("\\s+") method, which handles multiple spaces between words.
The words are then reversed by iterating through the array from the end to the beginning.
A StringBuilder is used to append the reversed words and form the final sentence.
The trim() method is used to remove any trailing spaces from the result.
Output: The program prints the sentence with the words reversed, while preserving the order of characters in each word.

-Output



-Time and Space complexity

Time Complexity: **O(n)**
The program splits the sentence into words, which takes linear time with respect to the size of the sentence. It then reverses the words, which is also done in linear time.

Space Complexity: **O(n)**
The program requires additional space to store the words and the reversed sentence, proportional to the size of the input.

7. Reverse a Number
Problem: Write a Java program to reverse a given number.
Test Cases:
Input: 12345
Output: 54321
Input: -9876
Output: -6789

-Program

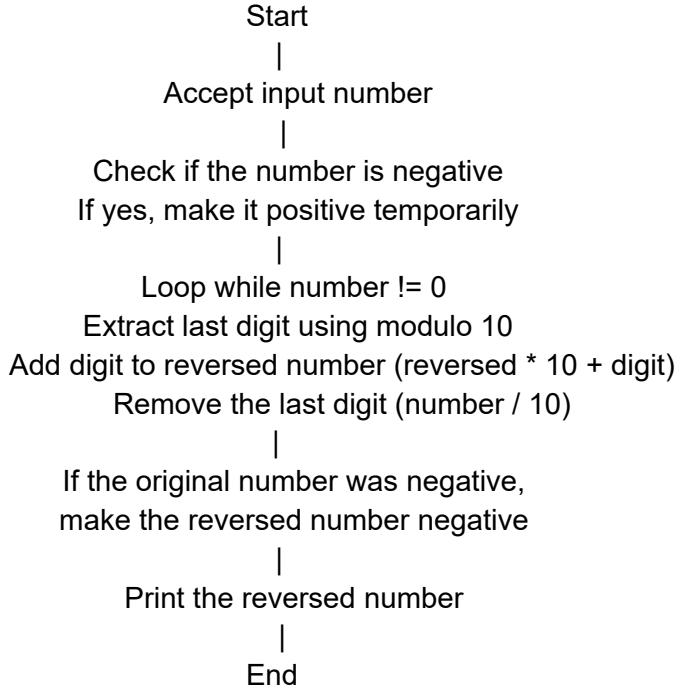
```
import java.util.Scanner;

public class ReverseNumber {
    public static int ReverseNumber(int number) {
        int reversed = 0;
        boolean isNegative = number < 0;

        if (isNegative) {
            number = -number;
        }
        while (number != 0) {
            int digit = number % 10;
            reversed = reversed * 10 + digit;
            number /= 10;
        }
        return isNegative ? -reversed : reversed;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a number: ");
        int number = scanner.nextInt();

        int result = ReverseNumber(number);
        System.out.println("Reversed number: " + result);
    }
}
```

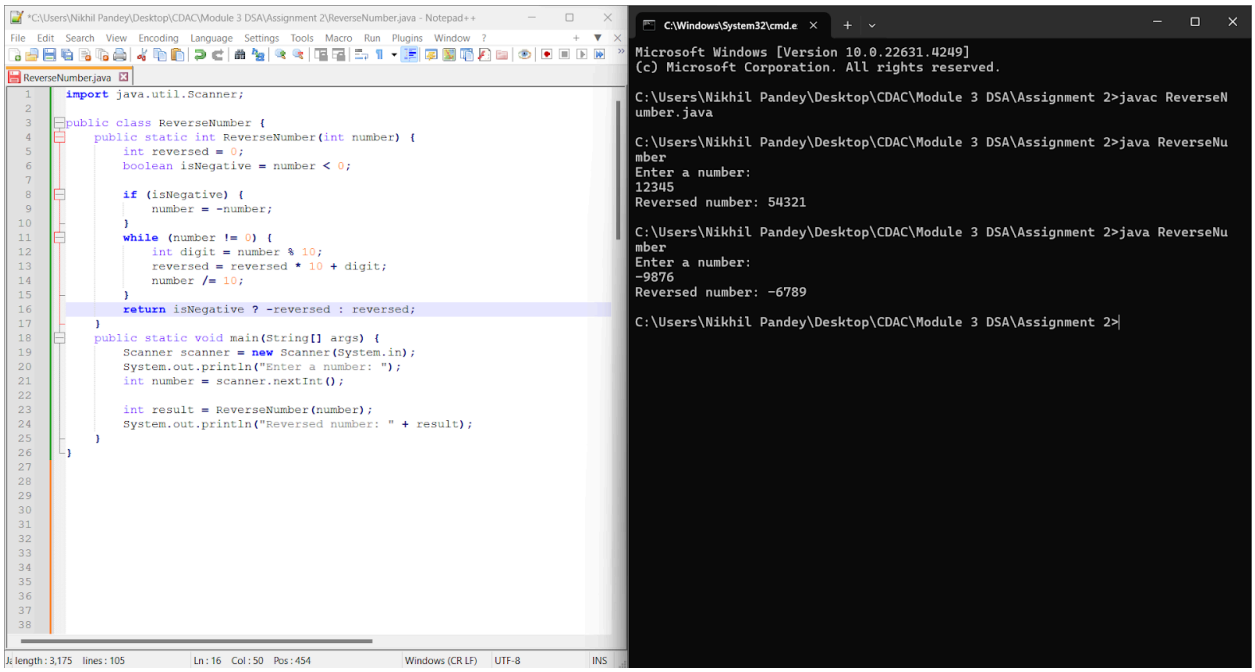
-Flow chart



-Explanation

Input: The program accepts an integer from the user.
Logic: The program first checks if the input number is negative. If so, it converts the number to positive for the reversal process.
It then extracts each digit by taking the modulo of 10 (number % 10) and appends it to the reversed number.
The number is divided by 10 to remove the last digit after appending.
After reversing, the program restores the negative sign if the original number was negative.
Output: The program prints the reversed number while maintaining the correct sign.

-Output



-Time and Space complexity

Time Complexity: **$O(\log_{10}(n))$**
The time complexity is proportional to the number of digits in the number, which is logarithmic with respect to the number.

Space Complexity: **O(1)**
The program only uses a fixed amount of additional space, making the space complexity constant.

8. Array Manipulation
Problem: Perform a series of operations to manipulate an array based on range update queries. Each query adds a value to a range of indices.

Test Cases:
Input: n = 5, queries = [[1, 2, 100], [2, 5, 100], [3, 4, 100]]
Output: 200
Input: n = 4, queries = [[1, 3, 50], [2, 4, 70]]
Output: 120

-Program

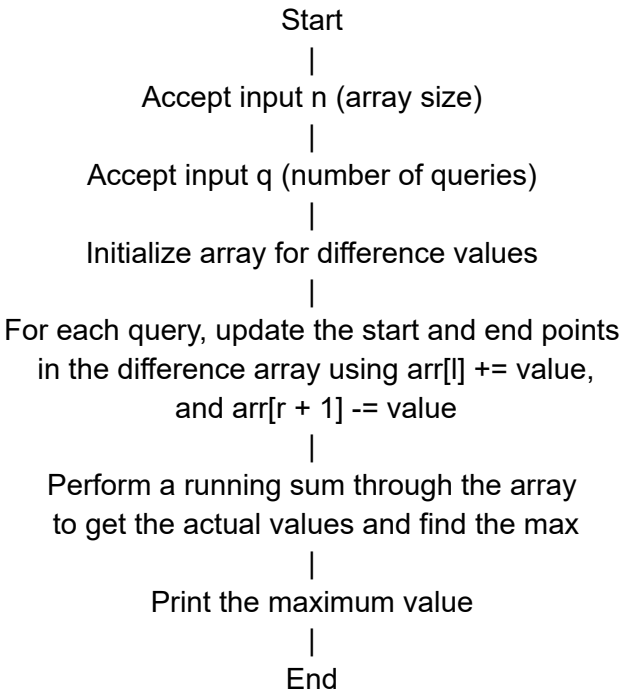
```
import java.util.Scanner;

public class ArrayManipulation {

    public static long arrayManipulation(int n, int[][] queries) {
        long[] arr = new long[n + 1];
        for (int[] query : queries) {
            int l = query[0] - 1;
            int r = query[1];
            int value = query[2];
            arr[l] += value;
            if (r < n) {
                arr[r] -= value;
            }
        }
        long max = Long.MIN_VALUE;
        long current = 0;
        for (int i = 0; i < n; i++) {
            current += arr[i];
            if (current > max) {
                max = current;
            }
        }
        return max;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = scanner.nextInt();
        System.out.println("Enter the number of queries:");
        int q = scanner.nextInt();
        int[][] queries = new int[q][3];
```

```
        System.out.println("Enter the queries in the format [l, r, value]:");
        for (int i = 0; i < q; i++) {
            queries[i][0] = scanner.nextInt();
            queries[i][1] = scanner.nextInt();
            queries[i][2] = scanner.nextInt();
        }
        long result = arrayManipulation(n, queries);
        System.out.println("Maximum value after operations: " + result);
    }
}
```

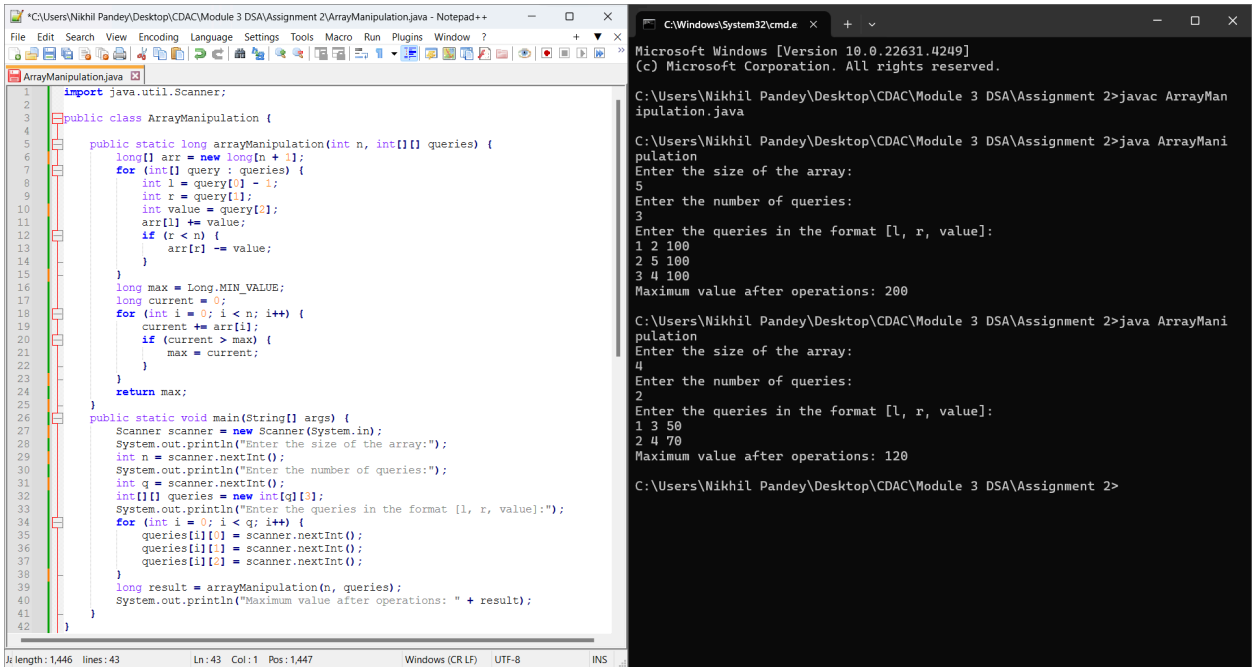
-Flow chart



-Explanation

Input: The program first accepts the size of the array n.
It then accepts the number of queries and their details, where each query consists of three integers [l, r, value].
Logic: An array arr of size n+1 is created to store the difference values.
For each query, the program updates the starting point of the range by adding the value at index l. It also stops adding the value after index r by subtracting the value at index r + 1.
After processing all queries, the program performs a running sum through the array to calculate the actual values and find the maximum value.
Output: The program prints the maximum value after performing all range update operations.

-Output



-Time and Space complexity

Time Complexity: **O(n + q)**
We process each query in constant time, i.e., O(1) using the difference array technique, so the total time complexity for q queries is O(q). Then, we compute the actual array values by doing a single pass through the array, which takes O(n) time.

Space Complexity: **O(n)**
We use an additional array of size n+1 for storing the difference values, so the space complexity is O(n).

9. String Palindrome
Problem: Write a Java program to check if a given string is a palindrome.

Test Cases:
Input: "madam"
Output: true
Input: "hello"
Output: false

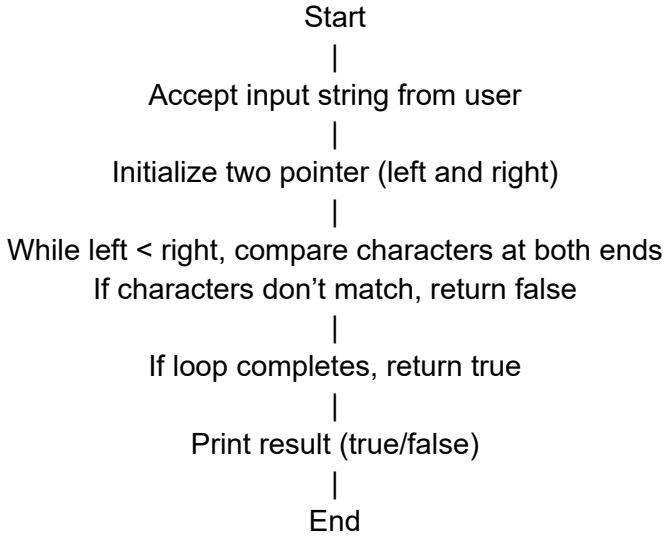
-Program

```
import java.util.Scanner;

public class PalindromeCheck {
    public static boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;

        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter a string: ");
        String input = scanner.nextLine();
        boolean result = isPalindrome(input);
        System.out.println("Is the string a palindrome? " + result);
    }
}
```

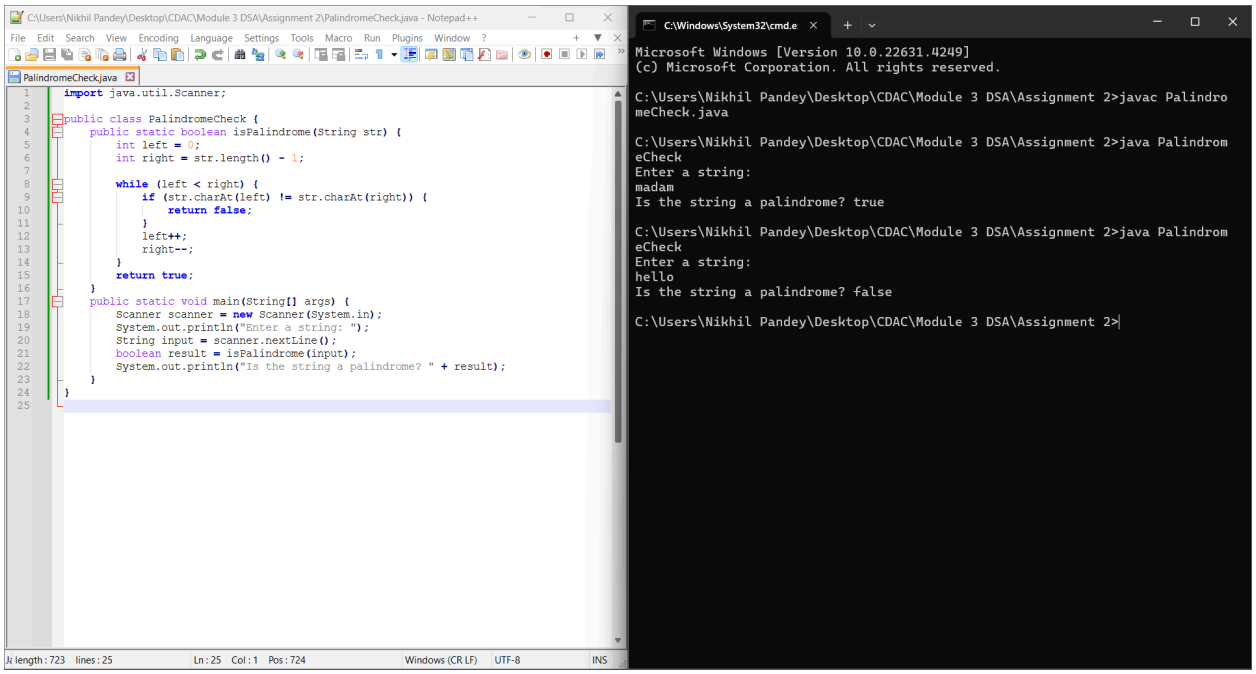
-Flow chart



-Explanation

Input: The program takes a string as input from the user.
Logic: Two pointers are initialized: left starts at the beginning, and right starts at the end of the string.
Characters at these two pointers are compared. If they differ, the string is not a palindrome, and the function returns false.
If the loop completes without finding a mismatch, the string is a palindrome, and the function returns true.
Output: The program prints true if the string is a palindrome and false otherwise.

-Output



-Time and Space complexity

Time Complexity: **O(n)**, where n is the length of the string. The program compares each character at most once.

Space Complexity: **O(1)**, since no extra space is used other than a few variables.

10. Array Left Rotation
Problem: Write a Java program to rotate an array to the left by d positions.

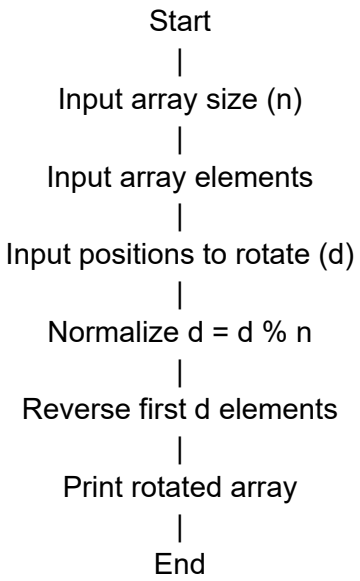
Test Cases:
Input: arr = [1, 2, 3, 4, 5], d = 2
Output: [3, 4, 5, 1, 2]
Input: arr = [10, 20, 30, 40], d = 1
Output: [20, 30, 40, 10]

-Program

```
import java.util.Scanner;
public class ArrayLeftRotation {
    public static void leftRotate(int[] arr, int d) {
        int n = arr.length;
        d = d % n;
        reverse(arr, 0, d - 1);
        reverse(arr, d, n - 1);
        reverse(arr, 0, n - 1);
    }
    public static void reverse(int[] arr, int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }
    public static void printArray(int[] arr) {
        for (int value : arr) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
```

```
        System.out.println("Enter the number of positions to rotate:");
        int d = scanner.nextInt();
        leftRotate(arr, d);
        System.out.println("Rotated array:");
        printArray(arr);
    }
}
```

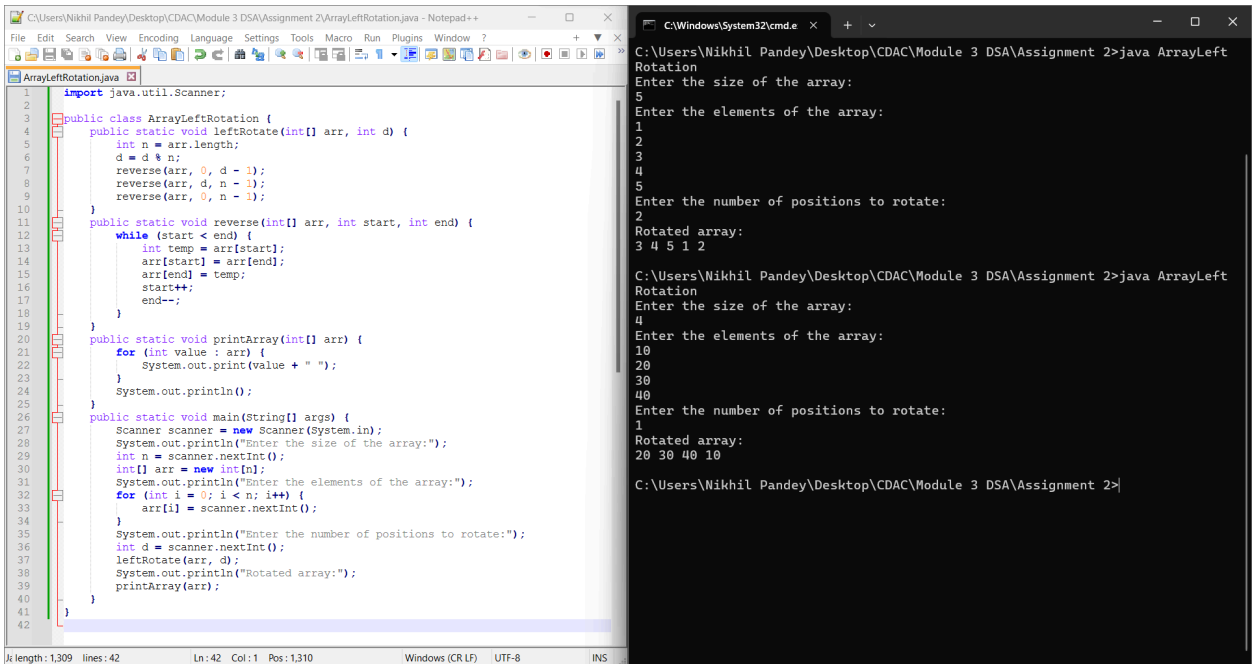
-Flow chart



-Explanation

Input: The program accepts an integer array and the number of positions d to rotate the array.
Logic: The array is rotated by reversing the first d elements, the remaining elements, and finally the entire array. This is an efficient way to rotate the array in-place.
Output: The array after performing the left rotation by d positions is printed.

-Output



```
import java.util.Scanner;

public class ArrayLeftRotation {
    public static void leftRotate(int[] arr, int d) {
        int n = arr.length;
        d = d % n;
        reverse(arr, 0, d - 1);
        reverse(arr, d, n - 1);
        reverse(arr, 0, n - 1);
    }
    public static void reverse(int[] arr, int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start] = arr[end];
            arr[end] = temp;
            start++;
            end--;
        }
    }
    public static void printArray(int[] arr) {
        for (int value : arr) {
            System.out.print(value + " ");
        }
        System.out.println();
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter the size of the array:");
        int n = scanner.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            arr[i] = scanner.nextInt();
        }
        System.out.println("Enter the number of positions to rotate:");
        int d = scanner.nextInt();
        leftRotate(arr, d);
        System.out.println("Rotated array:");
        printArray(arr);
    }
}
```

```
C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 2>java ArrayLeft
Rotation
Enter the size of the array:
5
Enter the elements of the array:
1
2
3
4
5
Enter the number of positions to rotate:
2
Rotated array:
3 4 5 1 2

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 2>java ArrayLeft
Rotation
Enter the size of the array:
4
Enter the elements of the array:
10
20
30
40
Enter the number of positions to rotate:
1
Rotated array:
20 30 40 10

C:\Users\Nikhil Pandey\Desktop\CDAC\Module 3 DSA\Assignment 2>
```

-Time and Space complexity

Time Complexity: **O(n)**, where n is the size of the array. We reverse the array three times, but each reversal takes linear time.

Space Complexity: **O(1)**, as the rotation is done in-place without using any extra space