**Assignment 1**

**Solve the assignment with 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 code:**

**package Star.pattern;**

**import java.util.Scanner;**

**public class StarPattern {**

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

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

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

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

**for (int i = 1; i <= n; i++) {**

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

**System.*out*.print("\*");**

**}**

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

**}**

**sc.close();**

**}**

**}**

**Explanation:**

1. **The program starts by taking the number of rows (n) from the user.**
2. **The outer loop runs from 1 to n, where each iteration represents a row.**
3. **The inner loop prints stars (\*) equal to the current row number (i).**
4. **After printing stars for a row, the program prints a newline.**
5. **The process continues until the pattern is printed for all rows.**

**Time Complexity: O(n^2)**

**Space Complexity:O(1)**

**Output:**

**Enter the number of rows: 3**

**\***

**\*\***

**\*\*\***

**Enter the number of rows: 5**

**\***

**\*\***

**\*\*\***

**\*\*\*\***

**\*\*\*\*\***

**2. Remove Array Duplicates**

**Problem: Write a Java program to remove duplicates from a sorted array and return the new length of the array.**

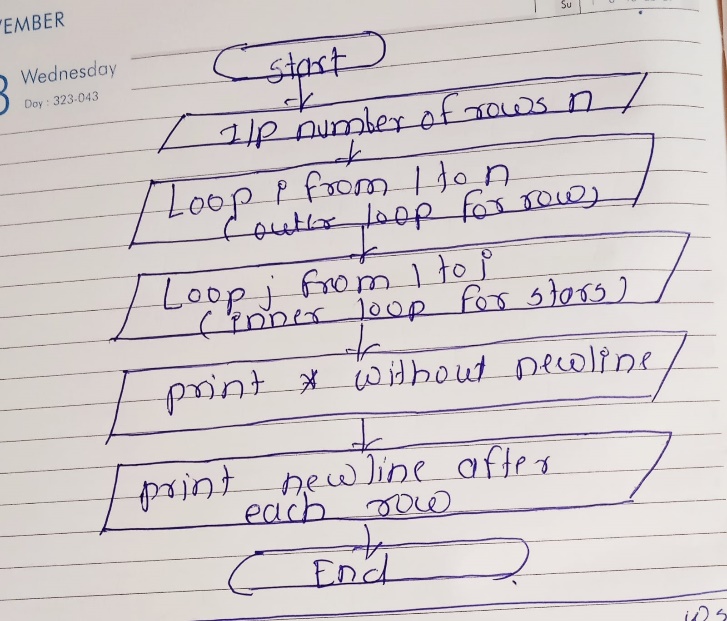
**Test Cases:**

**Input: arr = [1, 1, 2]**

**Output: 2**

**Input: arr = [0, 0, 1, 1, 2, 2, 3, 3]**

**Output: 4**

**flowchart**

**Program code:**

**package Star.pattern;**

**import java.util.Arrays;**

**public class RemoveDuplicates {**

**public static int removeDuplicates(int[] arr) {**

**if (arr.length == 0) return 0;**

**int uniqueIndex = 0;**

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

**if (arr[i] != arr[uniqueIndex]) {**

**uniqueIndex++;**

**arr[uniqueIndex] = arr[i];**

**}**

**}**

**return uniqueIndex + 1;**

**}**

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

**int[] arr1 = {1, 1, 2};**

**int[] arr2 = {0, 0, 1, 1, 2, 2, 3, 3};**

**System.*out*.println("New length after removing duplicates (Test Case 1): " + *removeDuplicates*(arr1));**

**System.*out*.println("New length after removing duplicates (Test Case 2): " + *removeDuplicates*(arr2));**

**System.*out*.println("Array after removing duplicates (Test Case 1): " + Arrays.*toString*(Arrays.*copyOf*(arr1, *removeDuplicates*(arr1))));**

**System.*out*.println("Array after removing duplicates (Test Case 2): " + Arrays.*toString*(Arrays.*copyOf*(arr2, *removeDuplicates*(arr2))));**

**}**

**}**

**Output:**

**New length after removing duplicates (Test Case 1): 2**

**New length after removing duplicates (Test Case 2): 4**

**Array after removing duplicates (Test Case 1): [1, 2]**

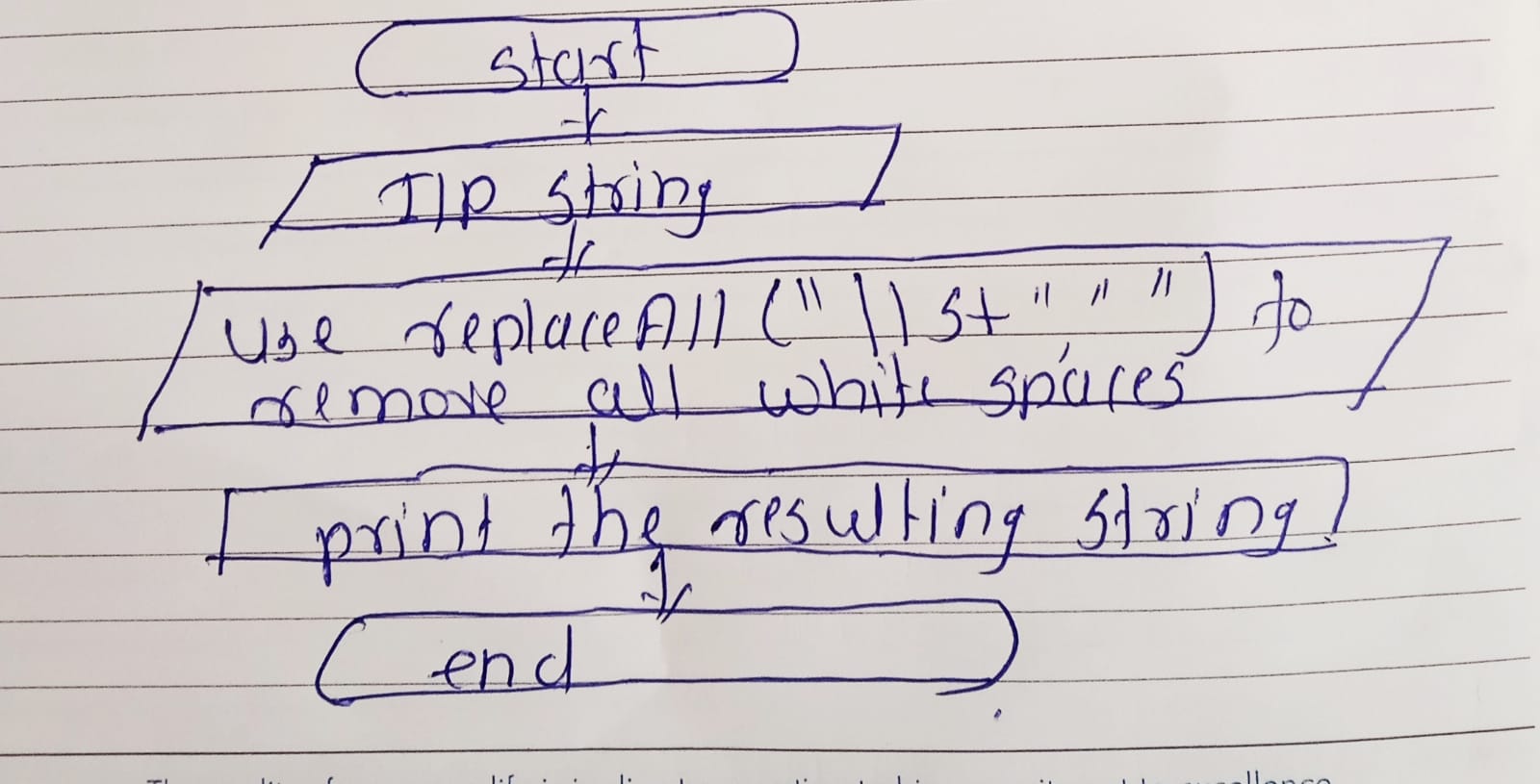
**Array after removing duplicates (Test Case 2): [0, 1, 2, 3, 2, 3]**

**Explanation:**

1. **The program takes a sorted array and removes duplicates in-place.**
2. **If the array is empty, the function returns 0.**
3. **We initialize uniqueIndex to track the position of unique elements.**
4. **We iterate through the array starting from index 1. If the current element is different from the element at uniqueIndex, we increment uniqueIndex and update the array at that position.**
5. **After processing all elements, the new length of the array is uniqueIndex + 1, as it represents the count of unique elements.**

**Time Complexity: O(n)**

**Space Complexity:O(1)**

 **flowchart**

**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 code:**

**package Star.pattern;**

**import java.util.Scanner;**

**public class RemoveWhiteSpaces {**

**public static String removeSpaces(String str) {**

**return str.replaceAll("\\s+", "");**

**}**

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

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

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

**String input = sc.nextLine();**

**String result = *removeSpaces*(input);**

**System.*out*.println("String after removing white spaces: " + result);**

**sc.close();**

**}**

**}**

**Output:**

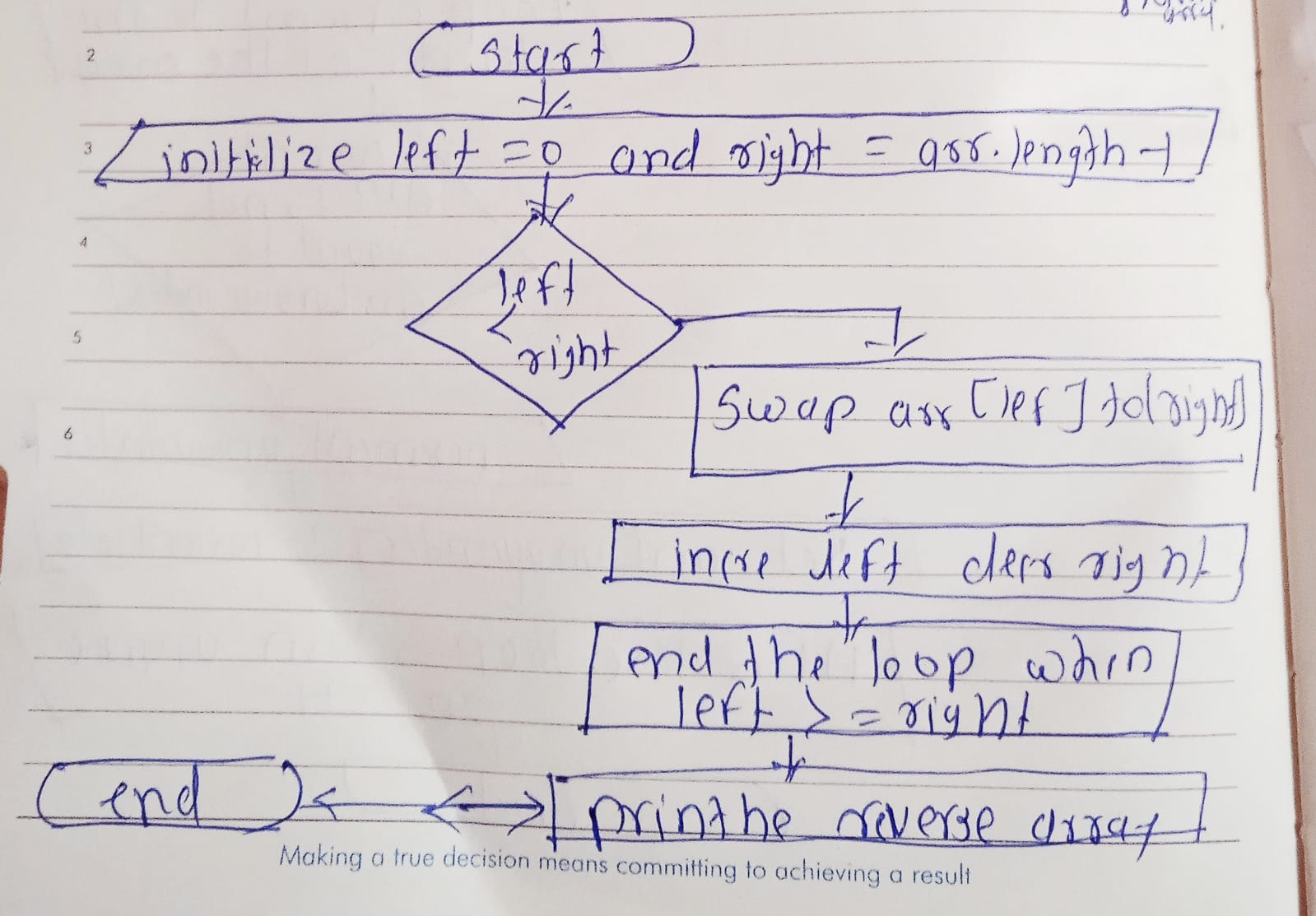
**Enter a string: Hello World**

**String after removing white spaces: HelloWorld**

**Explanation:**

1. **The program takes a string input from the user.**
2. **It uses the replaceAll() method, which matches all occurrences of whitespace characters (\\s+) and replaces them with an empty string.**
3. **The resulting string is printed, with all spaces removed.**

**Time and Space Complexity:O(n)**

 **flowchart**

**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 code:**

**package Star.pattern;**

**import java.util.Scanner;**

**public class ReverseString {**

**public static String reverse(String str) {**

**StringBuilder reversed = new StringBuilder(str);**

**return reversed.reverse().toString();**

**}**

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

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

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

**String input = sc.nextLine();**

**String result = *reverse*(input);**

**System.*out*.println("Reversed string: " + result);**

**sc.close();**

**}**

**}**

**Output:**

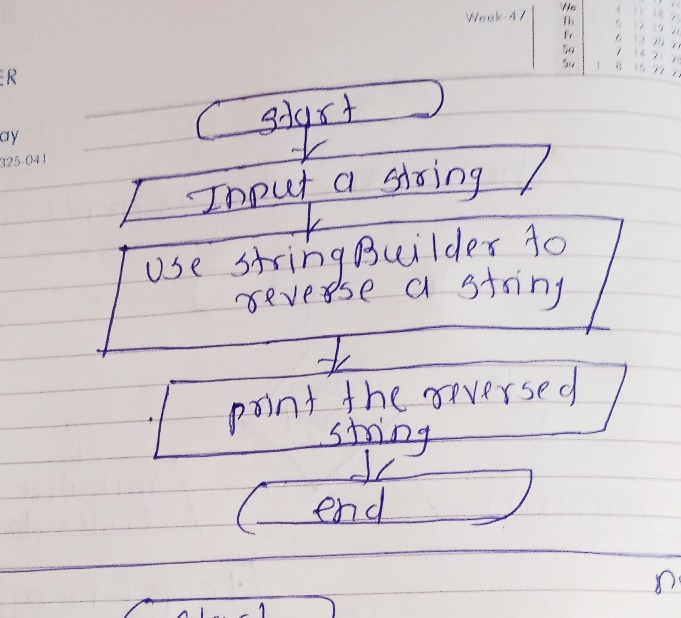
**Enter a string: Hello**

**Reversed string: olleH**

**Explanation:**

1. **The program takes a string as input from the user.**
2. **The reverse() method uses the StringBuilder class, which has a built-in reverse() function that reverses the string in place.**
3. **The reversed string is then returned and printed.**
4. **The reverse() method of StringBuilder is efficient as it modifies the string in place without creating extra space for each character.**

**Time and Space Complexity:O(n)**

 **flowchart**

**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 code:**

**package Star.pattern;**

**import java.util.Arrays;**

**public class ReverseArray {**

**public static void reverseArray(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) {**

**int[] arr1 = {1, 2, 3, 4};**

**int[] arr2 = {7, 8, 9};**

**System.*out*.println("Array before reverse (Test Case 1): " + Arrays.*toString*(arr1));**

***reverseArray*(arr1);**

**System.*out*.println("Array after reverse (Test Case 1): " + Arrays.*toString*(arr1));**

**System.*out*.println("Array before reverse (Test Case 2): " + Arrays.*toString*(arr2));**

***reverseArray*(arr2);**

**System.*out*.println("Array after reverse (Test Case 2): " + Arrays.*toString*(arr2));**

**}**

**}**

**Output:**

**Array before reverse (Test Case 1): [1, 2, 3, 4]**

**Array after reverse (Test Case 1): [4, 3, 2, 1]**

**Array before reverse (Test Case 2): [7, 8, 9]**

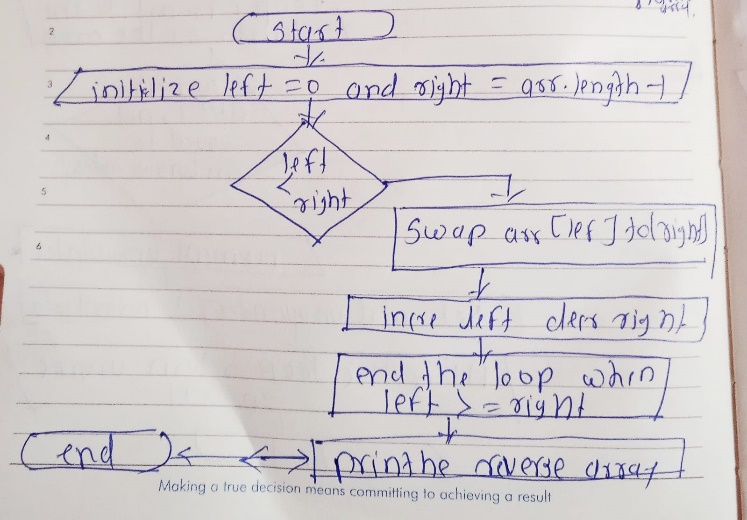
**Array after reverse (Test Case 2): [9, 8, 7]**

**Explanation:**

1. **The program takes an integer array as input.**
2. **Two pointers, left and right, are used to point to the first and last elements of the array.**
3. **While left is less than right, the elements at these two positions are swapped.**
4. **The left pointer is incremented, and the right pointer is decremented to move inward toward the center of the array.**
5. **This continues until the pointers meet in the middle, and the array is fully reversed.**

**Time Complexity: O(n)**

**Space Complexity:O(1)**

 **flowchart**

**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 code:**

**package Star.pattern;**

**import java.util.Scanner;**

**public class ReverseWords {**

**public static String reverseWords(String str) {**

**String[] words = str.trim().split("\\s+");**

**StringBuilder reversedString = new StringBuilder();**

**for (int i = words.length - 1; i >= 0; i--) {**

**reversedString.append(words[i]).append(" ");**

**}**

**return reversedString.toString().trim();**

**}**

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

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

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

**String input = sc.nextLine();**

**String result = *reverseWords*(input);**

**System.*out*.println("Reversed words: " + result);**

**sc.close();**

**}**

**}**

**Output:**

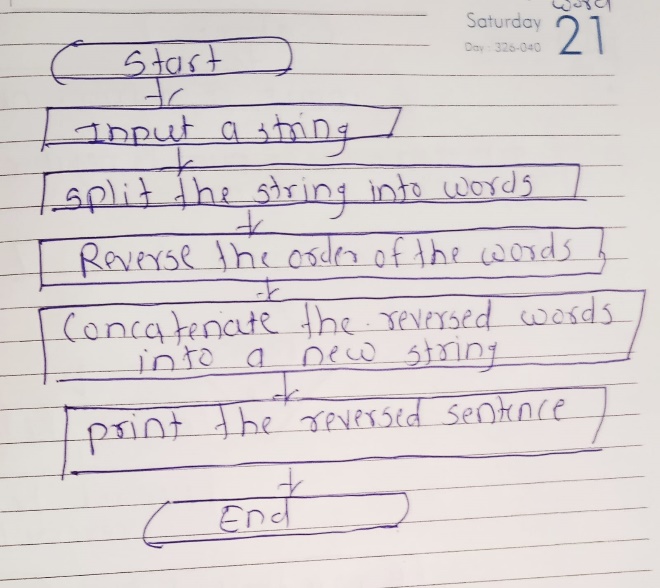
**Enter a sentence: Hello World**

**Reversed words: World Hello**

**Explanation:**

1. **The program takes a sentence as input.**
2. **The reverseWords() function splits the input string into words using the split("\\s+") method, which divides the string by one or more spaces.**
3. **We use a loop to iterate through the array of words in reverse order and append them to a new string.**
4. **The result is returned as a string of words in reverse order, trimmed to remove any extra spaces.**
5. **The reversed sentence is then printed.**

**Time and Space Complexity:O(n)**

 **flowchart**

**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 code:**

**package Star.pattern;**

**import java.util.Scanner;**

**public class ReverseNumber {**

**public static int reverseNumber(int num) {**

**int reversed = 0;**

**int sign = num < 0 ? -1 : 1;**

**num = Math.*abs*(num);**

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

**int digit = num % 10;**

**reversed = reversed \* 10 + digit;**

**num /= 10;**

**}**

**return reversed \* sign;**

**}**

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

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

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

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

**int result = *reverseNumber*(num);**

**System.*out*.println("Reversed number: " + result);**

**sc.close();**

**}**

**}**

**Output code:**

**Enter a number: -8765**

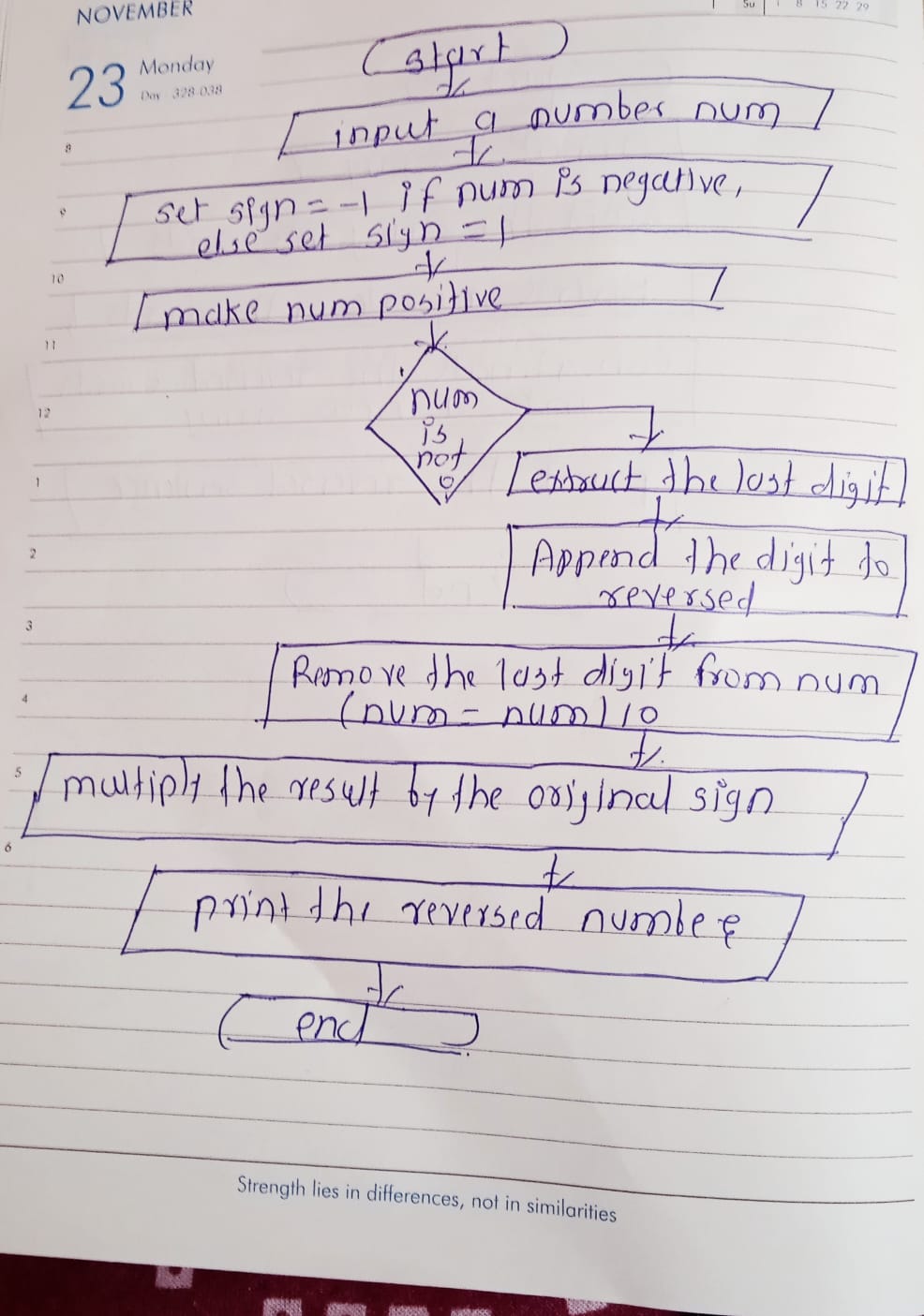
**Reversed number: -5678**

**Explanation:**

1. **The program reads an integer from the user.**
2. **We check whether the number is negative or positive by using a sign variable.**
3. **The number is reversed digit by digit:**
   * **Extract the last digit using num % 10.**
   * **Append this digit to the reversed number by multiplying the current reversed number by 10 and adding the extracted digit.**
   * **Remove the last digit from num by dividing it by 10.**
4. **After processing all digits, we multiply the reversed number by sign to restore the original sign.**
5. **The result is printed.**

**Time Complexity:O(d)**

**Space Complexity:O(1)**

 **flowchart**

**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 code:**

**package Star.pattern;**

**import java.util.Scanner;**

**public class ArrayManipulation {**

**public static long arrayManipulation(int n, int[][] queries) {**

**long[] arr = new long[n + 1]; // Use a 1-indexed array for easier range manipulation**

**for (int[] query : queries) {**

**int start = query[0] - 1;**

**int end = query[1] - 1;**

**int value = query[2];**

**arr[start] += value;**

**if (end + 1 < n) {**

**arr[end + 1] -= value;**

**}**

**}**

**long max = 0;**

**long current = 0;**

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

**current += arr[i];**

**max = Math.*max*(max, current);**

**}**

**return max;**

**}**

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

**// Test case 1**

**int n1 = 5;**

**int[][] queries1 = { {1, 2, 100}, {2, 5, 100}, {3, 4, 100} };**

**System.*out*.println("Max value after manipulations: " + *arrayManipulation*(n1, queries1));**

**// Test case 2**

**int n2 = 4;**

**int[][] queries2 = { {1, 3, 50}, {2, 4, 70} };**

**System.*out*.println("Max value after manipulations: " + *arrayManipulation*(n2, queries2));**

**}**

**}**

**Output:**

**Max value after manipulations: 200**

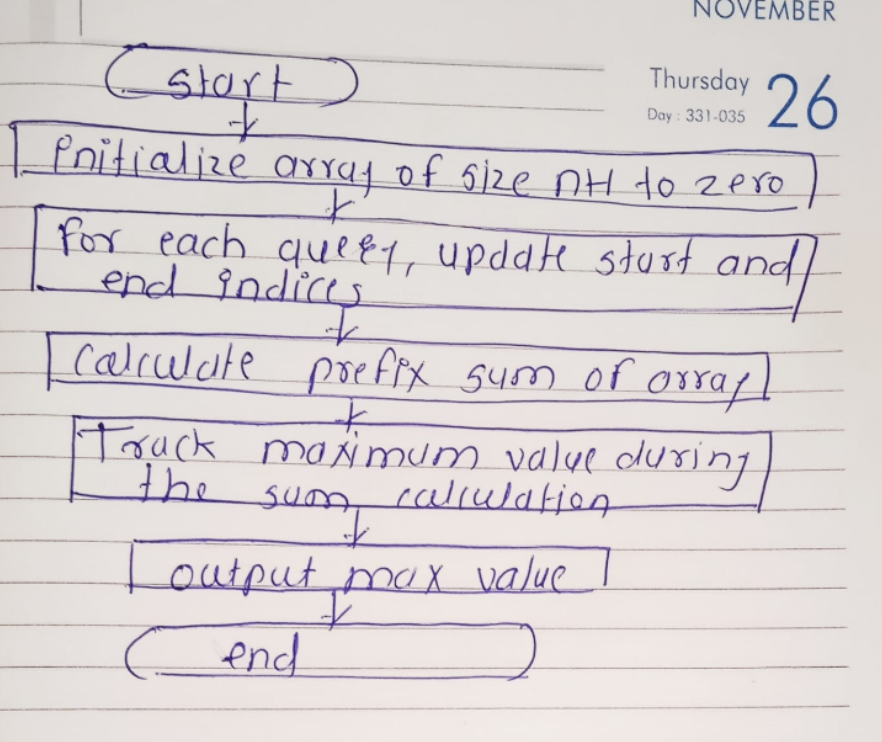
**Max value after manipulations: 120**

**Explanation:**

1. **Input: The program accepts an integer n (the size of the array) and a list of queries, where each query consists of three values: a start index, an end index, and a value to add to the range.**
2. **Logic:**
   * **Use a difference array technique to efficiently update the range. Instead of updating all elements between start and end indices for each query, the difference is only applied at the start index, and a negation is applied just after the end index.**
   * **After processing all the queries, the array is traversed, computing the prefix sum to calculate the final values at each index. The maximum value is tracked during this traversal.**

**Time Complexity: O(n + m)**

**Space Complexity: O(n)**

** flowchart**

**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 code:**

**package Star.pattern;**

**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*);**

**String input1 = "madam";**

**System.*out*.println("Is \"" + input1 + "\" a palindrome? " + *isPalindrome*(input1));**

**String input2 = "hello";**

**System.*out*.println("Is \"" + input2 + "\" a palindrome? " + *isPalindrome*(input2));**

**scanner.close();**

**}**

**}**

**Output:**

**Is "madam" a palindrome? true**

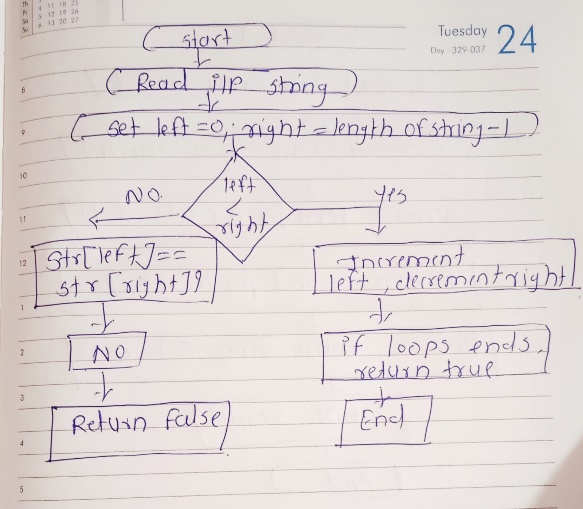
**Is "hello" a palindrome? false**

**Explanation**

1. **Input: The program accepts a string and checks whether it is a palindrome or not.**
2. **Logic:**
   * **Two pointers (left and right) are initialized, one starting from the beginning of the string and the other from the end.**
   * **The characters at both ends are compared. If they are not equal, the string is not a palindrome.**
   * **The pointers move towards the center until they meet or cross each other.**
   * **If all the characters from both ends match, the string is a palindrome.**
3. **Output:**
   * **For the string "madam", the program outputs true because it reads the same forward and backward.**
   * **For the string "hello", the program outputs false because the characters do not match in reverse.**

**Time Complexity: O(n)**

**Space Complexity:O(1)**



**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 code:**

**package Star.pattern;**

**import java.util.Arrays;**

**public class ArrayLeftRotation {**

**public static void rotateLeft(int[] arr, int d) {**

**int n = arr.length;**

**d = d % n;**

***reverseArray*(arr, 0, d - 1);**

***reverseArray*(arr, d, n - 1);**

***reverseArray*(arr, 0, n - 1);**

**}**

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

**int[] arr1 = {1, 2, 3, 4, 5};**

**int d1 = 2;**

***rotateLeft*(arr1, d1);**

**System.*out*.println("Rotated array: " + Arrays.*toString*(arr1));**

**int[] arr2 = {10, 20, 30, 40};**

**int d2 = 1;**

***rotateLeft*(arr2, d2);**

**System.*out*.println("Rotated array: " + Arrays.*toString*(arr2));**

**}**

**}**

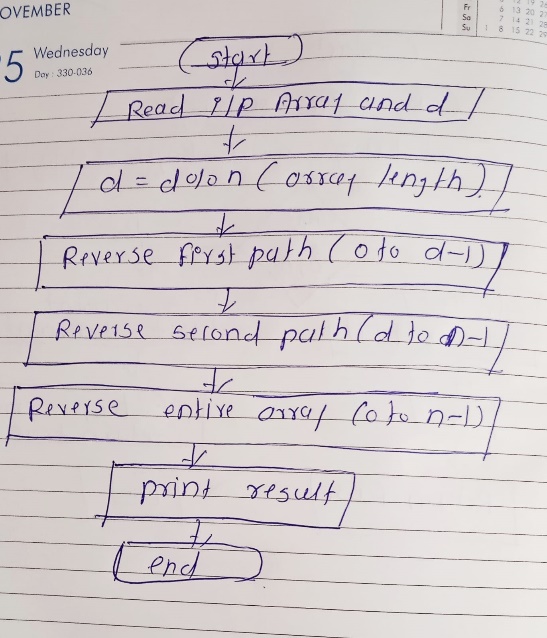
**Output code:**

**Rotated array: [3, 4, 5, 1, 2]**

**Rotated array: [20, 30, 40, 10]**

**Explanation**

1. **Input: An array and a number d, representing the number of left rotations.**
2. **Logic:**
   * **The problem can be solved by reversing parts of the array:**
     1. **Reverse the first d elements.**
     2. **Reverse the remaining elements.**
     3. **Reverse the entire array.**
   * **This method effectively shifts the elements left by d positions.**
3. **Time Complexity: O(n)**
4. **Space Complexity:O(1)**

**flochart**