ASSIGNMENT 2

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

Input: n=5

```
package com.assignment;

import java.util.Scanner;

public class Solution1 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("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();
}
}
```

```
Enter the number of rows: 3
*
* *
* * *
Enter the number of rows: 5
*
* *
* * *
* * * *
* * * * *
```

Flowchart

1. Start
2. Input number of rows (n)
3. Initialize loop `i = 1` to n
    a. Initialize nested loop `j = 1` to i
        i. Print star (*)
    b. Print new line
4. Repeat until all rows are printed
5. End

Explanation

- take the number of rows n as input from the user.
- A nested loop is used:
    - The outer loop runs from 1 to n for each row.

    - The inner loop runs from 1 to `i` (current row number) to print stars in that row.
- After printing the stars for each row, a newline is printed to move to the next row.
- **Time Complexity**: O(n^2)
    - The outer loop runs n times, and the inner loop runs `i` times for each row, resulting in approximately n`*(`n+1`)/`2 iterations.
- **Space Complexity**: O(1)
    - The program only uses a few variables (ignoring input/output), making it constant space.

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

```
package com.assignment;

public class Solution2 {
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++;
```

```java
arr[uniqueIndex] = arr[i];
}
}
return uniqueIndex + 1;
}
public static void main(String[] args) {
int[] arr = {0, 0, 1, 1, 2, 2, 3, 3};
int newLength = removeDuplicates(arr);
System.out.println("New length: " + newLength);
}
}
```

`arr = [0, 0, 1, 1, 2, 2, 3, 3]`:

New length: 4

`arr = [1, 1, 2]`:

New length: 2

Flowchart

1. Start
2. Initialize `uniqueIndex = 0`
3. Loop through the array:
    a. If current element is not equal to the element at `uniqueIndex`, update the next unique position.
4. Return the new length (`uniqueIndex + 1`).
5. End

Explanation

- We traverse the sorted array once.
- The index `uniqueIndex` keeps track of the position to place the next unique element.
- When we find a new unique element, we increment `uniqueIndex` and update the array in place.

Time Complexity

- **Time Complexity**: O(n)
  - The array is traversed only once.
- **Space Complexity**: O(1)
  - No extra space is used apart from variables for the index.

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"

```
package com.assignment;

import java.util.Scanner;

public class Solution3 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a string: ");
String input = scanner.nextLine();

String result = input.replaceAll("\\s", "");

System.out.println("String without white spaces: " + result);
scanner.close();
}
}
```

```
Enter a string: HelloWorld
String without white spaces: HelloWorld
Enter a string: Java Programming
String without white spaces: JavaProgramming
```

Flowchart

1. Start
2. Input string from user
3. Use `replaceAll("\\s", "")` to remove all white spaces
4. Print result
5. End

Explanation

- The program takes a string as input.
- The `replaceAll("\\s", "")` method removes all white spaces in the string by using a regular expression `\\s`, which matches any whitespace character (spaces, tabs, etc.).

Time Complexity

- **Time Complexity**: O(n), where n is the length of the input string.
- **Space Complexity**: O(n), as the new string without spaces is stored.

4. Reverse a String

Problem: Write a Java program to reverse a given string.

Test Cases:

Input: "hello"

Output: "olleh"

Input: "Java"

Output: "avaJ"

```
package com.assignment;

import java.util.Scanner;

public class Solution4 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a string: ");
String input = scanner.nextLine();

StringBuilder reversed = new StringBuilder(input);
reversed.reverse();

System.out.println("Reversed string: " + reversed.toString());
scanner.close();
}
}
```

```
Enter a string: hello
Reversed string: olleh
Enter a string: Java
Reversed string: avaJ
```

Flowchart

1. Start
2. Input string from user
3. Use `StringBuilder.reverse()` to reverse the string
4. Print reversed string
5. End

Explanation

- The program reads the input string.
- It uses `StringBuilder` to reverse the string in place using the `reverse()` method, which is efficient for string manipulation.

Time Complexity

- **Time Complexity**: O(n), where n is the length of the string.
- **Space Complexity**: O(n), as a new reversed string is created

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]

```java
package com.assignment;

import java.util.Arrays;

public class Solution5 {
public static void reverseArray(int[] arr) {
int start = 0;
int end = arr.length - 1;

while (start < end) {
int temp = arr[start];
arr[start] = arr[end];
arr[end] = temp;

start++;
end--;
}
}
public static void main(String[] args) {
int[] arr = {1, 2, 3, 4};
reverseArray(arr);
System.out.println("Reversed array: " + Arrays.toString(arr));
}
}
Reversed array: [4, 3, 2, 1]

Reversed array: [9, 8, 7]
```

Flowchart

1. Start
2. Initialize two pointers (`start`, end)
3. Swap elements at `start` and end
4. Move pointers towards the center
5. Repeat until `start >= end`
6. End

Explanation

- Two pointers (`start` and end) are used to traverse the array from both ends.
- The elements at these positions are swapped, and the process continues until the pointers meet in the middle.

Time Complexity

- **Time Complexity**: O(n), where n is the length of the array.
- **Space Complexity**: O(1), as we are modifying the array in place.

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"

```
package com.assignment;
```

```
import java.util.Scanner;

public class Solution6 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a sentence: ");
String sentence = scanner.nextLine();

String[] words = sentence.split(" ");
StringBuilder reversedSentence = new StringBuilder();

for (int i = words.length - 1; i >= 0; i--) {
reversedSentence.append(words[i]).append(" ");
}

System.out.println("Reversed words: " +
reversedSentence.toString().trim());
scanner.close();
}
}
```

```
Enter a sentence: Hello World
Reversed words: World Hello
Enter a sentence: Java Programming
Reversed words: Programming Java
```

Flowchart

1. Start
2. Input sentence from user
3. Split the sentence into words
4. Reverse the order of words
5. Print the reversed sentence
6. End

Explanation

- The sentence is split into words using the `split(" ")` method.

- The words are then reversed and concatenated to form the final sentence.

Time Complexity

- **Time Complexity**: O(n), where n is the length of the sentence.
- **Space Complexity**: O(n), as a new string is created for the reversed sentence.

7. Reverse a Number

Problem: Write a Java program to reverse a given number.

Test Cases:

Input: 12345

Output: 54321

Input: -9876

Output: -6789

```
package com.assignment;

import java.util.Scanner;

public class Solution7 {
public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a number: ");
int num = scanner.nextInt();

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

System.out.println("Reversed number: " + reversed);
scanner.close();
}
}
```

```
Enter a number: 12345
Reversed number: 54321
Enter a number: -6789
Reversed number: -9876
```

Flowchart

1. Start
2. Input number
3. Initialize reversed = 0
4. Extract digits one by one and append to reversed
5. Output the reversed number
6. End

Explanation

- The number is reversed by repeatedly extracting the last digit using modulus (`% 10`) and appending it to the reversed number.

Time Complexity

- **Time Complexity**: O(d), where d is the number of digits in the number.
- **Space Complexity**: O(1), as no extra space is used apart from a few variables.

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

```java
package com.assignment;


public class Solution8 {
public static long arrayManipulation(int n, int[][] queries) {
long[] arr = new long[n + 1];
for (int[] query : queries) {
int start = query[0] - 1;
int end = query[1];
int value = query[2];

arr[start] += value;
if (end < n) arr[end] -= value;
}

long max = 0;
long current = 0;
for (long val : arr) {
current += val;
if (current > max) {
max = current;
}
}

return max;
}

public static void main(String[] args) {
int[][] queries = {{1, 2, 100}, {2, 5, 100}, {3, 4, 100}};
int n = 5;
System.out.println("Max value after operations: " +
arrayManipulation(n, queries));
}
}
```

```
Max value after operations: 200
Max value after operations: 120
```

Flowchart

1. Start
2. Initialize an array `arr` of size `n+1` to zeros.
3. For each query, update the start index by adding the value and the end index by subtracting the value.
4. Traverse the array and calculate the prefix sum to find the maximum value.
5. Output the maximum value.
6. End

Explanation

- We use the **difference array** technique:
  - For each query `(start, end, value)`, we add `value` at `start` and subtract `value` at `end+1`.
  - This way, during prefix sum traversal, we simulate adding the `value` to the entire range in the query.
- After processing all the queries, the prefix sum of the array will give us the desired array, and we can find the maximum value.

Time Complexity

- **Time Complexity**: O(n + m), where n is the length of the array and `m` is the number of queries.
  - Each query is processed in constant time, and we traverse the array once for the prefix sum.
- **Space Complexity**: O(n), for the auxiliary array used to store the range updates.

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

```java
package com.assignment;

import java.util.Scanner;

public class Solution9 {
public static boolean isPalindrome(String str) {
int start = 0;
int end = str.length() - 1;

while (start < end) {
if (str.charAt(start) != str.charAt(end)) {
return false;
}
start++;
end--;
}

return true;
}

public static void main(String[] args) {
Scanner scanner = new Scanner(System.in);
System.out.print("Enter a string: ");
String input = scanner.nextLine();

if (isPalindrome(input)) {
System.out.println(input + " is a palindrome");
} else {
System.out.println(input + " is not a palindrome");
}
scanner.close();
}
}
```

```
Enter a string: madam
madam is a palindrome
Enter a string: hello
hello is not a palindrome
```

Flowchart

1. Start
2. Input string from user
3. Initialize two pointers (`start`, end) at the beginning and end of the string.
4. Compare characters at `start` and end:
    a. If they are not equal, return false.
    b. Move `start` forward and end backward.
5. If the entire string is traversed and all characters match, return true.
6. Output the result.
7. End

Explanation

- The program uses two pointers to check if the string is the same when read forward and backward.
- The two pointers start from opposite ends and move towards the center, comparing characters at each position.

Time Complexity

- **Time Complexity**: O(n), where n is the length of the string.
- **Space Complexity**: O(1), as we only use a few variables to store indices.

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]

```java
package com.assignment;

import java.util.Arrays;

public class Solution10 {
public static int[] rotateLeft(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);
return arr;
}

private 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 main(String[] args) {
int[] arr = {1, 2, 3, 4, 5};
int d = 2;

System.out.println("Array after left rotation: " +
Arrays.toString(rotateLeft(arr, d)));
}
}
```

```
Array after left rotation: [3, 4, 5, 1, 2]
```

```
Array after left rotation: [20, 30, 40, 10]
```

Flowchart

1. Start
2. Input array and number of rotations d.
3. Reverse the first part of the array (from 0 to d-1).
4. Reverse the second part of the array (from d to n-1).
5. Reverse the entire array.
6. Output the rotated array.
7. End

Explanation

- We use the **reversal algorithm** for array rotation:
    - Reverse the first d elements.
    - Reverse the remaining n - d elements.
    - Reverse the entire array.
- This gives the desired left-rotated array.

Time Complexity

- **Time Complexity**: O(n), where n is the length of the array.
    - Each reverse operation takes linear time.
- **Space Complexity**: O(1), as no extra space is used except for temporary variables.