**Task 1 – list explorer**

**Algorithm**1. Create an integer array to store temperatures.

2. Initialize it with a set of sample values.

3. Print all values in the array.

4. Calculate the sum of all values.

5. Compute the average (sum divided by number of elements).

6. Find the highest value in the array.

**Psudocode**

DECLARE array of integers

INITIALIZE array with temperature values

FOR each element in array

PRINT value

ADD to sum

IF value > max, UPDATE max

CALCULATE average = sum / number of elements

PRINT sum, average, and highest temperature

**Code(java)  
public class App {**

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

**int [] temperatures = {30, 32, 31, 29, 28, 27, 26, 25, 24, 23};**

**int highest = Integer.MIN\_VALUE;**

**int sum = 0;**

**for (int i=0; i<temperatures.length; i++) {**

**if (highest < temperatures[i]) {**

**highest = temperatures[i];**

**}**

**sum += temperatures[i];**

**System.out.println("Temperature at index " + i + ": " + temperatures[i]);**

**}**

**double average = (double) sum / temperatures.length;**

**System.out.println("sum: " + sum);**

**System.out.println("Average temperature: " + average);**

**System.out.println("Highest temperature: " + highest);**

**}**

**}**

**Output (test case 1)**

mUp\Bridge-course\day-7\task - 1\List explorer'; & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 1\List explorer\bin' 'App'

NIVASA\x5cDocuments\x5cStemUp\x5cBridge-course\x5cday-7\x5ctask - 1\x5cList explorer\x5cbin' 'App' ;0fbe4a85-635d-46ca-82e8-92f240080922Temperature at index 0: 30

Temperature at index 1: 32

Temperature at index 2: 31

Temperature at index 3: 29

Temperature at index 4: 28

Temperature at index 5: 27

Temperature at index 6: 26

Temperature at index 7: 25

Temperature at index 8: 24

Temperature at index 9: 23

sum: 275

Average temperature: 27.5

Highest temperature: 32

**Observations**

- Demonstrates use of arrays and enhanced for-loop.

- Applies basic aggregation operations: sum, average, and max.

- Highlights practical use of arrays in data processing.

**Task 2 – product of evens**

**Algorithm**1. Create an integer array containing numbers from 1 to 10.

2. Initialize a variable to store the product (start with 1).

3. Traverse the array using a loop.

4. For each number:

- Check if it is even (number % 2 == 0).

- If it is, multiply it to the product.

5. Print the final product.

**Psudocode**

DECLARE array of integers from 1 to 10

SET product = 1

FOR each number in array

IF number is even

MULTIPLY number with product

PRINT product

**Code(java)**

**public class ProductOfEvens {**

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

**int[] numbers = {1,2,3,4,5,6,7,8,9,10};**

**int product = 1;**

**for (int num : numbers) {**

**if (num % 2 == 0) {**

**product \*= num;**

**}**

**}**

**System.out.println("Product of all even numbers: " + product);**

**}**

**}**

**Output (test case 1)**

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 2\Product of evens> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 2\Product of evens\bin' 'ProductOfEvens'

c8e-a49a-4e53-a156-619ba0ca2ccaProduct of all even numbers: 3840

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 2\Product of evens>

**Output (test case 2)**

**Output (test case 3)**

**Observations**

- Demonstrates use of conditionals inside loops.

- Efficiently filters even numbers using modulo operator.

- Shows how to compute a cumulative product using an array.

**Task 3 – reverse array**

**Algorithm**1. Create a string array and initialize it with a list of items.

2. Print the elements in reverse order using a loop.

**Psudocode**

DECLARE string array with items

FOR index from array.length - 1 down to 0

PRINT item at current index

**Code(java)  
public class ReverseList {**

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

**String[] items = {"Apple", "Banana", "Cherry", "Date", "Elderberry"};**

**System.out.println("Items in reverse order:");**

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

**System.out.println(items[i]);**

**}**

**}**

**}**

**Output (test case 1)**

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\SRINIVASA\AppData\Roaming\Code\User\workspaceStorage\5f3cf6ecac6b70dc66c3a5118d392f79\redhat.java\jdt\_ws\Bridge-course\_6c8d8adb\bin' 'ReverseList'

jdt\_ws\x5cBridge-course\_6c8d8adb\x5cbin' 'ReverseList' ;95fea55a-5bb8-43f3-8f80-25330122fe7eItems in reverse order:

Elderberry

Date

Cherry

Banana

Apple

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course>

**Observations**

- Demonstrates basic string array usage and reverse iteration.

- Shows how to control loop direction using index manipulation.

- Reinforces how array indexes work in Java (0-based).

**Task 4 – Word search**

**Algorithm**1. Create a predefined string array with a few words.

2. Take a word as input from the user using Scanner.

3. Search for the word in the array:

- Use a loop to compare each element with the input word.

4. If a match is found, set a flag to true and break the loop.

5. After the loop, print whether the word was found.

**Psudocode**

DECLARE string array with some words

READ word from user input

SET found = false

FOR each word in array

IF word equals input (ignore case)

SET found = true

BREAK loop

IF found

PRINT "Word found"

ELSE

PRINT "Word not found"

**Code(java)  
import java.util.Scanner;**

**public class WordSearch {**

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

**String[] words = {"apple", "banana", "grape", "mango", "orange"};**

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

**System.out.print("Enter a word to search: ");**

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

**boolean found = false;**

**for (String word : words) {**

**if (word.equalsIgnoreCase(input)) {**

**found = true;**

**break;**

**}**

**}**

**if (found) {**

**System.out.println("Word found in the list.");**

**} else {**

**System.out.println("Word not found in the list.");**

**}**

**scanner.close();**

**}**

**}**

**Output (test case 1)**

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 4\word search> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 4\word search\bin' 'WordSearch'

4e2b-8642-37daa287c0a4Enter a word to search: mango

Word found in the list.

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 4\word search>

**Output (test case 2)**

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 4\word search> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 4\word search\bin' 'WordSearch'

4e2b-8642-37daa287c0a4Enter a word to search: water

Word not found in the list.

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 4\word search>

**Observations**

- Uses linear search to find a match in a string array.

- Demonstrates user input, case-insensitive comparison, and control flow with flags.

- Practical for implementing search features in lists or databases.

**Task 5 – GCD**

**Algorithm**1. Define a method to calculate GCD using the Euclidean algorithm.

2. Take two integers as input.

3. Repeat until the second number becomes zero:

- Set temp = b

- Set b = a % b

- Set a = temp

4. Return a as the GCD.

5. Test with multiple input pairs.

**Psudocode**FUNCTION gcd(a, b)

WHILE b != 0

temp = b

b = a % b

a = temp

RETURN a

MAIN

CALL gcd with various input pairs

PRINT results

**Code(java)  
public class GCDCalculator {**

**public static int gcd(int a, int b) {**

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

**int temp = b;**

**b = a % b;**

**a = temp;**

**}**

**return a;**

**}**

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

**System.out.println("GCD of 20 and 30: " + gcd(20, 30));**

**System.out.println("GCD of 81 and 27: " + gcd(81, 27));**

**System.out.println("GCD of 13 and 7: " + gcd(13, 7));**

**}**

**}**

**Output (test case 1)**

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 5\GCD> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 5\GCD\bin' 'GCDCalculator'

b446-81ca890aeedaGCD of 20 and 30: 10

GCD of 81 and 27: 27

GCD of 13 and 7: 1

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 5\GCD>

**Observations**

- Demonstrates an efficient method (Euclidean Algorithm) for finding GCD.

- The algorithm uses modulo to reduce the problem until one number is zero.

- The GCD is useful when simplifying fractions or reducing ratios.

**Task 6 – LCM**

**Algorithm**1. Use the formula: LCM(a, b) = (a × b) / GCD(a, b)

2. Define a method to calculate LCM that:

- Calls the existing GCD method

- Returns (a \* b) / gcd

3. Test it with the same input pairs used in the GCD program.

**Psudocode**

FUNCTION gcd(a, b)

WHILE b ≠ 0

temp = b

b = a % b

a = temp

RETURN a

FUNCTION lcm(a, b)

RETURN (a \* b) / gcd(a, b)

MAIN

CALL lcm with various input pairs

PRINT results

**Code(java)  
public class LCMCalculator {**

**public static int gcd(int a, int b) {**

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

**int temp = b;**

**b = a % b;**

**a = temp;**

**}**

**return a;**

**}**

**public static int lcm(int a, int b) {**

**return (a \* b) / gcd(a, b);**

**}**

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

**System.out.println("LCM of 20 and 30: " + lcm(20, 30));**

**System.out.println("LCM of 81 and 27: " + lcm(81, 27));**

**System.out.println("LCM of 13 and 7: " + lcm(13, 7));**

**}**

**}**

**Output (test case 1)**

PS C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 6\LCM> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\SRINIVASA\Documents\StemUp\Bridge-course\day-7\task - 6\LCM\bin' 'LCMCalculator'

b8ec-cdc29a8821b7LCM of 20 and 30: 60

LCM of 81 and 27: 81

LCM of 13 and 7: 91

**Observations**

- Demonstrates reusability by building LCM on top of GCD.

- LCM is helpful when aligning intervals or synchronizing events.

- Ensures minimum common multiple for real-world repetitive tasks.

**Task 7 –**

**Algorithm**

**Psudocode**

**Code(java)**

**Output (test case 1)**

**Observations**