

Digvijay Thakare (digvijaythakare2017@gmail.com)

Day 4 Java Assignments

Task 1: Array Sorting and Searching

a) Implement a function called BruteForceSort that sorts an array using the brute force approach. Use this function to sort an array created with InitializeArray.

b) Write a function named PerformLinearSearch that searches for a specific element in an array and returns the index of the element if found or -1 if not found.

Solution-

```
package com.wipro.ep;

import java.util.Random;

public class BruteForceSearchingAndSorting
{

    // Function to initialize an array with random integers
    public static int[] InitializeArray(int size) {
        int[] array = new int[size];
        Random random = new Random();
        for (int i = 0; i < size; i++) {
            array[i] = random.nextInt(100); // Random integers between 0 and
99
        }
        return array;
    }

    // Brute force sorting function
    public static void BruteForceSort(int[] array) {
        int n = array.length;
        for (int i = 0; i < n - 1; i++) {
            for (int j = i + 1; j < n; j++) {
```

```

        if (array[i] > array[j]) {
            // Swap array[i] and array[j]
            int temp = array[i];
            array[i] = array[j];
            array[j] = temp;
        }
    }
}

// Function to perform linear search
public static int PerformLinearSearch(int[] array, int target) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == target) {
            return i;
        }
    }
    return -1;
}

// Function to print the array
public static void printArray(int[] array) {
    for (int i : array) {
        System.out.print(i + " ");
    }
    System.out.println();
}

public static void main(String[] args) {
    int[] array = InitializeArray(10);
    System.out.println("Original Array:");
    printArray(array);

    BruteForceSort(array);
    System.out.println("Sorted Array:");
    printArray(array);

    int target = 50; // Example target to search
    int index = PerformLinearSearch(array, target);

```

```

        if (index != -1) {
            System.out.println("Element " + target + " found at index " +
index);
        } else {
            System.out.println("Element " + target + " not found in the array.");
        }
    }
}

```

Output-

```

Original Array:
50 5 71 49 3 16 43 43 43 95
Sorted Array:
3 5 16 43 43 43 49 50 71 95
Element 50 found at index 7

```

Task 2: Two-Sum Problem a) Given an array of integers, write a program that finds if there are two numbers that add up to a specific target. You may assume that each input would have exactly one solution, and you may not use the same element twice. Optimize the solution for time complexity.

Solution-

```

package com.wipro.ep;

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class TwoSumProblem {

    // Function to find two numbers that add up to a specific target
    public static int[] findTwoSum(int[] nums, int target) {
        // Create a hash map to store the numbers and their indices
        Map<Integer, Integer> map = new HashMap<>();
    }
}

```

```

        // Iterate through the array
        for (int i = 0; i < nums.length; i++) {
            // Calculate the complement of the current number
            int complement = target - nums[i];

            // Check if the complement is already in the map
            if (map.containsKey(complement)) {
                // Return the indices of the two numbers
                return new int[] { map.get(complement), i };
            }

            // Add the current number and its index to the map
            map.put(nums[i], i);
        }

        // If no solution is found, return an empty array
        return new int[] {};
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Input array
        System.out.print("Enter the number of elements in the array: ");
        int n = scanner.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements of the array:");
        for (int i = 0; i < n; i++) {
            nums[i] = scanner.nextInt();
        }

        // Target value
        System.out.print("Enter the target value: ");
        int target = scanner.nextInt();

        // Find the two numbers that add up to the target
        int[] result = findTwoSum(nums, target);
    }

```

```

        // Print the result
        if (result.length == 2) {
            System.out.println(
                "Indices of the two numbers that add up to " + target
+ ": " + result[0] + ", " + result[1]);
        } else {
            System.out.println("No two numbers found in the array that add
up to the target.");
        }

        scanner.close();
    }
}

```

Output-

```

Enter the number of elements in the array: 5
Enter the elements of the array:
7
2
6
4
5
Enter the target value: 10
Indices of the two numbers that add up to 10: 2, 3

```

Task 3: Understanding Functions through Arrays a) Write a recursive function named SumArray that calculates and returns the sum of elements in an array, demonstrate with example.

```
package com.wipro.ep;

public class SumArrayExample {
    public static int sumArray(int[] arr, int n) {
        if (n <= 0) {
            return 0;
        } else {
            return arr[n - 1] + sumArray(arr, n - 1);
        }
    }

    public static void main(String[] args) {
        int[] myArray = {4,3,7,5,2,6};
        int arraySize = myArray.length;
        int sum = sumArray(myArray, arraySize);
        System.out.println("Sum of array elements: " + sum);
    }
}
```

Explanation-

- The sumArray function takes an integer array arr and an integer n as parameters.
- If n is less than or equal to 0, the function returns 0 (base case).
- Otherwise, it recursively calculates the sum by adding the last element of the array (arr[n - 1]) to the sum of the remaining elements (sumArray(arr, n - 1)).

Output-

```
Sum of array elements: 27
```

Task 4: Advanced Array Operations

a) Implement a method `SliceArray` that takes an array, a starting index, and an end index, then returns a new array containing the elements from the start to the end index.

b) Create a recursive function to find the nth element of a Fibonacci sequence and store the first n elements in an array

Solution- A) Sliced Array

```
package com.wipro.ep;

import java.util.Arrays;

public class ArraySlicer {
    public static int[] sliceArray(int[] arr, int startIndex, int endIndex) {
        int sliceSize = endIndex - startIndex + 1;
        int[] slicedArray = new int[sliceSize];

        for (int i = 0; i < sliceSize; i++) {
            slicedArray[i] = arr[startIndex + i];
        }

        return slicedArray;
    }

    public static void main(String[] args) {
        int[] originalArray = {23, 56, 78, 22, 45, 90, 67, 91, 0, 31};
        int startIndex = 3;
        int endIndex = 8;

        int[] result = sliceArray(originalArray, startIndex, endIndex);
        System.out.println("Sliced Array: " + Arrays.toString(result));
    }
}
```

Output-

```
Sliced Array: [22, 45, 90, 67, 91, 0]
```

B-Finding nth Element of fibonacci using recursion.

```
package com.wipro.ep;

import java.util.Arrays;

public class FibonacciSeries {
    public static int fibonacci(int n) {
        if (n <= 1) {
            return n; // Base case: Fibonacci(0) = 0, Fibonacci(1) = 1
        } else {
            return fibonacci(n - 1) + fibonacci(n - 2); // Recursive call
        }
    }

    public static void main(String[] args) {
        int n = 10; // Find the first 10 Fibonacci numbers
        int[] fibonacciArray = new int[n];

        for (int i = 0; i < n; i++) {
            fibonacciArray[i] = fibonacci(i);
        }

        System.out.println("Fibonacci Series (first " + n + " elements): " +
            Arrays.toString(fibonacciArray));
    }
}
```

Output-

```
Fibonacci Series (first 10 elements): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```