Task 1: Iterators and Comparators

Write a custom Comparator to sort a list of Employee objects by their salary and then by name if the salary is the same.

```
import java.util.Comparator;

import java.util.List; import

java.util.ArrayList;


class Employee {

private String name;

private double salary;


   // Constructor, getters, and setters


   public Employee(String name, double salary) {

this.name = name;        this.salary = salary;

   }


   public String getName() {

return name;

   }


   public double getSalary() {

return salary;

   }

}


public class Main {
```

```
    public static void main(String[] args) {
        List<Employee> employees = new ArrayList<>();
employees.add(new Employee("John", 50000));        employees.add(new
Employee("Alice", 60000));        employees.add(new Employee("Bob",
50000));

        // Sort employees by salary first, then by name if salary is the same
employees.sort(
            Comparator.comparingDouble(Employee::getSalary)
                    .thenComparing(Employee::getName)
        );

        // Print sorted employees
        employees.forEach(emp -> System.out.println(emp.getName() + " - " +
emp.getSalary()));
    }
}
```

Output:

Bob - 50000.0

John - 50000.0

Alice - 60000.0

Task 2: 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.

```
import java.util.Arrays;

public class Main {
   public static void main(String[] args) {         int[] array =
initializeArray(10); // Create an array of size 10
       System.out.println("Original array: " + Arrays.toString(array));

       bruteForceSort(array); // Sort the array using brute force
       System.out.println("Sorted array: " + Arrays.toString(array));
   }

   // Function to initialize an array with random values
public static int[] initializeArray(int size) {         int[]
array = new int[size];         for (int i = 0; i < size; i++)
{
           array[i] = (int) (Math.random() * 100); // Random values 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 = 0; j < n - i - 1; j++) {                if
(array[j] > array[j + 1]) {                    // Swap
array[j] and array[j+1]                    int temp =
array[j];                    array[j] = array[j + 1];
array[j + 1] = temp;
            }
          }
       }
   }
} Output:
original array: [94, 15, 97, 1, 55, 31, 55, 69, 19, 44]
Sorted array: [1, 15, 19, 31, 44, 55, 55, 69, 94, 97]
```

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.

```
public class Main {     public static void
main(String[] args) {         int[] array = {
2, 5, 7, 9, 11, 15 };         int target = 11;

      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");
        }
    }

    // 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 the index of the target element if found
            }
        }
        return -1; // Return -1 if target element is not found
    }
}
```

Output:

Element 11 found at index 4

Task 3: 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.

```
import java.util.HashMap;
```

```
public class Main {     public static void
main(String[] args) {         int[] nums =
{2, 7, 11, 15};         int target = 9;


        int[] result = twoSum(nums, target);


        if (result != null) {
            System.out.println("Indices of the two numbers that add up to
target:");
            System.out.println("Index 1: " + result[0] + ", Index 2: " + result[1]);
} else {
            System.out.println("No solution found.");
        }
    }

    public static int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>();


        for (int i = 0; i < nums.length; i++) {             int
complement = target - nums[i];             if
(map.containsKey(complement)) {                 return
new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
```

```
        }


        return null; // If no solution found
    }
} Output:
indices of the two numbers that add up to target:
Index 1: 0, Index 2: 1
```

Task 4: Understanding Functions through Arrays

a) Write a recursive function named SumArray that calculates and returns the sum of elements in an array, demonstarte with example.

```
public class Main {
    public static void main(String[] args) {
int[] array = {1, 2, 3, 4, 5};

        int sum = sumArray(array);
        System.out.println("Sum of elements in the array: " + sum);
    }


    public static int sumArray(int[] array) {        return
sumArrayRecursive(array, array.length - 1);
    }
```

```
    private static int sumArrayRecursive(int[] array, int index) {
// Base case: if index is less than 0 (no elements left)        if
(index < 0) {            return 0;
        }

        // Recursive case: sum the current element with the sum of the rest
of the array
        return array[index] + sumArrayRecursive(array, index - 1);
    }
}
```

Output:

Sum of elements in the array: 15

Task 5: 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.

```
import java.util.Arrays; public class Main
{    public static void main(String[] args)
{        int[] array = {1, 2, 3, 4, 5, 6, 7, 8,
9};        int start = 2;        int end = 5;

        int[] slicedArray = sliceArray(array, start, end);
        System.out.println("Sliced array: " + Arrays.toString(slicedArray));
    }
```

```
    public static int[] sliceArray(int[] array, int start, int end) {
if (start < 0 || end < start || end >= array.length) {
            throw new IllegalArgumentException("Invalid start or end
index");
        }


        return Arrays.copyOfRange(array, start, end + 1);
    }
}
```

Output:

Sliced array: [3, 4, 5, 6]

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

```
public class Fibonacci {
    public static int fibonacci(int n, int[] arr) {        if (n ==
0) {            arr[n] = 0;            return 0;        } else if (n
== 1) {            arr[n] = 1;            return 1;        } else {
arr[n] = fibonacci(n - 1, arr) + fibonacci(n - 2, arr);
return arr[n];
        }
    }
```

```
    public static void main(String[] args) {
        int n = 10;         int[]
    arr = new int[n];
    fibonacci(n - 1, arr);
        System.out.println("The first " + n + " elements of the Fibonacci
    sequence are:");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Output:

The first 10 elements of the Fibonacci sequence are:

0 1 1 2 3 5 8 13 21 34

Task 6: Creating and Managing Threads

Write a program that starts two threads, where each thread prints numbers from 1 to 10 with a 1-second delay between each number

```
public class Main {
    public static void main(String[] args) {
        // Create and start the first thread
        Thread thread1 = new Thread(() -> printNumbers(1));
        thread1.start();
```

```
        // Create and start the second thread
        Thread thread2 = new Thread(() -> printNumbers(2));
        thread2.start();
    }

    public static void printNumbers(int threadId) {
        for (int i = 1; i <= 10; i++) {
            System.out.println("Thread " + threadId + ": " + i);
            try {
                Thread.sleep(1000); // 1-second delay
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Output:

```
Thread 1: 1
Thread 2: 1
Thread 1: 2
Thread 2: 2
Thread 2: 3
Thread 1: 3
```

Thread 2: 4

Thread 1: 4

Thread 1: 5

Thread 2: 5

Thread 1: 6

Thread 2: 6

Thread 2: 7

Thread 1: 7

Thread 2: 8

Thread 1: 8

Thread 2: 9

Thread 1: 9

Thread 2: 10

Thread 1: 10