CSW 2

ASSIGNMENT 4

Q1. import java.util.*;

```
public class FirstRepeatedElement {
    public static void main(String[] args) {
        // create an unsorted list of n elements
        List<Integer> list = Arrays.asList(3, 5, 1, 4, 3, 6, 2, 5,
7);
        int n = list.size();

        // use a set to keep track of the elements that have been
seen before
        Set<Integer> seen = new HashSet<>();

        // iterate over the list and check if each element has
been seen before
        for (int i = 0; i < n; i++) {
            int element = list.get(i);
```

```
            if (seen.contains(element)) {
                System.out.println("The first repeated element is
" + element);
                return;
            } else {
                seen.add(element);
            }
        }

        System.out.println("There are no repeated elements in
the list");
    }
}
```

Q2.

```
import java.util.*;

public class FindDuplicateElements {
    public static void main(String[] args) {
        int[] nums = {1, 2, 3, 4, 2, 5, 6, 7, 7, 8, 9, 9};
        Set<Integer> seen = new HashSet<Integer>();
        Set<Integer> duplicates = new HashSet<Integer>();

        for (int i = 0; i < nums.length; i++) {
```

```
            if (seen.contains(nums[i])) {
                duplicates.add(nums[i]);
            } else {
                seen.add(nums[i]);
            }
        }

        System.out.println("Duplicate elements in the array are: " + duplicates);
    }
}
```

Q3.
```
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

public class RemoveDuplicates {

    public static void main(String[] args) {
        List<Integer> listWithDuplicates = new ArrayList<>();
        listWithDuplicates.add(1);
```

```
        listWithDuplicates.add(2);
        listWithDuplicates.add(3);
        listWithDuplicates.add(1);
        listWithDuplicates.add(2);
        listWithDuplicates.add(4);

        List<Integer> listWithoutDuplicates =
removeDuplicates(listWithDuplicates);

        System.out.println("List with duplicates: " +
listWithDuplicates);
        System.out.println("List without duplicates: " +
listWithoutDuplicates);
    }

    public static List<Integer>
removeDuplicates(List<Integer> listWithDuplicates) {
        Set<Integer> setWithoutDuplicates = new
HashSet<>(listWithDuplicates);
        return new ArrayList<>(setWithoutDuplicates);
    }
}
```

Q4. public class FindMissingElement {

```
    public static void main(String[] args) {
        int[] arr = {1, 2, 4, 5, 6}; // Example array
        int missingElement = findMissingElement(arr,
arr.length + 1);
        System.out.println("Missing element: " +
missingElement);
    }

    public static int findMissingElement(int[] arr, int n) {
        int sum = n * (n + 1) / 2; // Sum of first n integers
        for (int i = 0; i < arr.length; i++) {
            sum -= arr[i];
        }
        return sum;
    }
}
```

Q5.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class MaxMinRange {
```

```
    public static void main(String[] args) {
        int[] arr = {2, 5, 1, 7, 9, 3, 6, 8}; // Example array
        int min = findMin(arr);
        int max = findMax(arr);
        System.out.println("Minimum value: " + min);
        System.out.println("Maximum value: " + max);
        List<Integer> missing = findMissingInRange(arr, min,
max);
        System.out.println("Values missing in range: " +
missing);
    }

    public static int findMin(int[] arr) {
        int min = Integer.MAX_VALUE;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] < min) {
                min = arr[i];
            }
        }
        return min;
    }
```

```
    public static int findMax(int[] arr) {
        int max = Integer.MIN_VALUE;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] > max) {
                max = arr[i];
            }
        }
        return max;
    }

    public static List<Integer> findMissingInRange(int[] arr,
int min, int max) {
        boolean[] present = new boolean[max - min + 1];
        Arrays.fill(present, false);
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] >= min && arr[i] <= max) {
                present[arr[i] - min] = true;
            }
        }
        List<Integer> missing = new ArrayList<>();
        for (int i = 0; i < present.length; i++) {
            if (!present[i]) {
                missing.add(i + min);
```

```
            }
        }
        return missing;
    }
}
```

Q6.
```
public class FindOddOccurrence {

    public static void main(String[] args) {
        int[] arr = {2, 2, 5, 5, 6, 6, 8, 8, 9}; // Example array
        int oddOccurrence = findOddOccurrence(arr);
        System.out.println("Element that appears odd number of times: " + oddOccurrence);
    }

    public static int findOddOccurrence(int[] arr) {
        int result = 0;
        for (int i = 0; i < arr.length; i++) {
            result ^= arr[i]; // Bitwise XOR operation
        }
        return result;
    }
}
```

Q7. public class FindTwoOddOccurrence {

```
    public static void main(String[] args) {
        int[] arr = {2, 2, 5, 5, 6, 6, 8, 9}; // Example array
        int[] oddOccurrences = findTwoOddOccurrences(arr);
        System.out.println("Elements that appear odd number of times: " + oddOccurrences[0] + " and " + oddOccurrences[1]);
    }

    public static int[] findTwoOddOccurrences(int[] arr) {
        int xorResult = 0;
        for (int i = 0; i < arr.length; i++) {
            xorResult ^= arr[i]; // Bitwise XOR operation
        }

        // Find the rightmost set bit in the XOR result
        int rightmostSetBit = xorResult & -xorResult;

        int oddOccurrence1 = 0;
        int oddOccurrence2 = 0;

        // Divide the elements into two groups based on the rightmost set bit
```

```
        for (int i = 0; i < arr.length; i++) {
            if ((arr[i] & rightmostSetBit) != 0) {
                oddOccurrence1 ^= arr[i];
            } else {
                oddOccurrence2 ^= arr[i];
            }
        }

        int[] oddOccurrences = {oddOccurrence1, oddOccurrence2};
        return oddOccurrences;
    }
}
```

Q8.

```
public class SumOfDistinctElements {

    public static void main(String[] args) {
        int[] arr = {1, 2, 3, 3, 4, 4, 4, 5}; // Example array
        int sum = sumOfDistinctElements(arr);
        System.out.println("Sum of distinct elements: " + sum);
    }
```

```
    public static int sumOfDistinctElements(int[] arr) {
        int sum = arr[0];

        for (int i = 1; i < arr.length; i++) {
            // If the current element is not equal to the previous
element,
            // add it to the sum
            if (arr[i] != arr[i-1]) {
                sum += arr[i];
            }
        }

        return sum;
    }
}
```

Q9.

```
import java.util.Arrays;

public class SumClosestToZero {

    public static void main(String[] args) {
        int[] arr = {-5, 4, -2, 8, 3}; // Example array
        int[] closestSum = sumClosestToZero(arr);
```

```
        System.out.println("Two elements with sum closest to zero: " + Arrays.toString(closestSum));
    }

    public static int[] sumClosestToZero(int[] arr) {
        Arrays.sort(arr); // Sort the array
        int left = 0; // Pointer to leftmost element
        int right = arr.length-1; // Pointer to rightmost element
        int minSum = Integer.MAX_VALUE; // Minimum sum found so far
        int[] closestSum = new int[2]; // Array to hold the two elements with closest sum

        while (left < right) {
            int sum = arr[left] + arr[right];
            if (Math.abs(sum) < Math.abs(minSum)) {
                // Update minimum sum found so far
                minSum = sum;
                closestSum[0] = arr[left];
                closestSum[1] = arr[right];
            }
            if (sum < 0) {
                // If sum is negative, move left pointer to the right
                left++;
```

```
            } else {
                // If sum is non-negative, move right pointer to
the left
                right--;
            }
        }

        return closestSum;
    }
}
```

Q10.

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

public class TwoElementsSumValue {

    public static void main(String[] args) {
        int[] arr = {1, 5, 6, 9, 12}; // Example array
        int value = 11; // Example value
        int[] elements = findTwoElementsSumValue(arr,
value);
        if (elements != null) {
            System.out.println("Elements with sum " + value +
" are " + elements[0] + " and " + elements[1]);
```

```
        } else {
            System.out.println("No elements found with sum "
+ value);
        }
    }

    public static int[] findTwoElementsSumValue(int[] arr,
int value) {
        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < arr.length; i++) {
            int complement = value - arr[i];
            if (map.containsKey(complement)) {
                return new int[] {arr[i], complement};
            }
            map.put(arr[i], i);
        }
        return null;
    }
}
```

Q11.

```
import java.util.*;

public class FindPairXY {
```

```
    public static void main(String[] args) {
        List<Integer> X = Arrays.asList(2, 3, 5, 7, 9); // Example list X
        List<Integer> Y = Arrays.asList(1, 4, 6, 8, 10); // Example list Y
        int value = 11; // Example value

        List<Integer> pair = findPairXY(X, Y, value);

        if (pair != null) {
            System.out.println("Pair found: (" + pair.get(0) + ", " + pair.get(1) + ")");
        } else {
            System.out.println("No pair found with sum " + value);
        }
    }

    public static List<Integer> findPairXY(List<Integer> X, List<Integer> Y, int value) {
        Set<Integer> set = new HashSet<>(X);
        for (int y : Y) {
            if (set.contains(value - y)) {
                return Arrays.asList(value - y, y);
```

```
            }
        }
        return null;
    }
}
```

Q12.

```
import java.util.Arrays;

public class MinDiffPair {
    public static void main(String[] args) {
        int[] arr = {4, 2, 6, 8, 5, 1, 3, 7}; // Example array
        int[] pair = findMinDiffPair(arr);

        System.out.println("Minimum difference pair: (" +
pair[0] + ", " + pair[1] + ")");
    }

    public static int[] findMinDiffPair(int[] arr) {
        Arrays.sort(arr); // Sort the array in ascending order
        int minDiff = Integer.MAX_VALUE;
        int[] minPair = new int[2];

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

```
            int diff = arr[i] - arr[i-1];
            if (diff < minDiff) {
                minDiff = diff;
                minPair[0] = arr[i-1];
                minPair[1] = arr[i];
            }
        }

        return minPair;
    }
}
```

Q13.

```
import java.util.Arrays;

public class MinDiffPair {
    public static void main(String[] args) {
        int[] arr1 = {4, 2, 6, 8, 5, 1, 3, 7};
        int[] arr2 = {9, 12, 15, 18, 10, 11, 13};
        int[] pair = findMinDiffPair(arr1, arr2);

        System.out.println("Minimum difference pair: (" +
pair[0] + ", " + pair[1] + ")");
    }
```

```
public static int[] findMinDiffPair(int[] arr1, int[] arr2) {
    Arrays.sort(arr1);
    Arrays.sort(arr2);

    int minDiff = Integer.MAX_VALUE;
    int[] minPair = new int[2];

    int i = 0, j = 0;
    while (i < arr1.length && j < arr2.length) {
        int diff = Math.abs(arr1[i] - arr2[j]);
        if (diff < minDiff) {
            minDiff = diff;
            minPair[0] = arr1[i];
            minPair[1] = arr2[j];
        }
        if (arr1[i] < arr2[j])
            i++;
        else
            j++;
    }

    return minPair;
```

```
    }
}
```

Q14.

```
import java.util.Arrays;

public class TripletSumZero {
    public static void main(String[] args) {
        int[] arr = {5, -1, -2, 3, -4, -6, 0};
        int[] triplet = findTripletSumZero(arr);

        System.out.println("Triplet whose sum is 0: (" +
triplet[0] + ", " + triplet[1] + ", " + triplet[2] + ")");
    }

    public static int[] findTripletSumZero(int[] arr) {
        Arrays.sort(arr);

        for (int i = 0; i < arr.length - 2; i++) {
            if (i > 0 && arr[i] == arr[i-1])
                continue;

            int j = i + 1;
            int k = arr.length - 1;
```

```
            while (j < k) {
                int sum = arr[i] + arr[j] + arr[k];
                if (sum == 0)
                    return new int[] {arr[i], arr[j], arr[k]};
                else if (sum < 0)
                    j++;
                else
                    k--;
            }
        }

        return new int[0];
    }
}
```

Q15.
```
import java.util.Arrays;

public class TripletSumGivenValue {
    public static void main(String[] args) {
        int[] arr = {5, 4, 6, 8, 1, 2, 3, 7};
        int value = 16;
        int[] triplet = findTripletSumGivenValue(arr, value);
```

```
        if (triplet.length == 0)
            System.out.println("No triplet found with sum " +
value);
        else
            System.out.println("Triplet whose sum is " + value
+ ": (" + triplet[0] + ", " + triplet[1] + ", " + triplet[2] + ")");
    }


    public static int[] findTripletSumGivenValue(int[] arr, int
value) {
        Arrays.sort(arr);


        for (int i = 0; i < arr.length - 2; i++) {
            int j = i + 1;
            int k = arr.length - 1;
            while (j < k) {
                int sum = arr[i] + arr[j] + arr[k];
                if (sum == value)
                    return new int[] {arr[i], arr[j], arr[k]};
                else if (sum < value)
                    j++;
                else
                    k--;
            }
```

```
        }

        return new int[0];
    }
}
```

Q16.

```
import java.util.Arrays;

public class CountTriangles {
    public static void main(String[] args) {
        int[] arr = {4, 6, 3, 7};
        int count = countTriangles(arr);
        System.out.println("Number of triangles that can be formed: " + count);
    }

    public static int countTriangles(int[] arr) {
        Arrays.sort(arr);
        int count = 0;

        for (int i = 0; i < arr.length - 2; i++) {
            int k = i + 2;
            for (int j = i + 1; j < arr.length - 1; j++) {
```

```
            while (k < arr.length && arr[i] + arr[j] > arr[k]) {
                k++;
            }
            count += k - j - 1;
        }
    }

    return count;
  }
}
```

Q17. To identify the second largest element in an unsorted list of n distinct elements with the minimum number of comparisons, we can use the following algorithm:

1. Initialize two variables, `largest` and `secondLargest`, to the first and second elements of the array, respectively.
2. Iterate over the remaining elements of the array, comparing each element to the `largest` and `secondLargest` variables.
3. If the current element is larger than `largest`, update `secondLargest` to be equal to `largest`, and update `largest` to be equal to the current element.
4. If the current element is smaller than `largest` but larger than `secondLargest`, update `secondLargest` to be equal to the current element.
5. After iterating over all elements, the value of `secondLargest` will be the second largest element in the array.

This algorithm requires only n - 1 comparisons, which is the minimum number of comparisons necessary to identify the second largest element in an unsorted list.

Here's the Java code to implement this algorithm:

code:
```
public static int findSecondLargest(int[] arr) {
    int largest = arr[0];
    int secondLargest = arr[1];
```

```
    if (secondLargest > largest) {
        int temp = largest;
        largest = secondLargest;
        secondLargest = temp;
    }

    for (int i = 2; i < arr.length; i++) {
        if (arr[i] > largest) {
            secondLargest = largest;
            largest = arr[i];
        } else if (arr[i] > secondLargest) {
            secondLargest = arr[i];
        }
    }

    return secondLargest;
}
```

Q18.

```
public class Main {
    public static void main(String[] args) {
        int[] nums = {5, 2, 8, 1, 9, 4}; // unsorted array
        Arrays.sort(nums); // sort the array
```

```
        int midIndex = nums.length / 2; // index of the middle
element
        int midElement = nums[midIndex]; // middle element
        System.out.println("Element at index n/2 = " +
midElement);
    }
}
```

Q19.
```
public class Main {
    public static void main(String[] args) {
        int[] nums = {1, 3, 5, 7, 9, 8, 6, 4, 2}; // bitonic list
        int target = 6; // element to find
        int index = findElement(nums, target);
        if (index == -1) {
            System.out.println("Element not found");
        } else {
            System.out.println("Element found at index " +
index);
        }
    }

    public static int findElement(int[] nums, int target) {
        int peakIndex = findPeakIndex(nums);
        int index = binarySearch(nums, target, 0, peakIndex);
```

```
        if (index != -1) {
            return index;
        }
        return binarySearch(nums, target, peakIndex + 1,
nums.length - 1);
    }

    public static int findPeakIndex(int[] nums) {
        int left = 0;
        int right = nums.length - 1;
        while (left < right) {
            int mid = (left + right) / 2;
            if (nums[mid] < nums[mid + 1]) {
                left = mid + 1;
            } else {
                right = mid;
            }
        }
        return left;
    }

    public static int binarySearch(int[] nums, int target, int
left, int right) {
```

```
        while (left <= right) {
            int mid = (left + right) / 2;
            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
        return -1;
    }
}
```