Assignment 3 Solution - Arrays | DSA

Question 1

Given an integer array nums of length n and an integer target, find three integers in nums such that the sum is closest to the target.Return the sum of the three integers. You may assume that each input would have exactly one solution.

Example 1:

Input: nums = [-1,2,1,-4], target = 1

Output**: 2**

Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).

Solution Code :

```
package in.ineuron.pptAssignment03;
import java.util.Arrays;

public class ThreeSumClosest {

	public static void main(String[] args) {
		int[] nums = { -1, 2, 1, -4 };
		int target = 1;

		System.out.println("Three SUM Closest :: " + threeSumClosest(nums, target));
	}

	public static int threeSumClosest(int[] nums, int target) {
		Arrays.sort(nums);
		int closestSum = Integer.MAX_VALUE;
		int curSum = 0;

		for (int i = 0; i < nums.length - 2; i++) {
			int left = i + 1;
			int right = nums.length - 1;

			while (left < right) {
				curSum = nums[i] + nums[left] + nums[right];
				if (curSum == target) {
					return curSum;
				} else if (Math.abs(target - curSum) < Math.abs(target -
				closestSum)) {
					closestSum = curSum;
				}
				if (curSum <= target) {
					left += 1;
					while (nums[left] == nums[left - 1] && left < right) {
						left += 1;
					}
				} else {
```

```
                                    right -= 1;
                                }
                            }
                        }
                        return closestSum;
                }

        }
```

Question 2

Given an array nums of n integers, return an array of all the unique quadruplets [nums[a], nums[b], nums[c], nums[d]] such that:

- 0 <= a, b, c, d < n
- a, b, c, and d are distinct.
- nums[a] + nums[b] + nums[c] + nums[d] == target

You may return the answer in any order.

Example 1:
Input: nums = [1,0,-1,0,-2,2], target = 0
Output**: [[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]**

Solution Code:

```
package in.ineuron.pptAssignment03;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class FourSum {
    public static void main(String[] args) {
        FourSum solution = new FourSum();
        int[] nums = { 1, 0, -1, 0, -2, 2 };
        int target = 0;
        List<List<Integer>> quadruplets = solution.fourSum(nums, target);

        System.out.println(quadruplets);
    }

    public List<List<Integer>> fourSum(int[] nums, int target) {
        int n = nums.length;
        Arrays.sort(nums);
        List<List<Integer>> ans = new ArrayList<>();
        if (n == 0 || n < 3) {
            return ans;
        }
        if (target == -294967296 || target == 294967296) {
            return ans;
```

```
                }
                for (int i = 0; i < n; i++) {
                        for (int j = i + 1; j < n; j++) {
                                int low = j + 1;
                                int high = n - 1;
                                int sum = target - nums[i] - nums[j];
                                while (low < high) {
                                        if (nums[low] + nums[high] == sum) {
                                                List<Integer> temp = new ArrayList<>();
                                                temp.add(nums[i]);
                                                temp.add(nums[j]);
                                                temp.add(nums[low]);
                                                temp.add(nums[high]);
                                                ans.add(temp);
                                                while (low < high && nums[low] == nums[low + 1]) {
                                                        low++;
                                                }
                                                while (low < high && nums[high] == nums[high - 1]) {
                                                        high--;
                                                }
                                                low++;
                                                high--;
                                        } else if (nums[low] + nums[high] < sum) {
                                                low++;
                                        } else {
                                                high--;
                                        }
                                }
                                while (j + 1 < n && nums[j + 1] == nums[j]) {
                                        j++;
                                }
                        }
                        while (i + 1 < n && nums[i + 1] == nums[i]) {
                                i++;
                        }
                }
                return ans;
        }
}
```

Question 3

A permutation of an array of integers is an arrangement of its members into a sequence or linear order.

For example, for arr = [1,2,3], the following are all the permutations of arr: [1,2,3], [1,3,2], [2, 1, 3], [2, 3, 1], [3,1,2], [3,2,1].

The next permutation of an array of integers is the next lexicographically greater permutation of its integer. More formally, if all the permutations of the array are sorted in one container according to their lexicographical order, then the next permutation of that array is the permutation that follows it in the sorted container.

If such an arrangement is not possible, the array must be rearranged as the lowest possible order (i.e., sorted in ascending order).

- For example, the next permutation of arr = [1,2,3] is [1,3,2].
- Similarly, the next permutation of arr = [2,3,1] is [3,1,2].
- While the next permutation of arr = [3,2,1] is [1,2,3] because [3,2,1] does not have a lexicographical larger rearrangement.

Given an array of integers nums, find the next permutation of nums.
The replacement must be in place and use only constant extra memory.

Example 1:
Input: nums = [1,2,3]
Output**: [1,3,2]**

Solution Code:

```
package in.ineuron.pptAssignment03;
public class NextPermutation {
	public static void main(String[] args) {

		int[] nums = { 1, 2, 3 };
		System.out.println(NextPermutation.nextPermutation(nums));

	}

	public static void nextPermutation(int[] nums) {
		int i = nums.length - 2; // i=3-2=>1
		while (i >= 0 && nums[i + 1] <= nums[i]) { // 3<2
			i--;
		}
		if (i >= 0) { // i=1
			int j = nums.length - 1; // j=3-1=>2
			while (nums[j] <= nums[i]) { // 2<=1
				j--;
```

```
                }
                swap(nums, i, j);
            }
            reverse(nums, i + 1); // [1,2,3],1+1
        }

        private static void reverse(int[] nums, int start) {
            int i = start, j = nums.length - 1; // i=1,j=2
            while (i < j) {
                swap(nums, i, j); // [1,2,3],1,2
                i++;
                j--;
            }
        }

        private static void swap(int[] nums, int i, int j) {
            int temp = nums[i]; // t=2
            nums[i] = nums[j]; // n[1]=n[2]=>3
            nums[j] = temp; // n[2]=num[i]=>2

        }
    }
```

Question 4

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with O(log n) runtime complexity.

Example 1:
Input: nums = [1,3,5,6], target = 5
Output**: 2**

Solution Code:

```
package in.ineuron.pptAssignment03;
public class SearchInsertPosition {
	public static void main(String[] args) {
		int[] nums = { 1, 3, 5, 6 };
		int target = 5;
		int index = searchInsert(nums, target);

		System.out.println("Index: " + index);
	}

	public static int searchInsert(int[] nums, int target) {
		int left = 0;
		int right = nums.length - 1;

		while (left <= right) {
			int mid = left + (right - left) / 2;

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

		return left;
	}
}
```

Question 5

You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

Example 1:
Input: digits = [1,2,3]
Output: **[1,2,4]**

Explanation: The array represents the integer 123.
Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].

Solution Code:

```
package in.ineuron.pptAssignment03;
import java.util.Arrays;
public class PlusOne {

	public static void main(String[] args) {
		PlusOne solution = new PlusOne();
		int[] digits = { 4,3,2,1 };
		int[] result = solution.plusOne(digits);
		System.out.println(Arrays.toString(result)); // Output: [1, 2, 4]
	}

	public int[] plusOne(int[] digits) {
		int n = digits.length;

		// Start from the least significant digit
		for (int i = n - 1; i >= 0; i--) {
			// Increment the digit by 1
			digits[i] += 1;

			// If the digit becomes 10, carry the 1 to the next digit
			if (digits[i] == 10) {
				digits[i] = 0;
			} else {
				// No carry, return the updated digits
				return digits;
			}
		}

		// If all digits were 9 and carried to become 0, create a new array with one
		// more digit
```

```
                int[] newDigits = new int[n + 1];
                newDigits[0] = 1;
                return newDigits;
        }
    }
```

Question 6

Given a non-empty array of integers nums, every element appears twice except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:
Input: nums = [2,2,1]
Output: **1**

Solution Code:

```
package in.ineuron.pptAssignment03;
public class SingleNumber {
    public static void main(String[] args) {
        SingleNumber singleNumber = new SingleNumber();
        int[] nums = { 2, 2, 1 };
        int single = singleNumber.findSingleNumber(nums);
        System.out.println("Single Number: " + single);
    }

    public int findSingleNumber(int[] nums) {
        int result = 0;

        // XOR all the elements in the array
        for (int num : nums) {
            result ^= num;
        }

        return result;
    }
}
```

Question 7

You are given an inclusive range [lower, upper] and a sorted unique integer array nums, where all elements are within the inclusive range.

A number x is considered missing if x is in the range [lower, upper] and x is not in nums.

Return the shortest sorted list of ranges that exactly covers all the missing numbers. That is, no element of nums is included in any of the ranges, and each missing number is covered by one of the ranges.

Example 1:
Input: nums = [0,1,3,50,75], lower = 0, upper = 99
Output: **[[2,2],[4,49],[51,74],[76,99]]**

Explanation: The ranges are:
[2,2]
[4,49]
[51,74]
[76,99]

Solution Code:

```
package in.ineuron.pptAssignment03;
import java.util.ArrayList;
import java.util.List;
public class MissingRanges {
	public static void main(String[] args) {
		int[] nums = { 0, 1, 3, 50, 75 };
		int lower = 0;
		int upper = 99;
		List<String> missingRanges = findMissingRanges(nums, lower, upper);
		System.out.println(missingRanges);
	}

	public static List<String> findMissingRanges(int[] nums, int lower, int upper) {
		List<String> result = new ArrayList<>();

		int n = nums.length;
		int prev = lower - 1;

		for (int i = 0; i <= n; i++) {
			int curr = (i == n) ? upper + 1 : nums[i];
			if (curr - prev >= 2) {
				result.add(getRange(prev + 1, curr - 1));
			}
			prev = curr;
```

```
                }

                return result;
        }

        // Helper function to get the range string based on the lower and upper values
        private static String getRange(int lower, int upper) {
                return (lower == upper) ? String.valueOf(lower) : lower + "->" + upper;
        }
}
```

Question 8

Given an array of meeting time intervals where intervals[i] = [starti, endi], determine if a person could attend all meetings.

Example 1:
Input: intervals = [[0,30],[5,10],[15,20]]
Output: **false**

Solution Code:

```
package in.ineuron.pptAssignment03;
import java.util.Arrays;
public class CanAttendMeetings {

        public static void main(String[] args) {
                int[][] nums = { { 0, 30 }, { 5, 10 }, { 15, 20 } };
                System.out.println(canAttendMeetings(nums));
        }

        public static boolean canAttendMeetings(int[][] intervals) {
                // Sort the intervals based on their start time
                Arrays.sort(intervals, (a, b) -> a[0] - b[0]);

                // Check for any overlap
                for (int i = 0; i < intervals.length - 1; i++) {
                // If the end time of the current interval is greater than or equal to the start
                        // time of the next interval, there is an overlap
                        if (intervals[i][1] >= intervals[i + 1][0]) {
                                return false;
                        }
                }
                return true; // No overlap found, person can attend all meetings
        }
}
```