DSA ASSIGNMENT-3(ARRAYS)

Q1. 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).

Soln.

```
class Solution {
    public int threeSumClosest(int[] nums, int target) {
         Arrays.sort(nums);
        int n = nums.length;
        int closest = nums[0] + nums[1] + nums[n - 1];
        for (int i = 0; i < n - 2; i++) {
            int j = i + 1;
            int k = n - 1;
            while (j < k) {
                int sum = nums[i] + nums[j] + nums[k];
                if (sum <= target) {
                    j++;
                } else {
                    k--;
                }
                if (Math.abs(closest - target) > Math.abs(sum -
target)) {
                    closest = sum;
                }
            }
        }
        return closest;
    }
}
```

Q2. 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]]

Soln.

```java
public class FourSum {

    private static List<List<Integer>> fourSum(int[] nums, int
target) {
        // Resultant list
        List<List<Integer>> quadruplets = new ArrayList<>();
        // Base condition
        if (nums == null || nums.length < 4) {
            return quadruplets;
        }
        // Sort the array
        Arrays.sort(nums);
        // Length of the array
        int n = nums.length;
        // Loop for each element in the array
        for (int i = 0; i < n - 3; i++) {
            // Check for skipping duplicates
            if (i > 0 && nums[i] == nums[i - 1]) {
                continue;
            }
            // Reducing problem to 3Sum problem
            for (int j = i + 1; j < n - 2; j++) {
                // Check for skipping duplicates
                if (j != i + 1 && nums[j] == nums[j - 1]) {
                    continue;
                }
                // Left and right pointers
                int k = j + 1;
                int l = n - 1;
                // Reducing to two sum problem
                while (k < l) {
                    int currentSum = nums[i] + nums[j] + nums[k] +
nums[l];
                    if (currentSum < target) {
                        k++;
                    } else if (currentSum > target) {
                        l--;
                    } else {
                        quadruplets.add(Arrays.asList(nums[i],
nums[j], nums[k], nums[l]));
                        k++;
                        l--;
```

```
                        // Check for skipping duplicates
                        while (k < l && nums[k] == nums[k - 1]) {
                            k++;
                        }
                        while (k < l && nums[l] == nums[l + 1]) {
                            l--;
                        }
                    }
                }
            }
        }
        return quadruplets;
    }
}
```

Solution 3:

```
class Solution {
    public void nextPermutation(int[] nums) {
        // Length of the array
        int n = nums.length;
        // Index of the first element that is smaller than
        // the element to its right.
        int index = -1;
        // Loop from right to left
        for (int i = n - 1; i > 0; i--) {
            if (nums[i] > nums[i - 1]) {
                index = i - 1;
                break;
            }
        }
        // Base condition
        if (index == -1) {
            reverse(nums, 0, n - 1);
            return;
        }
        int j = n - 1;
        // Again swap from right to left to find first element
        // that is greater than the above find element
        for (int i = n - 1; i >= index + 1; i--) {
            if (nums[i] > nums[index]) {
                j = i;
```

```
                break;
            }
        }
        // Swap the elements
        swap(nums, index, j);
        // Reverse the elements from index + 1 to the nums.length
        reverse(nums, index + 1, n - 1);
    }

    private static void reverse(int[] nums, int i, int j) {
        while (i < j) {
            swap(nums, i, j);
            i++;
            j--;
        }
    }

    private static void swap(int[] nums, int i, int index) {
        int temp = nums[index];
        nums[index] = nums[i];
        nums[i] = temp;
    }
}
```

Solution 4:

```
class Solution {
    public int searchInsert(int[] nums, int target) {
        int n=nums.length;
        int start=0;
        int end=n-1;
        while(start<=end)
        {
            int mid=start+(end-start)/2;
            if(nums[mid]==target)
            {
                return mid;
            }
            if(target>nums[mid])
            {
                start=mid+1;
            }else{
```

```
                end=mid-1;
            }
        }
        return start;
    }
}
```

Solution 5:

```
class Solution {
    public int[] plusOne(int[] digits) {
         int n = digits.length;
    for(int i=n-1; i>=0; i--) {
        if(digits[i] < 9) {
            digits[i]++; return digits;
        }
        digits[i] = 0;
    }

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

Solution 6:

```
class Solution {
    public int singleNumber(int[] nums) {
       int result=0;
        for(int i=0; i<nums.length; i++) {
            result = result^nums[i];
        }
        return result;
    }
}
```

Solution 7:

```
class Solution {
    public List<String> summaryRanges(int[] nums) {
         ArrayList<String> al=new ArrayList<>();
```

```
        for(int i=0;i<nums.length;i++){
            int start=nums[i];
            while(i+1<nums.length && nums[i]+1==nums[i+1])
                i++;

            if(start!=nums[i]){
                al.add(""+start+"->"+nums[i]);
            }
            else{
                al.add(""+start);
            }
        }
        return al;
    }
}
```

Solution 8:

```
class Solution {
    public boolean canAttendMeetings(int[][] intervals) {
      Arrays.sort(intervals, new Comparator<int[]>() {
        public int compare(int[] i1, int[] i2) {
          return i1[0] - i2[0];
        }
      });
      for (int i = 0; i < intervals.length - 1; i++) {
        if (intervals[i][1] > intervals[i + 1][0])
          return false;
      }
      return true;
    }
  }
```