Assignment 3 Answers - Arrays | DSA

Answer 1)

```
class Solution {
    public int threeSumClosest(int[] nums, int target) {
        Arrays.sort(nums);
        int minDistance = Integer.MAX_VALUE;
        int closestSum = 0;

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

            while(start < end) {
                int sum = nums[i] + nums[start] + nums[end];
                int distance = Math.abs(target - sum);

                if(sum == target) {
                    return sum;
                }

                if(distance < minDistance) {
                    minDistance = distance;
                    closestSum = sum;
                }

                if(sum < target) {
                    start++;
                } else {
                    end--;
                }
            }
        }
        return closestSum;
    }
}
```

Answer 2)

```
class Solution {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        Arrays.sort(nums);
        Set<List<Integer>> set = new HashSet<>();
        for(int i = 0; i < nums.length - 3; i++) {
            for(int j = i + 1; j < nums.length - 2; j++) {
                int k = j + 1;
                int l = nums.length - 1;
                while(k < l) {
                    long sum = (long)nums[i] + (long)nums[j] + (long)nums[k] +
(long)nums[l];
                    if(sum == target){
                        set.add(List.of(nums[i],nums[j],nums[k],nums[l]));
                        k++;
                        l--;
                    }
                    else if(sum < target)
                        k++;
                    else
                        l--;
                }
            }
        }
        return new ArrayList<>(set);

    }
}
```

Answer 3)

```
class Solution {
    public void nextPermutation(int[] nums) {
        int ind1=-1;
        int ind2=-1;
        for(int i=nums.length-2;i>=0;i--){
            if(nums[i]<nums[i+1]){
                ind1=i;
                break;
            }
        }
        if(ind1==-1){
            reverse(nums,0);
        }
        else{
            for(int i=nums.length-1;i>=0;i--){
                if(nums[i]>nums[ind1]){
                    ind2=i;
                    break;
                }
            }
            swap(nums,ind1,ind2);
            reverse(nums,ind1+1);
        }
    }
    void swap(int[] nums,int i,int j){
        int temp=nums[i];
        nums[i]=nums[j];
        nums[j]=temp;
    }
    void reverse(int[] nums,int start){
        int i=start;
        int j=nums.length-1;
        while(i<j){
            swap(nums,i,j);
            i++;
            j--;
        }
    }
}
```

Answer 4)

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

Answer 5)

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

        digits = new int[digits.length + 1];
        digits[0] = 1;
        return digits;
    }
}
```

Answer 6)

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

Answer 7)

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

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

        long start = lower;

        for (int num : nums) {
            if (num < start) {
                continue;
            }

            if (num == start) {
                start++;
                continue;
            }

            missingRanges.add(getRange(start, num - 1));
            start = (long) num + 1;
        }

        if (start <= upper) {
            missingRanges.add(getRange(start, upper));
        }

        return missingRanges;
    }

    private static String getRange(long start, long end) {
        return start == end ? String.valueOf(start) : start + "->" + end;
    }
    public static void main(String[] args) {
        int[] nums = {0, 1, 3, 50, 75};
        int lower = 0;
        int upper = 99;

        List<String> result = findMissingRanges(nums, lower, upper);
        System.out.println(result);
    }
}
```

Answer 8)

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