Assignment 24 Solution - Basics of DSA - Revision Class

1. Roman to Integer

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

Symbol	Value
I	1
V	5
X	10
L	50
C	100
D	500
M	1000

For example, 2 is written as II in Roman numeral, just two ones added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Roman numerals are usually written largest to smallest from left to right. However, the numeral for four is not IIII. Instead, the number four is written as IV. Because the one is before the five we subtract it making four. The same principle applies to the number nine, which is written as IX. There are six instances where subtraction is used:

- I can be placed before V (5) and X (10) to make 4 and 9.
- X can be placed before L (50) and C (100) to make 40 and 90.
- C can be placed before D (500) and M (1000) to make 400 and 900.

Given a roman numeral, convert it to an integer.

Example 1:
Input: s = "III"
Output: 3
Explanation: III = 3.

Example 2:
Input: s = "LVIII"
Output: 58
Explanation: L = 50, V= 5, III = 3.

Constraints:
- 1 <= s.length <= 15
- s contains only the characters ('I', 'V', 'X', 'L', 'C', 'D', 'M').
- It is guaranteed that s is a valid roman numeral in the range [1, 3999].

Solution:

```
package in.ineuron.pptAssignment24;

import java.util.HashMap;

public class RomanToInteger_1 {
	public static int romanToInt(String s) {
		// Create a HASHMAP to store the values of Roman numerals
		HashMap<Character, Integer> map = new HashMap<>();
		map.put('I', 1);
		map.put('V', 5);
		map.put('X', 10);
		map.put('L', 50);
		map.put('C', 100);
		map.put('D', 500);
		map.put('M', 1000);

		int result = 0;
		int prevValue = 0;

		// Traverse the Roman numeral string from right to left
		for (int i = s.length() - 1; i >= 0; i--) {
			char currentChar = s.charAt(i);
			int currentValue = map.get(currentChar);

			// Check if we need to subtract the current value
			if (currentValue < prevValue) {
				result -= currentValue;
			} else {
				result += currentValue;
			}

			// Update the previous value for the next iteration
			prevValue = currentValue;
		}

		return result;
	}

	public static void main(String[] args) {
		String s = "III";
		int result = romanToInt(s);
		System.out.println("Roman numeral: " + s);
		System.out.println("Integer value: " + result);
	}
}
```

2. Longest Substring Without Repeating Characters

Given a string s, find the length of the longest substring without repeating characters.

Example 1:

Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.

Example 2:

Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.

Example 3:

Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints:

- 0 <= s.length <= 50000
- s consists of English letters, digits, symbols and spaces.

Solution:

```
package in.ineuron.pptAssignment24;

import java.util.HashSet;

public class LongestSubstring_2 {
	public static int lengthOfLongestSubstring(String s) {
		int maxLength = 0;
		int left = 0;
		int right = 0;
		HashSet<Character> set = new HashSet<>();

		// Sliding window approach
		while (right < s.length()) {
			char currentChar = s.charAt(right);

	// If the current character is not in the set, add it to the set and expand the window
			if (!set.contains(currentChar)) {
				set.add(currentChar);
				maxLength = Math.max(maxLength, set.size());
				right++;
			} else {
	// If the current character is already in the set, remove the leftmost characterfrom the set
				char leftChar = s.charAt(left);
				set.remove(leftChar);
```

```
                    left++;
                }
            }

            return maxLength;
        }

        public static void main(String[] args) {
            String s1 = "abcabcbb";
            int result1 = lengthOfLongestSubstring(s1);
            System.out.println("Input: " + s1);
            System.out.println("Output: " + result1);

            String s2 = "bbbbb";
            int result2 = lengthOfLongestSubstring(s2);
            System.out.println("Input: " + s2);
            System.out.println("Output: " + result2);

            String s3 = "pwwkew";
            int result3 = lengthOfLongestSubstring(s3);
            System.out.println("Input: " + s3);
            System.out.println("Output: " + result3);
        }
    }
```

3. Majority Element

Given an array nums of size n, return the majority element.
The majority element is the element that appears more than ⌊n / 2⌋ times. You may assume that the majority element always exists in the array.

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

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

Constraints:
- n == nums.length
- 1 <= n <= 5 10^4
- -10^9 <= nums[i] <= 10^9

Solution:

```
package in.ineuron.pptAssignment24;

public class MajorityElement_3 {
	public static int majorityElement(int[] nums) {
		int majority = nums[0];
		int count = 1;

		// Moore's Voting Algorithm
		for (int i = 1; i < nums.length; i++) {
			if (nums[i] == majority) {
				count++;
			} else {
				count--;
				if (count == 0) {
					majority = nums[i];
					count = 1;
				}
			}
		}

		return majority;
	}

	public static void main(String[] args) {
		int[] nums1 = { 3, 2, 3 };
		int result1 = majorityElement(nums1);
		System.out.println("Input: [3, 2, 3]");
```

```
                System.out.println("Output: " + result1);

                int[] nums2 = { 2, 2, 1, 1, 1, 2, 2 };
                int result2 = majorityElement(nums2);
                System.out.println("Input: [2, 2, 1, 1, 1, 2, 2]");
                System.out.println("Output: " + result2);
        }
    }
```

4. Group Anagram

Given an array of strings strs, group the anagrams together. You can return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

Example 1:

Input: strs = ["eat","tea","tan","ate","nat","bat"]
Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

Example 2:

Input: strs = [""]
Output: [[""]]

Example 3:

Input: strs = ["a"]
Output: [["a"]]

Constraints:
- 1 <= strs.length <= 10000
- 0 <= strs[i].length <= 100
- strs[i] consists of lowercase English letters.

Solution:

```
package in.ineuron.pptAssignment24;

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

public class GroupAnagrams_4 {
    public static List<List<String>> groupAnagrams(String[] strs) {
        HashMap<String, List<String>> map = new HashMap<>();

        // Iterate through each string in the input array
        for (String str : strs) {
            // Convert the string to a character array, sort it, and convert it back to a
```

```
                // string
                char[] charArray = str.toCharArray();
                Arrays.sort(charArray);
                String sortedStr = String.valueOf(charArray);

                // Check if the sorted string exists as a key in the map
                // If it does not exist, create a new key-value pair with the sorted string as
                // the key and an empty list as the value
                if (!map.containsKey(sortedStr)) {
                        map.put(sortedStr, new ArrayList<>());
                }

                // Add the original string to the list corresponding to the sorted string key
                map.get(sortedStr).add(str);
        }

        // Convert the values of the map to a list and return it
        return new ArrayList<>(map.values());
    }

    public static void main(String[] args) {
        String[] strs1 = { "eat", "tea", "tan", "ate", "nat", "bat" };
        List<List<String>> result1 = groupAnagrams(strs1);
        System.out.println("Input: " + Arrays.toString(strs1));
        System.out.println("Output: " + result1);

        String[] strs2 = { "" };
        List<List<String>> result2 = groupAnagrams(strs2);
        System.out.println("Input: " + Arrays.toString(strs2));
        System.out.println("Output: " + result2);

        String[] strs3 = { "a" };
        List<List<String>> result3 = groupAnagrams(strs3);
        System.out.println("Input: " + Arrays.toString(strs3));
        System.out.println("Output: " + result3);
    }
}
```

5. Ugly Numbers

An ugly number is a positive integer whose prime factors are limited to 2, 3, and 5.
Given an integer n, return the nth ugly number.

Example 1:
Input: n = 10
Output: 12
Explanation: [1, 2, 3, 4, 5, 6, 8, 9, 10, 12] is the sequence of the first 10 ugly numbers.

Example 2:
Input: n = 1
Output: 1
Explanation: 1 has no prime factors, therefore all of its prime factors are limited to 2, 3, and 5.

Constraints:
- 1 <= n <= 1690

Solution:

```
package in.ineuron.pptAssignment24;

public class UglyNumbers_5 {
	public static int nthUglyNumber(int n) {
		int[] uglyNumbers = new int[n];
		uglyNumbers[0] = 1;

		int index2 = 0; // Index for the next number to multiply by 2
		int index3 = 0; // Index for the next number to multiply by 3
		int index5 = 0; // Index for the next number to multiply by 5

		int factor2 = 2; // Next multiple of 2
		int factor3 = 3; // Next multiple of 3
		int factor5 = 5; // Next multiple of 5

		// Generate the ugly numbers up to the nth number
		for (int i = 1; i < n; i++) {
			// Find the minimum of the three factors
			int min = Math.min(factor2, Math.min(factor3, factor5));

			// Store the minimum ugly number
			uglyNumbers[i] = min;

			// Update the indices and factors accordingly
			if (min == factor2) {
				index2++;
				factor2 = uglyNumbers[index2] * 2;
			}
			if (min == factor3) {
				index3++;
```

```
                factor3 = uglyNumbers[index3] * 3;
            }
            if (min == factor5) {
                index5++;
                factor5 = uglyNumbers[index5] * 5;
            }
        }

        return uglyNumbers[n - 1];
    }

    public static void main(String[] args) {
        int n1 = 10;
        int result1 = nthUglyNumber(n1);
        System.out.println("Input: " + n1);
        System.out.println("Output: " + result1);

        int n2 = 1;
        int result2 = nthUglyNumber(n2);
        System.out.println("Input: " + n2);
        System.out.println("Output: " + result2);
    }
}
```

6. Top K Frequent Words

Given an array of strings words and an integer k, return the k most frequent strings.
Return the answer sorted by the frequency from highest to lowest. Sort the words with the same frequency by their lexicographical order.

Example 1:
Input: words = ["i","love","leetcode","i","love","coding"], k = 2
Output: ["i","love"]
Explanation: "i" and "love" are the two most frequent words.
Note that "i" comes before "love" due to a lower alphabetical order.

Example 2:
Input: words = ["the","day","is","sunny","the","the","the","sunny","is","is"], k = 4
Output: ["the","is","sunny","day"]
Explanation: "the", "is", "sunny" and "day" are the four most frequent words, with the number of occurrence being 4, 3, 2 and 1 respectively.

Constraints:
- 1 <= words.length <= 500
- 1 <= words[i].length <= 10
- words[i] consists of lowercase English letters.
- k is in the range [1, The number of unique words[i]]

Solution:

```
package in.ineuron.pptAssignment24;
import java.util.*;
public class TopKFrequentWords_6 {
	public static List<String> topKFrequent(String[] words, int k) {
		// Count the frequency of each word using a hashmap
		HashMap<String, Integer> frequencyMap = new HashMap<>();
		for (String word : words) {
			frequencyMap.put(word, frequencyMap.getOrDefault(word, 0) + 1);
		}

// Create a priority queue to store the words based on frequency and lexicographical order
		PriorityQueue<String> pq = new PriorityQueue<>(new
		WordComparator(frequencyMap));

		// Iterate through the hashmap and add the words to the priority queue
		for (String word : frequencyMap.keySet()) {
			pq.offer(word);
			if (pq.size() > k) {
				pq.poll();
				// Remove the word with the lowest frequency from the queue
			}
		}
```

```
        // Create a list to store the k most frequent words in the correct order
        List<String> result = new ArrayList<>();
        while (!pq.isEmpty()) {
            result.add(pq.poll());
        }

        // Reverse the list to get the words in descending frequency order
        Collections.reverse(result);

        return result;
    }

// Custom comparator to compare words based on frequency and lexicographical order
    static class WordComparator implements Comparator<String> {
        private final HashMap<String, Integer> frequencyMap;

        public WordComparator(HashMap<String, Integer> frequencyMap) {
            this.frequencyMap = frequencyMap;
        }

        @Override
        public int compare(String word1, String word2) {
            int freq1 = frequencyMap.get(word1);
            int freq2 = frequencyMap.get(word2);

            if (freq1 == freq2) {
                return word1.compareTo(word2); // Lexicographical order
            } else {
                return freq1 - freq2; // Descending frequency order
            }
        }
    }

    public static void main(String[] args) {
        String[] words1 = { "i", "love", "leetcode", "i", "love", "coding" };
        int k1 = 2;
        List<String> result1 = topKFrequent(words1, k1);
        System.out.println("Input: " + Arrays.toString(words1) + ", k = " + k1);
        System.out.println("Output: " + result1);

    String[] words2 = { "the", "day", "is", "sunny", "the", "the", "the", "sunny", "is", "is" };
        int k2 = 4;
        List<String> result2 = topKFrequent(words2, k2);
        System.out.println("Input: " + Arrays.toString(words2) + ", k = " + k2);
        System.out.println("Output: " + result2);
    }
}
```

7. Sliding Window Maximum

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.
Return the max sliding window.

Example 1:

Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
Output: [3,3,5,5,6,7]
Explanation:

```
Window position                Max
---------------               -----
[1  3  -1] -3  5  3  6 7       3
 1 [3  -1  -3] 5  3  6 7       3
 1  3 [-1  -3  5] 3  6 7       5
 1  3  -1 [-3  5  3] 6 7       5
 1  3  -1  -3 [5  3  6]7       6
 1  3  -1  -3  5 [3  6  7]      7
```

Example 2:

Input: nums = [1], k = 1
Output: [1]

Constraints:

- 1 <= nums.length <= 100000
- -10000 <= nums[i] <= 10000
- 1 <= k <= nums.length

Solution:

```
package in.ineuron.pptAssignment24;

import java.util.ArrayDeque;
import java.util.Arrays;
import java.util.Deque;

public class SlidingWindowMaximum_7 {
	public static int[] maxSlidingWindow(int[] nums, int k) {
		int n = nums.length;
		if (n == 0)
			return new int[0];

		int[] result = new int[n - k + 1];
		Deque<Integer> deque = new ArrayDeque<>();

		// Process the first k elements separately to initialize the deque
		for (int i = 0; i < k; i++) {
			while (!deque.isEmpty() && nums[i] >= nums[deque.peekLast()]) {
				deque.removeLast();
```

```
                }
                deque.addLast(i);
            }

            // Process the remaining elements
            for (int i = k; i < n; i++) {
                result[i - k] = nums[deque.peekFirst()];

                while (!deque.isEmpty() && deque.peekFirst() <= i - k) {
                    deque.removeFirst();
                }

                while (!deque.isEmpty() && nums[i] >= nums[deque.peekLast()]) {
                    deque.removeLast();
                }

                deque.addLast(i);
            }

            result[n - k] = nums[deque.peekFirst()];

            return result;
        }

        public static void main(String[] args) {
            int[] nums1 = { 1, 3, -1, -3, 5, 3, 6, 7 };
            int k1 = 3;
            int[] result1 = maxSlidingWindow(nums1, k1);
            System.out.println("Input: " + Arrays.toString(nums1) + ", k = " + k1);
            System.out.println("Output: " + Arrays.toString(result1));

            int[] nums2 = { 1 };
            int k2 = 1;
            int[] result2 = maxSlidingWindow(nums2, k2);
            System.out.println("Input: " + Arrays.toString(nums2) + ", k = " + k2);
            System.out.println("Output: " + Arrays.toString(result2));
        }
    }
```

💡 8. Find K Closest Elements

Given a sorted integer array arr, two integers k and x, return the k closest integers to x in the array. The result should also be sorted in ascending order.
An integer a is closer to x than an integer b if:
- |a - x| < |b - x|, or
- |a - x| == |b - x| and a < b

Example 1:
Input: arr = [1,2,3,4,5], k = 4, x = 3
Output: [1,2,3,4]

Example 2:
Input: arr = [1,2,3,4,5], k = 4, x = -1
Output: [1,2,3,4]

Constraints:
- 1 <= k <= arr.length
- 1 <= arr.length <= 10000
- arr is sorted in ascending order.
- -10000 <= arr[i], x <= 10000

Solution:

```
package in.ineuron.pptAssignment24;

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

public class KClosestElements_8 {
	public static List<Integer> findClosestElements(int[] arr, int k, int x) {
		int left = 0;
		int right = arr.length - 1;

		// Binary search to find the closest element to x
		while (right - left >= k) {
			int mid = left + (right - left) / 2;

			// Compare the distances of the elements at mid and mid + k to x
			if (Math.abs(arr[mid] - x) > Math.abs(arr[mid + k] - x)) {
				left = mid + 1;
			} else {
				right = mid;
			}
		}

		// Create a list to store the k closest elements
		List<Integer> result = new ArrayList<>();
		for (int i = left; i < left + k; i++) {
```

```
                result.add(arr[i]);
            }

            return result;
        }

        public static void main(String[] args) {
            int[] arr1 = { 1, 2, 3, 4, 5 };
            int k1 = 4;
            int x1 = 3;
            List<Integer> result1 = findClosestElements(arr1, k1, x1);
        System.out.println("Input: arr = " + arrayToString(arr1) + ", k = " + k1 + ", x = " + x1);
            System.out.println("Output: " + result1);

            int[] arr2 = { 1, 2, 3, 4, 5 };
            int k2 = 4;
            int x2 = -1;
            List<Integer> result2 = findClosestElements(arr2, k2, x2);
        System.out.println("Input: arr = " + arrayToString(arr2) + ", k = " + k2 + ", x = " + x2);
            System.out.println("Output: " + result2);
        }

        private static String arrayToString(int[] arr) {
            StringBuilder sb = new StringBuilder();
            sb.append("[");
            for (int i = 0; i < arr.length; i++) {
                sb.append(arr[i]);
                if (i < arr.length - 1) {
                    sb.append(", ");
                }
            }
            sb.append("]");
            return sb.toString();
        }
    }
```