Q1) Sorting Elements of an Array by Frequency

Given an array A[] of integers, sort the array according to frequency of elements. That is elements that have higher frequency come first. If frequencies of two elements are same, then

smaller number comes first

import java.util.\*;

public class FrequencySort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int t = scanner.nextInt(); // number of test cases

while (t-- > 0) {

int n = scanner.nextInt(); // size of array

int[] arr = new int[n];

for (int i = 0; i < n; i++) {

arr[i] = scanner.nextInt(); // elements of the array

}

// Create a frequency map to store the count of each element

Map<Integer, Integer> frequencyMap = new HashMap<>();

for (int num : arr) {

frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);

}

// Sort the array based on frequency and element value

Arrays.sort(arr, (a, b) -> {

int freqCompare = frequencyMap.get(a).compareTo(frequencyMap.get(b));

if (freqCompare != 0) {

return -freqCompare; // Sort in descending order of frequency

} else {

return Integer.compare(a, b); // Sort in ascending order of element value

}

});

// Print the sorted array

for (int num : arr) {

System.out.print(num + " ");

}

System.out.println();

}

}

}

Q2) Longest consecutive subsequence

Given an array of positive integers. Find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the consecutive numbers can be in any order

import java.util.HashSet;

public class LongestConsecutiveSubsequence {

public static int findLongestConsecutiveSubsequence(int[] nums) {

HashSet<Integer> set = new HashSet<>();

int maxLength = 0;

// Add all elements to the set

for (int num : nums) {

set.add(num);

}

// Iterate through the array

for (int num : nums) {

// Check if the current number is the start of a subsequence

if (!set.contains(num - 1)) {

int currentNum = num;

int currentLength = 1;

// Find the length of the consecutive subsequence

while (set.contains(currentNum + 1)) {

currentNum++;

currentLength++;

}

// Update the maximum length if necessary

maxLength = Math.max(maxLength, currentLength);

}

}

return maxLength;

}

public static void main(String[] args) {

int[] nums = { 1, 9, 3, 10, 4, 20, 2 };

int longestLength = findLongestConsecutiveSubsequence(nums);

System.out.println("Length of the longest consecutive subsequence: " + longestLength);

}

}

Q3) Given an integer array coins[ ] of size N representing different denominations of currency

and an integer sum, find the number of ways you can make sum by using different combinations

from coins[ ].

public class CoinChange {

public static int countWays(int[] coins, int sum) {

int[] dp = new int[sum + 1];

dp[0] = 1;

for (int coin : coins) {

for (int i = coin; i <= sum; i++) {

dp[i] += dp[i - coin];

}

}

return dp[sum];

}

public static void main(String[] args) {

int[] coins = {1, 2, 5};

int sum = 10; // Example sum

int ways = countWays(coins, sum);

System.out.println("Number of ways to make " + sum + " using different combinations from coins: " + ways);

}

}