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.

```
package Java_Programs;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Map.Entry;

public class SortElementByFrequency {
      private static void sortArrayElementsByFrequency(int[]
inputArray)
          {
              //Create LinkedHashMap with elements as keys and
their occurrences as values
              //Remember LinkedHashMap maintains insertion order
of elements

              Map<Integer, Integer> elementCountMap = new
LinkedHashMap<>();

              //Check presence of each element in elementCountMap

              for (int i = 0; i < inputArray.length; i++)
              {
                  if (elementCountMap.containsKey(inputArray[i]))
                  {
                      //If element is present in elementCountMap,
increment its value by 1

                      elementCountMap.put(inputArray[i],
elementCountMap.get(inputArray[i])+1);
                  }
                  else
                  {
                      //If element is not present, insert this
element with 1 as its value

                      elementCountMap.put(inputArray[i], 1);
                  }
              }
```

```
            //Construct an ArrayList holding all Entry objects
of elementCountMap

            ArrayList<Entry<Integer, Integer>> listOfEntry =
new ArrayList<>(elementCountMap.entrySet());

            //Sort listOfEntry based on values

            Collections.sort(listOfEntry, new
Comparator<Entry<Integer, Integer>>()
                                                {
                                                    @Override
                                                    public int
compare(Entry<Integer, Integer> o1, Entry<Integer, Integer> o2)
                                                    {
                                                        return
o2.getValue().compareTo(o1.getValue());
                                                    }
                                                }
            );

            //Print sorted array elements in descending order
of their frequency

            System.out.println("Entered Array :
"+Arrays.toString(inputArray));

            System.out.println("Sorted Array Element using
Their Frequency : ");

            System.out.print("[ ");

            for (Entry<Integer, Integer> entry : listOfEntry)
            {
                int frequency = entry.getValue();

                while (frequency >= 1)
                {
                    System.out.print(entry.getKey()+" ");

                    frequency--;
                }
            }

            System.out.print("]");
        }
```

```
    public static void main(String[] args)
    {
        sortArrayElementsByFrequency(new int[] {8, 2, 3, 4,
8, 10, 10, 10, 2, 8, 2, 4, 5, 2, 9, 3, 10});
    }

}
```

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.

```
package Java_Programs;

import java.util.HashSet;

public class FindLongestSubSequent {

    static int findLongestConseqSubseq(int arr[], int n)
    {

        HashSet S = new HashSet();
        int ans = 0;


        // Hash all the array elements
        for (int i = 0; i < n; ++i)
            S.add(arr[i]);

        // check each possible sequence from the start
        // then update optimal length
        for (int i = 0; i < n; ++i)
        {
            // if current element is the starting
            // element of a sequence
            if (!S.contains(arr[i] - 1))
            {
                // Then check for next elements
                // in the sequence
                int j = arr[i];
                while (S.contains(j))
                    j++;

                // update  optimal length if this
                // length is more
                if (ans < j - arr[i])
```

```
                    ans = j - arr[i];
                }
            }
            return ans;
        }


        // Driver Code
        public static void main(String args[])

        {

            int arr[] = { 1, 8, 9, 15, 3, 18, 2, 35 };
            int n = arr.length;
            System.out.println(
                "Length of the Longest consecutive subsequence is "+
findLongestConseqSubseq(arr, n));
        }

}
```

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[ ].

```
package Java_Programs;

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

public class CombinationFromCoins {

     public static final int[] DENO = {1,2,5,10};
    public static final int AMOUNT = 35;
    public static int count = 0;

    public static void change(int amount) {
      change(amount, new ArrayList<>(),0);
    }

    private static void change(int rem, List<Integer> coins, int
pos) {
      if (rem == 0) {
```

```
            count++;
            System.out.println(count+")"+coins);
            return;
          }

          while(pos<DENO.length){
            if (rem >= DENO[pos]) {
              coins.add(DENO[pos]);
              change(rem - DENO[pos], coins,pos);
              coins.remove(coins.size() - 1);  //backtrack
            }
            pos++;
          }
        }

        public static void main(String[] args) {
              change(AMOUNT);
        }
}
```