Q 1. 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.

ANS :

```
import java.util.HashSet;
import java.util.Set;

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

        while (right < s.length()) {
            if (!set.contains(s.charAt(right))) {
                set.add(s.charAt(right));
                maxLength = Math.max(maxLength, right - left + 1);
                right++;
            } else {
                set.remove(s.charAt(left));
                left++;
            }
        }

        return maxLength;
    }

    public static void main(String[] args) {
        String s1 = "abcabcbb";
        System.out.println(Substring(s1));

        String s2 = "bbbbb";
        System.out.println(Substring(s2));
    }
}
```

Q 2. Given an integer array nums of unique elements, return all possible subsets (the power set).
The solution set must not contain duplicate subsets. Return the solution in any order. -
---
Example 1:
Input: nums = [1,2,3]
Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
Example 2:
Input: nums = [0] Output: [[],[0]]

ANS :

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

public class Subsets {
   public static List<List<Integer>> sets(int[] nums) {
      List<List<Integer>> sets = new ArrayList<>();
      backtrack(nums, 0, new ArrayList<>(), sets);
      return sets;
   }

   private static void backtrack(int[] nums, int start, List<Integer> currentSubset,
List<List<Integer>> sets) {
      sets.add(new ArrayList<>(currentSubset));

      for (int i = start; i < nums.length; i++) {
         currentSubset.add(nums[i]);
         backtrack(nums, i + 1, currentSubset, sets);
         currentSubset.remove(currentSubset.size() - 1);
      }
   }

   public static void main(String[] args) {
      int[] nums1 = {1, 2, 3};
      List<List<Integer>> subsets1 = sets(nums1);
      System.out.println(subsets1);

      int[] nums2 = {0};
      List<List<Integer>> subsets2 = sets(nums2);
      System.out.println("\n"+subsets2);
   }
}
```

Q 3. Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.
An input string is valid if:
Open brackets must be closed by the same type of brackets. Open brackets must be closed in the correct order.

Example 1:
Input: s = "()"
Output: true
Example 2:
Input: s = "()[]{}"
Output: true
Example 3:
Input: s = "(]"
Output: false

ANS :

```
import java.util.Stack;

public class ValidParentheses {
   public static boolean isValid(String s) {
      Stack<Character> stack = new Stack<>();

      for (char ch : s.toCharArray()) {
         if (ch == '(' || ch == '{' || ch == '[') {
            stack.push(ch);
         } else {
            if (stack.isEmpty()) {
               return false;
            }
            char top = stack.pop();
            if ((ch == ')' && top != '(') || (ch == '}' && top != '{') || (ch == ']' && top != '[')) {
               return false;
            }
         }
      }
      return stack.isEmpty();
   }
   public static void main(String[] args) {
      String s1 = "()";
      System.out.println(isValid(s1));
      String s2 = "()[]{}";
      System.out.println(isValid(s2));
      String s3 = "(]";
      System.out.println(isValid(s3));
   }
}
```

Q 4. Given a list/array of intiger containing 0's and 1's
write a program to fetch the starting index of longest subsequent 1's and length of longest subsequent 1's

---

Example 1:
Input: nums = [1,0,1,1,0,0,1,0,1,1,1,0,1,0,0,1]
Output: 8,3
Explanation: in index 8 we get 3 continues 1's
Example 2:
Input: nums = [1,0,1,0,1,1,0]
Output: 4,2
Explanation: in index 4 we get 2 continues 1's

ANS :

```
public class LSOnes {
   public static void main(String[] args) {
      int[] nums1 = {1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1};
      int[] nums2 = {1, 0, 1, 0, 1, 1, 0};

      int[] result1 = findLSOnes(nums1);
      System.out.println( result1[0] + " " + result1[1]);

      int[] result2 = findLSOnes(nums2);
      System.out.println( result2[0] + " " + result2[1]);
   }

   public static int[] findLSOnes(int[] nums) {
      int maxIndex = -1;
      int maxLength = 0;

      int startIndex = -1;
      int length = 0;

      for (int i = 0; i < nums.length; i++) {
         if (nums[i] == 1) {
            if (startIndex == -1) {
               startIndex = i;
            }
            length++;
         } else {
            if (length > maxLength) {
               maxIndex = startIndex;
               maxLength = length;
            }
            startIndex = -1;
            length = 0;
         }
      }
```

```
        if (length > maxLength) {
            maxIndex = startIndex;
            maxLength = length;
        }

        return new int[]{maxIndex, maxLength};
    }
}
```