Leet Code problems – 25/09/2024

Q1.

```
public class Solution {

    public int[] searchRange(int[] nums, int target) {
        int[] result = new int[2];
        result[0] = findFirst(nums, target);
        result[1] = findLast(nums, target);
        return result;
    }

    private int findFirst(int[] nums, int target) {
        int index = -1;
        int low = 0, high = nums.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (nums[mid] >= target) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }

            if (nums[mid] == target) {
                index = mid;
            }
        }
```

```
        return index;
    }

    private int findLast(int[] nums, int target) {
        int index = -1;
        int low = 0, high = nums.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (nums[mid] <= target) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }

            if (nums[mid] == target) {
                index = mid;
            }
        }

        return index;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
    }
}
```

Q2.

```
public class Solution {

    public int search(int[] nums, int target) {
        int low = 0, high = nums.length - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;

            if (nums[mid] == target) {
                return mid;
            }

            if (nums[low] <= nums[mid]) {
                if (nums[low] <= target && target < nums[mid]) {
                    high = mid - 1;
                } else {
                    low = mid + 1;
                }
            }
            else {
                if (nums[mid] < target && target <= nums[high]) {
                    low = mid + 1;
                } else {
                    high = mid - 1;
                }
            }
        }
```

```
        return -1;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

    }
}
```

Q3.

```
import java.util.Arrays;

public class Solution {

    public void nextPermutation(int[] nums) {
        int n = nums.length;
        int i = n - 2;

        while (i >= 0 && nums[i] >= nums[i + 1]) {
            i--;
        }

        if (i >= 0) {
            int j = n - 1;
            while (nums[j] <= nums[i]) {
                j--;
            }
            swap(nums, i, j);
```

```
        }

        reverse(nums, i + 1, n - 1);
    }

    private void swap(int[] nums, int i, int j) {
        int temp = nums[i];
        nums[i] = nums[j];
        nums[j] = temp;
    }

    private void reverse(int[] nums, int start, int end) {
        while (start < end) {
            swap(nums, start, end);
            start++;
            end--;
        }
    }

    public static void main(String[] args) {
        Solution solution = new Solution();

    }
}
```

Q4.

```
import java.util.Stack;

public class Solution {
```

```
public int calculate(String s) {
    Stack<Integer> stack = new Stack<>();
    int result = 0;
    int sign = 1;
    int n = s.length();
    int i = 0;

    while (i < n) {
        char c = s.charAt(i);

        if (Character.isDigit(c)) {
            int num = 0;
            while (i < n && Character.isDigit(s.charAt(i))) {
                num = num * 10 + (s.charAt(i) - '0');
                i++;
            }
            result += sign * num;
            continue;
        } else if (c == '+') {
            sign = 1;
        } else if (c == '-') {
            sign = -1;
        } else if (c == '(') {
            stack.push(result);
            stack.push(sign);
            result = 0;
            sign = 1;
        } else if (c == ')') {
```

```
            result = stack.pop() * result + stack.pop();
        }
        i++;
    }

    return result;
  }

  public static void main(String[] args) {
    Solution calculator = new Solution();

  }
}
```

Q5.

```
public class Solution {
  public int searchInsert(int[] nums, int target) {
    int low = 0;
    int high = nums.length - 1;

    while (low <= high) {
      int mid = low + (high - low) / 2;

      if (nums[mid] == target) {
        return mid;
      } else if (nums[mid] < target) {
        low = mid + 1;
      } else {
        high = mid - 1;
```

```
            }
        }

        return low;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
    }
}
```

Q6.

```
import java.util.*;

public class Solution {
    public List<Integer> findSubstring(String s, String[] words) {
        List<Integer> result = new ArrayList<>();
        if (s == null || words == null || words.length == 0 || s.length() == 0) {
            return result;
        }

        int wordLength = words[0].length();
        int numWords = words.length;
        int totalLength = wordLength * numWords;

        if (s.length() < totalLength) {
            return result;
        }
```

```
        Map<String, Integer> wordCount = new HashMap<>();
        for (String word : words) {
            wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);
        }

        for (int i = 0; i <= s.length() - totalLength; i++) {
            String currentSubstring = s.substring(i, i + totalLength);
            if (isValid(currentSubstring, wordCount, wordLength)) {
                result.add(i);
            }
        }

        return result;
    }

    private boolean isValid(String substring, Map<String, Integer> wordCount, int wordLength)
{
        Map<String, Integer> seenWords = new HashMap<>();

        for (int j = 0; j < substring.length(); j += wordLength) {
            String word = substring.substring(j, j + wordLength);

            if (!wordCount.containsKey(word)) {
                return false;
            }

            seenWords.put(word, seenWords.getOrDefault(word, 0) + 1);

            if (seenWords.get(word) > wordCount.get(word)) {
                return false;
```

```
            }
        }

        return true;
    }

    public static void main(String[] args) {
        Solution solution = new Solution();
    }
}
```