1. Easy 1

Example 1

Input: s=" fly me to the moon"

Output:=4

Explanation: The last word is " moon"

Answers:

```
class Solution {
    public int lengthOfLastWord(String s) {
        int i = s.length( ) - 1;
        while (i >= 0 && s.charAt(i) == ' ') {
            --i;
        }
        int j = i;
        while (j >= 0 && s.charAt(j) != ' ') {
            --j;
        }
        return i - j;
    }
}
```

2. Easy 2

Example 2:

Input: nums = [1,3]

Output: [3,1]

Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.

Ans:

```
class Solution {
    public List<List<Integer>> generate(int numRows) {
        List<List<Integer>> f = new ArrayList<>( );
        f.add(List.of(1));
        for (int i = 0; i < numRows - 1; ++i) {
            List<Integer> g = new ArrayList<>( );
            g.add(1);
            for (int j = 0; j < f.get(i).size( ) - 1; ++j) {
                g.add(f.get(i).get( j ) + f.get(i).get( j + 1));
            }
            g.add(1);
            f.add(g);
        }
            return f;
    }
}
```

Easy 3:

Example1

Input: numRows = 5

Output: [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

Ans:

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}

public class LowestCommonAncestor {

    public static TreeNode
lowestCommonAncestor(TreeNode root, int p, int q) {
        if (root == null) {
            return null;
```

```
    }

    if (p < root.val && q < root.val) {
        return lowestCommonAncestor(root.left, p, q);
    } else if (p > root.val && q > root.val) {
        return lowestCommonAncestor(root.right, p, q);
    } else {
        return root;
    }
}

public static void main(String[] args) {
    // Create the tree
    TreeNode root = new TreeNode(6);
    root.left = new TreeNode(2);
    root.right = new TreeNode(8);
    root.left.left = new TreeNode(0);
    root.left.right = new TreeNode(4);
    root.right.left = new TreeNode(7);
    root.right.right = new TreeNode(9);
    root.left.right.left = new TreeNode(3);
    root.left.right.right = new TreeNode(5);

    int p = 2;
    int q = 8;
```

```
        TreeNode lca = lowestCommonAncestor(root, p, q);

        System.out.println("The Lowest Common Ancestor of
nodes " + p + " and " + q + " is: " + lca.val);
    }
}
```

Medium:I

Example:I

Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8

Output: 6

Explanation: The LCA of nodes 2 and 8 is 6.

Ans:

```
 class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}

public class LowestCommonAncestor {

    public static TreeNode
lowestCommonAncestor(TreeNode root, int p, int q) {
        if (root == null) {
            return null;
        }

        if (p < root.val && q < root.val) {
```

```
            return lowestCommonAncestor(root.left, p, q);
        } else if (p > root.val && q > root.val) {
            return lowestCommonAncestor(root.right, p, q);
        } else {
            return root;
        }
    }

    public static void main(String[] args) {
        // Create the tree
        TreeNode root = new TreeNode(6);
        root.left = new TreeNode(2);
        root.right = new TreeNode(8);
        root.left.left = new TreeNode(0);
        root.left.right = new TreeNode(4);
        root.right.left = new TreeNode(7);
        root.right.right = new TreeNode(9);
        root.left.right.left = new TreeNode(3);
        root.left.right.right = new TreeNode(5);
```

Medium :2

Example:1

Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times.

Example 1:

Input: nums = [3,2,3]

Output: [3]

Ans:

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

public class MajorityElements {

    public static List<Integer> findMajorityElements(int[]
nums) {
        List<Integer> result = new ArrayList<>( );

        if (nums == null || nums.length == 0) {
            return result;
        }

        int n = nums.length;
```

```
        int targetCount = n / 3;

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

        for (int num : nums) {
            countMap.put(num, countMap.getOrDefault(num,
0) + 1);
        }

        for (Map.Entry<Integer, Integer> entry :
countMap.entrySet( )) {
            if (entry.getValue( ) > targetCount) {
                result.add(entry.getKey( ));
            }
        }

        return result;
    }

    public static void main(String[] args) {
        int[] nums = {3, 2, 3};
        List<Integer> majorityElements =
findMajorityElements(nums);
```

```
            System.out.println("Elements appearing more than ⌊
n/3 ⌋ times: " + majorityElements);
        }
}
```

Medium:3

m == matrix.length

n == matrix[i].length

1 <= m, n <= 300

matrix[i][j] is '0' or '1'.

Example:3

Input: matrix = [["0","1"],["1","0"]]

Output: 1

Ans:

```
 public class MaxSquareSubmatrix {

    public static int maximalSquare(char[][] matrix) {
        if (matrix == null || matrix.length == 0 ||
matrix[0].length == 0) {
            return 0;
        }

        int m = matrix.length;
        int n = matrix[0].length;
        int[][] dp = new int[m + 1][n + 1];
        int maxSquareSize = 0;

        for (int i = 1; i <= m; i++) {
            for (int j = 1; j <= n; j++) {
```

```
                if (matrix[i - 1][j - 1] == '1') {
                    dp[i][j] = Math.min(Math.min(dp[i - 1][j],
dp[i][j - 1]), dp[i - 1][j - 1]) + 1;
                    maxSquareSize = Math.max(maxSquareSize,
dp[i][j]);
                }
            }
        }

        return maxSquareSize * maxSquareSize;
    }

    public static void main(String[] args) {
        char[][] matrix = {
                {'0', '1'},
                {'1', '0'}
        };

        int result = maximalSquare(matrix);

        System.out.println("Maximum size of square
submatrix with all 1s: " + result);
    }
}
```

Hard:1

Example:2

1 <= nums.length <= 105

-104 <= nums[i] <= 104

1 <= k <= nums.length

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
```

Ans:

```
import java.util.ArrayDeque;
import java.util.Deque;

public class SlidingWindowMaximum {

    public static int[] maxSlidingWindow(int[] nums, int k) {
        if (nums == null || nums.length == 0) {
            return new int[0];
```

```
}

int n = nums.length;
int[] result = new int[n - k + 1];
int resultIndex = 0;

Deque<Integer> deque = new ArrayDeque<>( );

        for (int i = 0; i < n; i++) {
            // Remove elements outside the window
            while (!deque.isEmpty( ) && deque.peek( ) < i - k +
1) {
                deque.poll( );
            }

            // Remove elements smaller than the current
element from the back
            while (!deque.isEmpty( ) &&
nums[deque.peekLast( )] < nums[i] ) {
                deque.pollLast( );
            }

            deque.offer(i);
```

```
            // Add maximum element to the result array when
the window is complete
            if (i >= k - 1) {
                result[resultIndex++] = nums[deque.peek()];
            }
        }

        return result;
    }

    public static void main(String[] args) {
        int[] nums = {1, 3, -1, -3, 5, 3, 6, 7};
        int k = 3;

        int[] result = maxSlidingWindow(nums, k);

        // Print the result
        System.out.print("Output: [");
        for (int i = 0; i < result.length; i++) {
            System.out.print(result[i]);
            if (i < result.length - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
```

```
        }
    }
```

Hard:2

Example:1

Input: s = "aacecaad"

Output: "aaacecaad"

Ans:

```
public class ShortestPalindrome {

    public static String shortestPalindrome(String s) {
        int n = s.length( );

        // Create a new string by appending the reversed
substring of s to the end
        String rev = new StringBuilder(s).reverse( ).toString( );
        String newStr = s + "#" + rev;

        int[ ] lps = computeLPS(newStr);

        // Length of the longest palindromic prefix in the
concatenated string
        int len = lps[newStr.length( ) - 1];
```

```
        // Build the palindrome by appending characters from
the reversed substring
        String palindromeSuffix = rev.substring(0, n - len);

        return palindromeSuffix + s;
    }

    private static int[] computeLPS(String s) {
        int len = s.length( );
        int[] lps = new int[len];
        int j = 0;

        for (int i = 1; i < len; ) {
            if (s.charAt(i) == s.charAt(j)) {
                lps[i] = j + 1;
                j++;
                i++;
            } else {
```

```
            if (j != 0) {
                j = lps[j - 1];
            } else {
                lps[i] = 0;
                i++;
            }
        }
    }

    return lps;
}

public static void main(String[] args) {
    String s = "aacecaaa";
    String result = shortestPalindrome(s);

    // Print the result
    System.out.println("Output: " + result);
```

```
        }
    }
```