Question 01:

For inputs we have two strings named s and t. We will have three test cases. Test case 1: ("udacity", "ad"), test case 2: ("udacity", " "), and test case 3: ("udacity", "so"). For this problem we need to return a Boolean and at least need to use a for loop to cycle through the first string. For our code we first need the lengths of both strings. Convert string t into a list and sort it. We need a forloop to cycle through the strings. Create a list that stores a substring of s and sort it. We can then compare the substring to the sorted t string and return the results. We use sort which is merge sorting so the time complexity is going to be O(nlogn). The space complexity of this should be O(1).

Question 02:

For our input we will be using a string defined as a. We will have 4 test cases. Test case 1: (""), test case 2: ("racecar"), test case 3: ("udacity"), and test case 4: (5651651). For this problem we will at least need a nested for loop to cycle through the letters in the string and we will also need to check the length of the substring to figure out which is the longest. Check if the input is a string. We need a for loop that goes the length of the string and a nested for loop that increments the same range but starts and ends one after the previous for loop and store the values in a substring variable. Check to see if the substring is a palindromic and if it is the longest substring at the moment and storing it in a longest substring array. Return the value of the longest substring that we stored. Runtime is O(n^2) because it is two for loops. The space complexity is O(1).

Question 03:

For our input we have an undirected graph G. We will have 4 test cases. Test case 1: ({'A': [('B', 2)], 'B': [('A', 2), ('C', 5)], 'C': [('B', 5)]}), Test case 2: G is less than 2, no connection between the nodes ({'A': [('B', 2)]}), Test case 3: empty list ({}), Test case 4: not a dictionary (12323). From the keywords of undirected graph and minimum spanning tree we can deduce that we can use the logic behind Kruskal's Minimum Spanning Tree Algorithm. First, we need to sort the edges by weight. Then pick the smallest edges one by one but making sure no cycles form. Create the base cases to check if the input is empty, a dictionary type, or has a length less than 2. If none of the first three cases we need to create the graph so we use a nested for loop to turn G into a graph array. Start to create the function to create the minimum spanning tree using Kruskal's algorithm, defined as KruskalMST. Defined the find and union functions as helper functions for creating the tree. In KruskalMST we use a while loop to find the smallest available edge and check if adding it creates a cycle and if it does not, adds it to result array. We then use a for loop to store the contents of result into the results dictionary and return results. Sorting the edges takes O(ElogE) where E is the edge and the find and union operations are approximately O(logV) where v is the vertices so time complexity is approximately O(ElogE + ElogV). Space complexity is O(1).

Question 04:

For inputs we have the matrix of a tree defined as T, a non-negative number representing the root defined as r, and two non-negative integers defined as nodes n1 and n2. We have 3 test cases. Test case 1: ([[0, 1, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [1, 0, 0, 0, 1], [0, 0, 0, 0, 0]], 3, 1,) , test case 2: Not a tree (3,3,1,4), and test case 3: Tree is empty ([[]], 3, 1, 4). We need a Binary Search Tree. We also need to check both the left and right branches of each node to see if the values are bigger or smaller than the root. First, we need to create a Node class for our binary search tree. We then need to create a BinarySearchTree class to store functions for inserting nodes and searching for nodes. We then need to create a function to calculate what the least common ancestor is by comparing the root to n1 and n1. Time complexity of this solution is O(n) because it is determined by the depth of the tree. The space complexity is O(n) because the recursive function in lca needs extra space in the function call stack.

Question 05:

For our inputs we have a linked list LL and integer value m for elements from the end of the linked list. We have 3 test cases. Test case 1: question5(head, 3), test case 2: question5(head, 5), and test case 3: question5(head, 1). We need to create a linked list and should cycle through the linked list and find where the value is using m. First, we need to create the LL in the main function using the given Node class. Then we can define a copy of the LL in two list p1 and p2. Shift each value of p1 over 1 in a for loop with range m to create an empty value at the end. Produce a while loop that cycles until we get to the empty value in p1 and store the p2.next value into p2 and return the result. The time complexity of this solution is O(n log n) and the space time complexity is O(n^2).