Solution: 1

To check if whether two strings are anagram, all the charaters in both the strings are to be counted. If the count happens to be same, then both the strings are anagram.

Fisrt, a count dictionary for t is set to check with every substring set in s.the function returns true is set is anagram of t, else False. Comparing of the substrings can be done using loops. Looping through all the possible substring will consider the worst case of O(len(s)). The time complxity is O(len(s)).Since there are limited number of characters the space complexity is O(1).Empty strings are not considered anagram.

Solution: 2

A Palindrome is a string, which is same compared from both the ends.

Ex: ‘madam’ = ’madam’.

The Problem requires us to find the longest possible substring palindrome in a given string.Lets, find all the possible substrings og a string and check if a substring is a palindrome will solve the problem.If a palindrome is found, returns the longest palindrome found.space complexity is O(1) and time complexity is O(n\*n)

Solution: 3

Minimum spannning tree from an unidirected graph based on the weight of the edges between the vertices to be found. A Minimum spanning tree includes the edges with minimum weights from one vertex to another and must not have any cycles. Input is a graph represented in the form of dictionary.We need to iterate through the dictionary to read and then save all the edges to sort them by weight.we need to check if the path with a lesser wright is possible and if an edge exists in the minimum spanning tree.The time complexity for the algorithm will O(E) and space complexity will be O(V)

Solution: 4

We need to find the least common ancestor of two nodes in a binary search tree.

Binary search tree is representas as a 2D matrix , where each node is an integer.Given both the nodes given are in Bst.Lets take T as a tree and r as an integer valur of root, and n1 and n2 are non-negative integers. The approach is simple and can be recursively used considering Bst is sorted.The efficiency of the algorithm is O(n\*n)

Solution: 5

To solve this problem,we will loop through the linked list twice. The first iteration will provide its length . Then the second iteration can determine how many elements to traverse, to get the mth element from the back. The time will still be O(n) for traverse through ll twice. Since we only store the length, the space complexity will be O(1). One special case is to deal with circular linked list. If that case is not treated, then the get\_length() function might stuck in a infinite loop when it hits a circular linked list. We can use two nodes each traverse at different rate to deal with the case. If both of them ever meet, then we know the linked list is circular and we can terminate the loop. This part will not take additional time in traversal and will only need to store one additional node durning the traversal. Therefore the overall time and space complexity will not be affected.