Question1

To check whether the two strings are anagram or not, first we need to find the frequency of characters in each input string then for each character in the sub string we need to check if that character is present in the original string and the frequency of that character i.e., count of that particular character in sub string must be less than or equal to character count in original string. So, if a character is present in original string and the frequency of that character is less than original string then we return true else false. Because we compare only characters of sub string, the time complexity of this algorithm is O(len(substring)). To get frequency of characters we are using dictionary which takes O(len(str)) space complexity. If we exclude dictionary, we are just using Boolean variable as a flag and dictionary object of each string. So, the algorithm is taking only O(1) space complexity which is constant.

Question2

A palindromic string is one which is same when read from front and back. For example, if the given string is ‘getradarget’ then the longest palindromic substring is ‘radar’. So, we can get radar by checking each substring whether the substring is a palindrome or not, for odd length palindrome we fix a center and expand in both directions and for even length palindromes we fix two centre and expand in both directions. The time complexity of this algorithm is O(n^2) and constant space complexity because we only need to store left and right indexes.

Question3

A graph may have large number of spanning trees but not all of them are minimum. A minimum spanning tree is the one in which the sum of weights of edges is minimum. We can find minimum spanning tree using kruskal’s algorithm. So, first we need to sort all the edges in increasing order, pick the smallest one and check if adding this edge to our spanning tree obtained so far forms a cycle or not, if not, then this edge will be part of our spanning tree.. We need to keep repeating this step for all edges in G until there are v-1 edges in our minimum spanning tree. Used python dictionary data structure to store sets and ranks, which allows find and union operation in constant time. Sorting edges has complexity of O(nlogn) and iterating over the edges has O(n) complexity. Overall space complexity of the algorithm is O(n), n is number of nodes.

Question4

I tried but I was unable to understand how input can be transformed into a BST, so I made a tree with nodes similar to question 5. Search operation is performed using recursion technique, to find a node in given BST, first we need to compare the node to the root, if the node is root we return the root. If node is smaller than root we do recursion technique for left sub tree, similarly if the node is greater than root we recur right sub tree. We can find LCA by recursively traversing BST from root. While traversing from top to bottom if we find a value n and if this value is less than n1 and greater than n2 or if it is same as n1 or n2 then n is LCA of n1 and n2. So, we just need to recursively traverse the BST, if node value is greater than n1 and n2 then LCA is in left sub tree or vice versa. The time complexity of this algorithm is O(logn), logn is height of the tree and space complexity is O(logn), this extra space is used for function call stack for recursive function calls.

Question5

To find an element in a singly linked list that’s m elements from the end. We need two pointers, first pointer points to the head of linked list and second element points mth node from head. Now if we traverse both node parallel until second node reaches end of the linked list, first node reaches mth element from end. Since we are traversing just once to find the desired element, time complexity of the algorithm is O(n) and we are using two pointers. So, space complexity is O(1).