Time Complexity Analysis

Note: I also put comments before algorithms and after statements defining the time complexity of each method. The comments stating time complexity after each statement show how I determined the values to add for each algorithm to get the total time complexity.

Question 1-Longest Common Subsequence

allSubsequences(String str) algorithm -
n = str.length();
O(1) + O(1) + O(1) + O(2^n * (1 + 2n + 1)) + O(1)
O(2*2^n + 2n*2^n)
O(2n*2^n)
O(n*2^n)

Total Time Complexity- O(n*2^n)

longestCommonSubsequences(String word1, String word2) algorithm -
p = word1.length()
k = word2.length()
O(p * 2^p) + O(k * 2^k) + O(1) + O(1) + O(1) + O(p * 2^p * (2 * (k * 2^k)) + O(1)
O(p * 2^p * 2k * 2^k)
O(p * 2^p * k * 2^k)

So , the worst case time complexity is O(p * 2^p * k * 2^k)

Since the for loops always iterates the same amount of times and there is not a break in the for loop the best case time complexity is also OMEGA(p * 2^p * k * 2^k)

Question 2-Longest Common Substring
n = text1.length()
m = text2.length()
i is the firstIteratorVariable in the first loop
j is the secondIterator variable in the second loop
k is the third iterator variable in the third loop
l is the fourth iterator variable in the fourth loop

O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O((n(n * (j-i)* (m (m (l - k + 4) -1) / 2)- 1)) / 2) + O(1)

O((n(n * (j-i) * (m * (m (l - k + 4) -1) / 2)- 1)) / 2)
O((n(n * (j-i) * (m * (ml -mk +4 - 1) / 2) - 1)) / 2)
O((n(n * (j-i) * (m^2*l - m^2*k + 3m) / 2) - 1)) / 2)
O((n(nj - ni * (m^2*l - m^2*k + 3m) - 1)) / 2)
O((n(nj - ni * (m^2*l - m^2*k + 3m))) / 2)
O((n^2j - n^2i * (m^2*l - m^2*k + 3m))) / 2)
O((n^2j - n^2i * (m^2*l - m^2*k))) / 2)
O((n^2*j * m^2*l - n^2*j * m^2*k - n^2i * m^2*l + n^2i * m^2*k)/2)
O(n^2*j * m^2*l - n^2*j * m^2*k - n^2i * m^2*l + n^2i * m^2*k)

So , the worst case time complexity is O(n^2*j * m^2*l - n^2*j * m^2*k - n^2i * m^2*l + n^2i * m^2*k)

Since the for loops always iterates the same amount of times and there is not a break in any of the for loops the best case time complexity is also OMEGA(n^2*j * m^2*l - n^2*j * m^2*k - n^2i * m^2*l + n^2i * m^2*k) and since the bounds are equal we can define a bound for theta as THETA(n^2*j * m^2*l - n^2*j * m^2*k - n^2i * m^2*l + n^2i * m^2*k)

Question 3-notFibonacci

O(1) + O(1) O(1) + O(1) + O(1) + O((n -2) * (1+1))
O((n -2) * (2))
O((n -2) * (2))
O(2n -4)
O(n)

So the total worst-case time complexity O(n)

The loops have no conditional breaks in them besides i < n, so the best case is the same as the worst case making OMEGA(n) the best case time complexity. Additionally, since the worst case and best case time complexity are the same theta can be defined as THETA(n).

Question 4-Where In Sequence
n = notFibonacci.length
O(1) + O(1) + O(1) + O(1) + O(1) + O(1) + O(target) + O(n)
O(target + n)

So the total worst case time complexity is, O(target + n).

The best case time complexity would be if target == 0 in which the first if statement would be true and 1 would be returned, then the time complexity would be OMEGA(1).

Problem 5-Remove Element

n = nums.length
m = numsWithoutVal.length
k = occurencesOfVal
O(1) + O(n(2)) + O(1) + O(1) + O(m(k + 1 + 1) + O(n(1+1))
O(2n) + O(mk+2m) + O(2n)
O(mk + 4n + 2m)
O(mk + n + m)

So the worst case time complexity is O(mk + n + m)

The best case time complexity would be OMEGA(1) which would happen if nums was an empty array,
which would imply that n=0, m=0, and k=0
which would lead to the following time complexity OMEGA(1) + OMEGA(n(2)) + OMEGA(1) + OMEGA(1) + OMEGA(m(k + 1 + 1) + OMEGA(n(2))=OMEGA(6)=OMEGA(1)