Question 2:

Longest Common Substring Algorithm: logic is explained through comments

public static char[] longestCommonSubstring(char[] text1, char[] text2) {

int m = text1.length; // find the length of the first string

int n = text2.length; // find the length of the second string

int[][] dp = new int[m + 1][n + 1]; // DP table to store lengths of common substrings

int maxLength = 0; // stores the length of the longest common substring

int endIndex = 0; // Ending index of the longest common substring in text1

// Fill the DP table

for (int i = 0; i < m; i++) { //iterates through the first string

for (int j = 0; j < n; j++) { //iterates through the second string

if (text1[i] == text2[j]) { //checks to see if characters match at the current positon

dp[i + 1][j + 1] = dp[i][j] + 1; // Increment if characters match

if (dp[i + 1][j + 1] > maxLength) { //update the maxlength and endindex if a longer common substring is found

maxLength = dp[i + 1][j + 1];

endIndex = i; // Track the end index in text1

}

} else {

dp[i + 1][j + 1] = 0; // Reset if characters don't match

}

}

}

// If no common substring is found, return an empty char array

if (maxLength == 0) {

return new char[0];

}

// Extract the longest common substring using maxLength and endIndex

char[] result = new char[maxLength];

for (int i = 0; i < maxLength; i++) {

result[i] = text1[endIndex - maxLength + 1 + i]; //copies the characters of the common substring

}

return result; // returns the longest common substring as a character array

}

Question 6:

1. Time Complexity of Longest Common Subsequence is O(n\*m). Since it is a nested loop, we must analyze from the inside out. The body consists of an if statement which is a constant O(1). Then, the inner loop runs the length of the second string which is equal to n. The outer loop runs m times with m being the length of the first string. Since it is a nested loop, you multiply and the result is O(n\*m), and we cut off the constant. In the best-case scenario, the algorithm still iterates through the words using the nested loop n\*m times. The body is still constant. So Ω(m⋅n).
2. Time complexity of Longest Common Substring is O(n\*m). The nested loop’s body is constant while the inner loop runs n times. The outer loop runs m times which results in O(n\*m). In the best-case scenario, the algorithm still has the time complexity of Ω(m⋅n). This is because it must iterate through all of the characters to check if there are any more substrings.
3. Time complexity of Not Fibonacci Sequence is O(t) (t stands for number of Terms) since it uses one for loop to execute the number of calculations. The body is O(1) since it calculates the next term of the sequence. The time complexity can get longer as larger numbers are calculated, but since we are calculating the first 10 terms, it is not as bad. The best-case scenario is if the input is 1 or less since it will not need to iterate through the loop to print out these values. This means that it would be Ω(1).
4. Time complexity of Not Fibonacci position is O(n) since the for loop is executed until the input is found. The body of the for loop is a constant because of arithmetic which results in the time complexity being O(n). The best-case scenario is Ω(n) since the loop will have to iterate through the sequence to compare values.
5. Time complexity of Remove Elements is O(n), n being the length of the nums array. The algorithm uses one for loop to iterate through the values to check if they are equal to the value that is going to be removed from the list. The body of the loop is O(1) while the loop is O(n). This results in the time complexity of O(n). In the best case scenario, the loop will still have to run n times since it has to check all of the values in the array to make sure. The time complexity would be Ω(n).

Extra Credit:

A computer screen shot of a program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

There was an overflow issue since the number grew so large to the point where it was not able to display properly. However, when I applied logarithms to the values to make it more manageable to view and address the issue of overflowing.

A graph with a line

AI-generated content may be incorrect.