Question 2:

```
public class quest2 {


   public static String question2(String word1, String word2) {
      int w1L = word1.length();
      int w2L = word2.length();
      int[][] table = new int[w1L + 1][w2L + 1];
      int maxLength = 0;
      int endIndex = 0;


      for (int i = 1; i <= w1L; i++) {
         for (int j = 1; j <= w2L; j++) {
            if (word1.charAt(i-1) == word2.charAt(j-1)) {
               table[i][j] = 1 + table[i-1][j-1];


               if (table[i][j] > maxLength) {
                  maxLength = table[i][j];
                  endIndex = i;
               }
            }
            else {
               table[i][j] = 0;
            }
         }
      }
      if (maxLength == 0) {
         return "No substring";
      }
```

```
        return word1.substring(endIndex - maxLength, endIndex);
    }



    public static void main(String[] args) {
        System.out.println(question2("gears of war", "history of war"));
        System.out.println(question2("spy family", "pickle"));
        System.out.println(question2("hi", "talk"));



    }
}
```

Question 6:

Algorithim 1:

Time Complexity: My first algorithim is Big-O(w1L x w2L) and Big-Ω(w1L x w2L). Where w1L is the length of word1 and w2L is the length of word2. There is a nested for loop where the outer loop runs w1L times and a inner loop that runs w2L times. We go to each grid exactly once which is w1L x w2L.

Space Complexity: We initialize a 2 dimensonial array that is of size w1L x w2L.

This means that our algorithim is Big-O(w1L x w2L) and Big-Ω(w1L x w2L). Each cell gets written into so itis Big-O(w1L x w2L) and Big-Ω(w1L x w2L).

Algorithim 2:

Time Complexity: This algorithim is also Big-O(w1L x w2L) and Big-Ω(w1L x w2L). Where w1L is the length of word1

and w2L is the length of word2. There is a nested for loop where the outer loop runs w1L times

and a inner loop that runs w2L times.

Space Complexity: This is once again the same and is Big-O(w1L x w2L) and Big-Ω(w1L x w2L) since we are creating a

w1L x w2L array and filling each square.

Algorithim 3:

Time Complexity:

The time complexity of this will be Big-O(N) and Big-Ω(N). This is because there is one for loop that runs

numTerms where numTerms is the length of the sequence. It is followed by another loop that also runs numTerms where

numTerms is the length of the sequence. N + N = 2N and we can just cut off the constants.

Space Complexity:

The space complexity is Big-O(N) and Big-Ω(N). This is because we initialize a long array that

is the size of the sequence and fill it with constants. This is the biggest use of ram.

Algorithim 4:

Time Complexity:

The time complexity of this is Big-O(KL) and Big-Ω(K) where K is the index of the number that is equal or > than

the target because it will run until it gets to either one of those conditions.

Space Complexity:

The space complexity is Big-O(1) and Big-Ω(1) because we only create primitive data types and never

any array that would grow and not be constant. 1 is the constant space. we can factor out a

1 from how many times we intialize a primitive data type.

Algorithim 5:

Time Complexity:

The time complexity is Big-O(N) and Big-Ω(N) because inside of the logic we have a for loop that runs

nums.length where that is the length of the array (the input). It checks each number in the array which is why it happens

N times.

Space Complexity:

The space complexity is Big-O(1) and Big-Ω(1) becuase we are only ever writing into the array in the logic.

There is no point where we make an array in the logic it is only passed to us.