Assignment 1 - Coding & Complexity Kevin G

2. Common substring:

```
public static void commonSub(String text1, String text2){
char[] t1Array= text1.toCharArray(); → Make strings to
char[] t2Array= text2.toCharArray();   char array
char[] subsequence = new char[10]; → add subsequence
                                     to array to make the
                                     strings

for(int i=0; i< t1Arrny.length && i< t2Array.length; i++) { → loop thrug char
                                                              arrays
  if (t1Array[i] == t2Array[i]) { → chak if element equals
     subsequence [i] = t1Array [i] → add current element to
  }                                  subsequence.
  System.out.println(subsequence);

}
```

6. Algorithm Analysis

1 and 2 common subsequence & common substring:

```
public static void commonSub(String text1, String text2)
char[] t1Array = text1.toCharArray();   ] O(n)
char[] t2Array = text2.toCharArray();   ] O(m)
char[] subsequence = new char[10];      ] O(1)
int subsequenceCounter = 0;             ] O(1)
for(int i=0; i< t1Array.length && i< t2Array.length; i++){ } → O(n+m)
  if (t1Array[i] == t2Array[i]){        ] O(1)
     subsequence[i] = t1Array[i]        ] O(1)
     subsequencecounter = subsequenceCounter + 1; ] O(1)
}
System.out.println(subsequence);        ] O(1)
System.out.println(subsequenceCounter); ] O(1)
}
```

Assignment 1 - Cont...

date / /

1 & 2. When adding the contributions from all parts, we depend on the largest growing variable. In this case, $O(n+m)$ where n and m are equal to the size of the strings received.

Due to the string to char array step and how we loop thrugh these char arrays, the best case scenario would be the same.

$$O(n+m) \ \& \ \Omega(n+m)$$
$$\text{aka} \quad \Theta=(n+m)$$

3. Not Fibbonacci:

```
public static void notFibbonacci(int target){
long num1=0;          ] O(1)+O(1)=O(2)→O(1)
long num2=1;          ]

for(int i=0; i<target; i++){ ] iterates O(n)  n=target
    system.out.println(num1+" "); ] O(1)

long num3=(3*num2)+(2*num1); ]
num1=num2;                   ] O(3)=O(1)
num2=num3;                   ]
```

This code is full of constant atomics that assign values and do basic math. the fastest growing variable would be be n=target since that is how many times our loop iterates. O(n)

Since this runs until its done iterating, the best case would also be $\Omega(n)$.

$$O(n)+\Omega(n)=\Theta(n)$$

Assignment 1 - Cont...

date / /

4. Where in Sequence:

```
public static void whereInSequence(int target)
int[] notFibonnacci = {0, 1, 3, 11, 39, 139, 495, 1763, 6279, 22363};  O(1)
for(int i=0; i < notFibonnacci[i]){ }  O(10) = O(1)
  if(target == notFibomacci[i]){
    System.out.println(i+1);
  } else if ( target < notFibonnacci[i]){
    system.out.println(i+1);
  } elself ( target != notFibonnacci[9]){
   system.out.println(10)
  break
}
```

O(1)

This code could haver been a code with a time complexity of O(n), however, I am making the array a fixed size so the loop will simply run 10 times in its worse scenario $O(10) = O(1)$

As for the best case, the target will be found on the first element of the array $\Omega(1)$

$$O(1) \; \} \; \Omega(1) = \Theta(1)$$

5. removeElement:

```
public static void removeElement(int[] nums, int val){
    int counter = 0;  O(1)
for(int i=0; i < nums.lengthi;i++){ }  O(n)
    if(nums[i] != val){
       counter++;
    }
  }
  system.out.println(counter)
}
```

O(4) = O(1)

Assignment 1: Cont...

date / /

Since the code needs to loop through the entire array to check every element. This loop will always run n= nums.

$$O(n) \text{ } \} \Omega(n) = \theta(n)$$