


Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Stelian Adrian		



## Activity 1. Validation results.

We test the first case: **GCCCTAGCG** and **GCGCAATG**

And we obtain the following results:

For the dynamic programming version:

	*	*	G	C	C	C	T	A	G	C	G
*	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)
G	0(0,0)	1(0,0)	1(1,1)	1(2,1)	1(3,1)	1(4,1)	1(5,1)	1(6,0)	1(7,1)	1(8,0)	
C	0(0,0)	1(1,1)	2(1,1)	2(2,1)	2(3,1)	2(4,2)	2(5,2)	2(6,2)	2(7,1)	2(8,2)	
G	0(0,0)	1(1,2)	2(2,2)	2(3,2)	2(4,2)	2(5,2)	2(6,2)	3(6,2)	3(7,3)	3(8,2)	
C	0(0,0)	1(1,3)	2(2,3)	3(2,3)	3(3,3)	3(4,4)	3(5,4)	3(7,3)	4(7,3)	4(8,4)	
A	0(0,0)	1(1,4)	2(2,4)	3(3,4)	3(4,4)	3(5,4)	4(5,4)	4(6,5)	4(8,4)	4(9,4)	
A	0(0,0)	1(1,5)	2(2,5)	3(3,5)	3(4,5)	3(5,5)	4(6,5)	4(7,5)	4(8,5)	4(9,5)	
T	0(0,0)	1(1,6)	2(2,6)	3(3,6)	3(4,6)	4(4,6)	4(6,6)	4(7,6)	4(8,6)	4(9,6)	
G	0(0,0)	1(1,7)	2(2,7)	3(3,7)	3(4,7)	4(5,7)	4(6,7)	5(6,7)	5(7,8)	5(8,7)	

Longest subsequence:

GCCAG

For the recursive version:

Longest subsequence:

GCCAG

Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci		
	Name: Stelian Adrian		

```

Printing longest subsequence...
GCCAG

/*****/

RECURSIVE:
Finding longest subsequence...
GCCAG
Program terminated.

```

We test the second case: [GCATGCAT](#) and [GAATTCAG](#)

And we obtain the following results:

For the dynamic programming version:

	*	*	G	C	A	T	G	C	A	T
*	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)	0(0,0)
G	0(0,0)	1(0,0)	1(1,1)	1(2,1)	1(3,1)	1(4,0)	1(5,1)	1(6,1)	1(7,1)	
A	0(0,0)	1(1,1)	1(2,1)	2(2,1)	2(3,2)	2(4,2)	2(5,2)	2(6,1)	2(7,2)	
A	0(0,0)	1(1,2)	1(2,2)	2(3,2)	2(4,2)	2(5,2)	2(6,2)	3(6,2)	3(7,3)	
T	0(0,0)	1(1,3)	1(2,3)	2(3,3)	3(3,3)	3(4,4)	3(5,4)	3(7,3)	4(7,3)	
T	0(0,0)	1(1,4)	1(2,4)	2(3,4)	3(4,4)	3(5,4)	3(6,4)	3(7,4)	4(8,4)	
C	0(0,0)	1(1,5)	2(1,5)	2(3,5)	3(4,5)	3(5,5)	4(5,5)	4(6,6)	4(8,5)	
A	0(0,0)	1(1,6)	2(2,6)	3(2,6)	3(4,6)	3(5,6)	4(6,6)	5(6,6)	5(7,7)	
G	0(0,0)	1(1,7)	2(2,7)	3(3,7)	3(4,7)	4(4,7)	4(6,7)	5(7,7)	5(8,7)	

Longest subsequence:

GATCA

For the recursive version:

Longest subsequence:

GATCA

Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci		
	Name: Stelian Adrian		

```

Currently at cell [ 0, 0 ]
Printing longest subsequence...
GATCA

/*****/

RECURSIVE:
Finding longest subsequence...
GATCA
Program terminated.

```

Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci		
	Name: Stelian Adrian		

## Activity 2. Experimental time measurements.

Experimental values:

n	t_dynamic (ms)
100	1.8
200	4.3
400	12.2
800	34.9
1600	152.9
3200	1071.8
6400	5160.4

n	t_recursive (ms)
2	0.0
4	0.2
6	1.6
8	26.8
10	444.7
12	13064.8
13	73363.4

## Activity 3. Answer to questions.

A) Determine theoretically complexities (time, memory space and waste of stack) for both implementations, recursive (approximated) and using programming dynamic.

Recursive:

The time complexity of this algorithm can be calculated using the following information:

### ➤ Complexity

- $O(n^k)$  if  $a < 1$  (never happens)
- $O(n^{k+1})$  if  $a = 1$
- $O(a^{n \div b})$  if  $a > 1$

Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci		
	Name: Stelian Adrian		

And taking into account that for this algorithm  $a = 3$  (because we make 3 recursive calls to the method) and  $b = 1$  (we reduce by 1 the size of the problem in each call). The value of  $k$  does not influence the complexity in this case. Having said that, we can see that the complexity of the algorithm is  $O(3^n)$ .

Regarding stack memory consumption, we have to take into account the height of the tree of states that this method generates. It is close to  $n$ , so the stack memory consumption is  $O(n)$ .

Dynamic:

We can deduce the time complexity looking at the code. We can see that it is  $O(n)$  as there is only one for loop, which depends on the size of the second string introduced as argument.

When taking measurements, though, we have to be careful, because if the `fillTable()` method is used the complexity goes up to  $O(n^2)$ . For the memory consumption we have to, again, look at the tree of states generated by the method, and we can conclude that the stack memory consumption is  $O(1)$ , as it does not consume much stack memory.

B) Compute theoretical times and compare them with the experimental measurements.

Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci		
	Name: Stelian Adrian		

Experimental values:

n	t_dynamic
100	2.0
200	6.5
400	15.5
800	33.5
1600	206.0
3200	951.0
6400	3748.0

n	t_recursive
2	0.0
4	0.2
6	1.6
8	26.8
10	444.7
12	13064.8
13	73363.4

We compute the theoretical values taking into account the following formula:

$$t2 = \frac{f(n2)}{f(n1)} * t1$$

And also considering the previously computed complexity of both versions.

First, we calculate the values of the recursive version  $O(3^n)$ :

n	t_recursive (theoretical)
10	241.2
12	2170.8
13	6512.4

And we can observe that there is some difference with the experimental values. This happens because when calculating the theoretical times we are

Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci		
	Name: Stelian Adrian		

not taking into account the constants, and that explains the difference. We can see, though, that the tendency of both sets of values is similar.

Now, we do the same for the dynamic version:

n	t_dynamic (theoretical)
1600	132
3200	824
6400	3296

We notice that the times are quite similar. In the first case, we used a value which is below the threshold of accepted values (33.5ms), which might explain why we had such a difference. For calculating  $t_2$  with  $n_2 = 3200$  and  $n_1 = 1600$  we made use of  $t_1 = 206\text{ms}$ , to not increase the difference due to this flawed value.

We have to mention that here the complexity used is  $O(n^2)$ , as we are using the method `fillTable()`, whose complexity is bigger than that of the dynamic algorithm itself.

C) Why large sequences cannot be processed with the recursive implementation? Explain why dynamic programming implementation raises an exception for large sequences.

You cannot process large sequences with the recursive version because the memory available in the stack is not enough and it raises a `StackOverflow` error.

Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci		
	Name: Stelian Adrian		

When large sequences are used in the dynamic programming implementation the following exception is raised:

```
Time [ms]= 956.0, n=3200
Time [ms]= 4737.0, n=6400
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at algstudent.s5.LCS.initTable(LCS.java:63)
    at algstudent.s5.LCSTimes.main(LCSTimes.java:20)
```

This happens because the amount of memory needed to store the table that we need to create is bigger than the amount we have available. We do not have space to store it in memory.

D) The amount of possible LCS can be more than one, e. g. GCCCTAGCG and GCGCAATG has two GCGCG and GCCAG. Find the code section that determines which subsequence is chosen, modify this code to verify that both solutions can be achieved.

The code section that we have to modify is this one (in the recursive version), which decides what string is the longest:



Algorithmics	Student information	Date	Number of session
	UO:UO277653	25/03/2021	7
	Surname: Stanci		
	Name: Stelian Adrian		

```

/**
 * Find a MSC directly by a recursive approach
 */
public String findLongestSubseq(String s1, String s2){
    String L1, L2, L3;
    if(!(s1.length()==0) && !(s2.length()==0)) {

        L1 = findLongestSubseq(s1.substring(0, s1.length() - 1), s2);
        L2 = findLongestSubseq(s1, s2.substring(0, s2.length() - 1));
        L3 = findLongestSubseq(s1.substring(0, s1.length() - 1), s2.substring(0, s2.length() - 1));

        if(s1.substring(s1.length() - 1, s1.length()).equals(s2.substring(s2.length() - 1, s2.length())) {
            L3 = L3 + s1.substring(s1.length() - 1, s1.length());
        }
        if (L1.length() > L2.length() && L1.length() > L3.length()) {
            return L1;
        }

        if (L2.length() >= L1.length() && L2.length() >= L3.length()) { // Remove the = for the other version
            return L2;
        }

        return L3;
    }
    return "";
    // TODO: find directly a MSC (without a table) of two input sequences using recursion
}

```

If in that condition we remove the = from the >= we get “GCCAG” when putting GCCCTAGCG and GCGCAATG as inputs, but, if we put those = we get GCGCG. Notice that in the end this change does not modify the length of the result, only the letters. Varying the == and != it is possible to obtain more combinations.