Algorithmics	Student information		Date	Number of session	
	UO:283428		02/03/2022	3	
	Surname: Triana FERNÁNDEZ				
	Name: Carlos				

Activity 1. Basic recursive models.

Subtraction1.java:

This algorithm implements a divide a conquer approach by subtraction in which:

The number of sub problems is only one, as there is only one recursive call per iteration.

The size of the new problem is n -1.

The complexity of the overall algorithm excluding the recursive calls is linear.

In order to calculate the equation that follows the algorithm, we have the following parameters: a = 1, b = 1, k = 0, therefore, a = 1; where a is the number of sub problems, b is the new size and k the degree of the complexity by itself.

The complexity of this algorithm is O(n^(k+1)), then O(n^1).

If we take measurements on this algorithm with 100000000 repetitions, we can see the linearity.

n:	1	2	4	8	16	32	64
time:	5	168	354	639	1226	2652	6212

Subtraction2.java:

This divide and conquer by subtraction algorithm:

Generates only one sub problem, as it only has one recursive call.

Reduces the problem in one unit.

When compared with the previous algorithm, its complexity without recursion is quadratic.

Therefore: a = 1, b = 1, k = 1;

The formula of the previous algorithm applies here too, its complexity is O(n^(k+1)), then O(n^2).

Measuring time with 100000000 repetitions shows a quadratic progression.

n:	1	2	4	8	16	32	64
time:	206	554	1214	3610	9433	32650	129525

Algorithmics	Student information	Date	Number of session
	UO:283428	02/03/2022	3
	Surname: Triana FERNÁNDEZ		
	Name: Carlos		

Subtraction3.java

This algorithm is a version of the Substraction1.java in which each recursive call generates two sub problems.

The parameters for finding its complexity are a = 2, b = 1 and k = 0.

As the number of sub problems generated is greater than 1, the complexity is $O(2^{(n/b)})$, then $O(2^{(n)})$. We can observe the exponential complexity.

n:	3	4	5	6	7	8	9
time:	655	2961	3188	12419	13455	50580	54410

Division1.java:

This algorithm is based on the same divide and conquer strategy, but by division and not by subtraction. We can see that every recursive call, the problems size is divided by a constant.

This version:

Generates only one sub problem with each call.

Divides the size of the problem by 3.

Has a complexity, all alone, of O(n).

We can infer that the parameters for finding the complexity are a = 1, b = 3, k = 1.

The complexity of the algorithm is $O(n^k)$, then $O(n)$.

n:	1	2	4	8	16	32	64
time:	6	241	502	854	1569	2785	5304

Division2.java:

This version of the algorithm divides the problem into two sub problems with a size 2 times smaller than the previous one.

The problem is divided into 2 new ones, a = 2.

The size of the problem is divided by 2, b = 2.

Algorithmics	Student information		Date	Number of session
	UO:283428		02/03/2022	3
	Surname: Triana FERNÁNDEZ			
	Name: Carlos			

The complexity of the problem is O(n), k = 1.

As a is equal to b, the problem follows a logarithmic complexity of O(n * logn).

n:	1	2	4	8	16	32	64
time:	8	1013	1678	5336	9323	2785	54199

Division3.java:

This version has the same number of sub problems through each iteration, with the same size, but its complexity alone is O(1). Therefore, a = 2, b = 2, k = 0.

As 2 is greater than 2^0, the temporal complexity of the algorithm is O(n^(log2(2))), then O(n).

n:	1	2	4	8	16	32	64
time:	6	648	738	3187	3593	12878	14926

Custom methods:

Subtraction4.java:

The aim of this algorithm is to have a number of sub problems greater than 1 to fit the formula O(a^(n/b)), where a is the number of sub problems or recursive calls, 3, and b is the number by which the size is reduced, in this case 2.

Division4.java:

The aim of this algorithm is to have a number of sub problems, a, lower than b, the number which the size is divided by, in order to have a complexity that depends on the complexity of the algorithm itself, k.

This way, if k is 2, the complexity will be O(n^2).

Activity 2. Tromino Times.

Algorithmics	Student information		Date	Number of session
	UO:283428		02/03/2022	3
	Surname: Triana FERNÁNDEZ			
	Name: Carlos			

1. What should be the time complexity of the algorithm?

This solution of the Tromino problem, is developed with a divide and conquer approach. The problem size is divided in two, by making four new recursive calls. The complexity of the algorithm without the recursion is n^0.

However, the overall complexity of the algorithm is calculated with the following expression: $O(n^{(\log b(a))})$, where b is two and a is four, this way the complexity is $O(n^2)$ as $\log 2(4)$ is 2.

2. Check if the time obtained in the previous section does or does not meet the theoretical complexity of the algorithm.

The following measurements are in microseconds:

n:	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
t:	3	1	4	12	18	63	209	713	2588	10136	48149	246338	907936

The theoretical complexity of the algorithm is quadratic, so we can conclude that each time t2 can be calculated by the square of the increment in size: $t2 = n^2 * t1$.

A time1 2588 will be preceded by a time2 that is 4 time the previous one, approximately 10352, the real following value is (10136). All the values follow the expected progression.