


Algorithmics	Student information	Date	Number of session
	UO: UO294786	06/03/24	5 & 6
	Surname: Álvarez Iglesias	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Rafael		



Activity 1. [DIVIDE AND CONQUER BY SUBTRACTION]

YOU ARE REQUESTED TO:

After analyzing the complexity of the three previous classes, you are not asked to make the timetables, but to reason whether the times match the theoretical time complexity of each algorithm.

For Substraction1 and 2 with complexities of $O(n)$ and $O(n^2)$ we can apply their corresponding formulas to determine if the time matches.

For Substraction1:

We take, for example, the time at $n = 64$ which is 17735ms. Applying the formula, we get that the theoretical time at $n = 32$ should be 8868ms. Comparing it with the obtained time of 9406ms we can ensure that Substraction1 behaves according to its complexity.

For Substraction2:

We take, for example, the time at $n = 16$ which is 55202ms . Applying the formula, we get that the theoretical time at $n = 8$ should be 13800ms. Comparing it with the obtained time of 17582ms we can ensure that Substraction2 behaves according to its complexity.

For what value of n do the Subtraction1 and Subtraction2 classes stop giving times (aborted)?

After an $n = 8192$ both programs stop due to a StackOverflowError. Both programs stop at the same time because both have the same $O(n)$ memory expenditure.

Why does that happen?

Because of the used approach, which tends to use a lot of memory, therefore rapidly overflowing the stack.

How many years would it take to complete the Subtraction3 execution for $n=80$?

If we calculate the time it would take to complete for $n = 80$, using the formula for its complexity $O(2^n)$ and its time of 110369 for $n = 27$ we obtain a time of $9,9411 \cdot 10^{17}$ s which equates to $3,152 \cdot 10^{10}$ years.

Algorithmics	Student information	Date	Number of session
	UO: UO294786	06/03/24	5 & 6
	Surname: Álvarez Iglesias		
	Name: Rafael		

Implement a Subtraction4.java class with a complexity $O(n^3)$ and then fill in a table showing the time (in milliseconds) for $n=100, 200, 400, 800, \dots$ (until OoT).


n	Time (ms)
100	58
200	453
400	3606
800	28814
1600	OoT

Implement a Subtraction5.java class with a complexity $O(3^{n/2})$ and then fill in a table showing the time (in milliseconds) for $n=30, 32, 34, 36, \dots$ (until OoT).

n	Time (ms)
30	847
32	2524
34	7556
36	22627
38	OoT

How many years would it take to complete the Subtraction5 execution for $n=80$?

If we calculate the time it would take to complete for $n = 80$, using the formula for its complexity $O(3^{n/2})$ and its time of 67860 for $n = 38$ we obtain a time of $7,425 * 10^{21}$ s which equates to $2,354 * 10^{14}$ years.

Algorithmics	Student information	Date	Number of session
	UO: UO294786	06/03/24	5 & 6
	Surname: Álvarez Iglesias	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Rafael		



Activity 2. [DIVIDE AND CONQUER BY DIVISION]

YOU ARE REQUESTED TO:

After analyzing the complexity of the three previous classes, you are not asked to make the timetables, but to reason whether the times match the theoretical time complexity of each algorithm.

Division1 and 3 both have a complexity of $O(n)$, whereas Division2 has $O(n * \log(n))$, we can apply their corresponding formulas to determine if the time matches.

For Division1:

We take, for example, the time at $n = 1024$ which is 26523ms. Applying the formula, we get that the theoretical time at $n = 512$ should be 13261ms. Comparing it with the obtained time of 13036ms we can ensure that Division1 behaves according to its complexity.

For Division2:

We take, for example, the time at $n = 256$ which is 73498ms . Applying the formula, we get that the theoretical time at $n = 128$ should be 32155ms. Comparing it with the obtained time of 34401ms we can ensure that Division2 behaves according to its complexity.

For Division3:

We take, for example, the time at $n = 256$ which is 44319ms . Applying the formula, we get that the theoretical time at $n = 128$ should be 22160ms. Comparing it with the obtained time of 21999ms we can ensure that Division3 behaves according to its complexity.

Algorithmics	Student information	Date	Number of session
	UO: UO294786	06/03/24	5 & 6
	Surname: Álvarez Iglesias		
	Name: Rafael		

Implement a **Division4.java** class with a complexity $O(n^2)$ (with $a < bk$) and then fill in a table showing the time (in milliseconds) for $n=1000, 2000, 4000, 8000, \dots$ (up to OoT).

n	Time (ms)
1000	187
2000	751
4000	2980
8000	11989
16000	47782
32000	OoT

Implement a **Division5.java** class with a complexity $O(n^2)$ (with $a > bk$) and then fill in a table showing the time (in milliseconds) for $n=1000, 2000, 4000, 8000, \dots$ (up to OoT).

n	Time (ms)
1000	74
2000	241
4000	913
8000	3754
16000	14545
32000	58220
64000	OoT

Algorithmics	Student information	Date	Number of session
	UO: UO294786	06/03/24	5 & 6
	Surname: Álvarez Iglesias		
	Name: Rafael		

Activity 3. [TWO BASIC EXAMPLES]

YOU ARE REQUESTED TO:

After analyzing the complexity of the various algorithms within the two classes, executing them and after putting the times obtained in a table, compare the efficiency of each algorithm.

n	Time(ms)		
	sum1	sum2	sum3
3	0,0000635	0,0001452	0,0001797
6	0,0000979	0,000211	0,000374
12	0,0001474	0,0003391	0,000749
24	0,0002418	0,0006076	0,0015121
48	0,0004333	0,0012314	0,0030216
96	0,000822	0,0022395	0,006027
192	0,001579	0,0044327	OoT
384	0,0031088	OoT	OoT
768	OoT	OoT	OoT

In VectorSum we have 1 iterative and 2 recursive methods, all with a complexity of $O(n)$. The best approach according to the data is the sum1 approach, the iterative method. Even though sum2 and sum3 have the same complexity, and both have a recursive approach, sum2 seems to have a better performance. This could be explained by analyzing their divide and conquer variables, as sum2 has ($a=1$, $b=1$, $k=0$) and sum3 ($a=2$, $b=2$, $k=0$) and so each call to the method causes another 2.

n	Time(ms)			
	fib1	fib2	fib3	fib4
100	0,0009781	0,0017337	0,0020205	OoT
200	0,0018979	0,0033632	0,004048	OoT
400	0,0037181	0,0065557	OoT	OoT
800	0,0073595	OoT	OoT	OoT
1600	OoT	OoT	OoT	OoT

In the Fibonacci case, first two methods are iterative, and the last two are recursive. All fib1,2 and 3 have a complexity of $O(n)$, whereas fib4 has a complexity of $O(1.6^n)$. The best performing method is fib1, followed by fib2 and fib3. The difference in performance of the methods is quite noticeable. Finally, the worst complexity of fib4 means that no result fell inside the timeframe. It is worth noting that to obtain conclusive measurements, an n starting from 100 and being multiplied by 2 was used.

Algorithmics	Student information	Date	Number of session
	UO: UO294786	06/03/24	5 & 6
	Surname: Álvarez Iglesias		
	Name: Rafael		

Activity 4. [ANOTHER TASK]

YOU ARE REQUESTED TO:

Implement a `Mergesort.java` class that orders the elements of a vector. Implement a `MergesortTimes.java` class that, after being executed, serves to fill in the following table.

n	Time(ms)		
	Ordered	Reverse	Random
31250	LoR	LoR	LoR
62500	LoR	LoR	61
125000	104	99	128
250000	203	199	266
500000	433	413	533
1000000	884	871	1144
2000000	1750	1724	2258
4000000	3623	3554	4802
8000000	7583	7252	9545
16000000	15378	14996	20137
32000000	31456	30689	40910
64000000	OoT	OoT	OoT

Fill in the following table, transferring to it the times obtained for the Quicksort in the random case (from lab 2) and those obtained here for the Mergesort, also in the random case. What constant do you get as a comparison of both algorithms?

n	Time(ms)		
	tMergesort(t1)	tQuicksort(t2)	t1/t2
250000	266	161	1,65217391
500000	533	344	1,5494186
1000000	1144	728	1,57142857
2000000	2258	1585	1,42460568
4000000	4802	3552	1,35191441
8000000	9545	8394	1,13712175

Algorithmics	Student information	Date	Number of session
	UO: UO294786	06/03/24	5 & 6
	Surname: Álvarez Iglesias		
	Name: Rafael		

By obtaining the constant for the algorithms, we can compare them and confidently say that, even though they share its complexity for a random case, Quicksort is faster than Mergesort given that the constant is greater than 1.