Algorithmics	Student information	Date	Number of session
	UO: UO300535	27-02-2025	3
	Surname: Cabo Stroup		
	Name: José David		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. D&C by Subtraction

- 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.

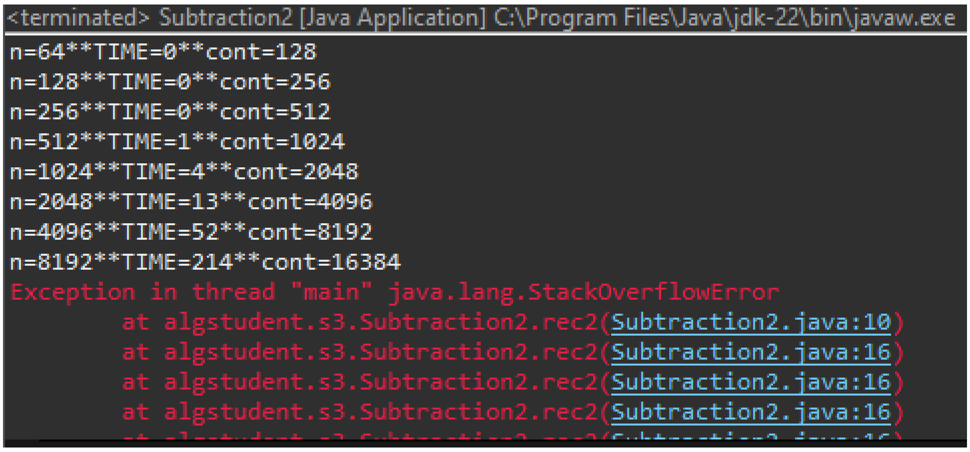
The algorithms in Subtraction1.java and Subtraction2.java are respectively linear and quadratic in their time complexity, although the empirical data is so small in the first one as to make the trend unnoticeable, and Subtraction2.java looks linear in the first few iterations; we'd have to measure them with a lot greater problem sizes for their real growth trends to become apparent.

- For what value of n do the Subtraction1 and Subtraction2 classes stop giving times (we abort the algorithm because it exceeds 1 minute)? Why does that happen?

They both seem to overflow once n > 8192, which is way before the times get even close to a minute. This happens because of the massive amount of times the method calls itself, therefore pushing more information to the stack than it can actually hold.

```
<terminated> Subtraction1 [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe
n=64**TIME=0**cont=1
n=128**TIME=0**cont=1
n=256**TIME=0**cont=1
n=512**TIME=0**cont=1
n=1024**TIME=0**cont=1
n=2048**TIME=1**cont=1
n=4096**TIME=0**cont=1
n=8192**TIME=0**cont=1
Exception in thread "main" java.lang.StackOverflowError
	at algstudent.s3.Subtraction1.rec1(Subtraction1.java:10)
	at algstudent.s3.Subtraction1.rec1(Subtraction1.java:15)
	at algstudent.s3.Subtraction1.rec1(Subtraction1.java:15)
	at algstudent.s3.Subtraction1.rec1(Subtraction1.java:15)
```

Algorithmics	Student information	Date	Number of session
	UO: UO300535	27-02-2025	3
	Surname: Cabo Stroup		
	Name: José David		



```
<terminated> Subtraction2 [Java Application] C:\Program Files\Java\jdk-22\bin\javaw.exe
n=64**TIME=0**cont=128
n=128**TIME=0**cont=256
n=256**TIME=0**cont=512
n=512**TIME=1**cont=1024
n=1024**TIME=4**cont=2048
n=2048**TIME=13**cont=4096
n=4096**TIME=52**cont=8192
n=8192**TIME=214**cont=16384
Exception in thread "main" java.lang.StackOverflowError
	at algstudent.s3.Subtraction2.rec2(Subtraction2.java:10)
	at algstudent.s3.Subtraction2.rec2(Subtraction2.java:16)
	at algstudent.s3.Subtraction2.rec2(Subtraction2.java:16)
	at algstudent.s3.Subtraction2.rec2(Subtraction2.java:16)
```

- How many years would it take to complete the Subtraction3 execution for n=80? Reason the answer.

Subtraction3.java, however, is exponential (O(2^n)). Taking [n_1 = 30, t_1 = 38,051 ms] as a reference point, we get that:

$$t_2 = \frac{2^{n_2}}{2^{n_1}} \times t_1 = 2^{n_2 - n_1} \times t_1 = 2^{80-30} \times t_1 = 2^{50} \times 38051 \approx 4.28 \times 10^{19}\, ms$$

Which is about **1,358.5 million years.** Efficient!

- Implement a Subtraction4.java class with a complexity O(n3) and then fill in a table showing the time (in milliseconds) for n=100, 200, 400, 800, ... (until OoT).
- Implement a Subtraction5.java class with a complexity O(3n/2) and then fill in a table showing the time (in milliseconds) for n=30, 32, 34, 36, … (until OoT).

The times for Subtraction4.java and Subtraction5.java are as follows:

Algorithmics	Student information	Date	Number of session
	UO: UO300535	27-02-2025	3
	Surname: Cabo Stroup		
	Name: José David		

	Time (ms)
n	Subtraction4.java
100	LoR
200	LoR
400	81
800	615
1600	4870
3200	39568
6400	OoT
12800	OoT
25600	OoT
51200	OoT

	Time (ms)
n	Subtraction5.java
30	373
32	1083
34	3235
36	9934
38	29184
40	89494
42	OoT
44	OoT
46	OoT
48	OoT

- How many years would it take to complete the Subtraction5 execution for n=80? Reason the answer.

For n=80, Subtraction5.java would take

$$t_2 = \frac{3^{\frac{n_2}{2}}}{3^{\frac{n_1}{2}}} \times t_1 = 3^{\frac{n_2}{2}-\frac{n_1}{2}} \times t_1 = 3^{\frac{80}{2}-\frac{40}{2}} \times t_1 = 3^{20} \times 89494 \approx 3.12 \times 10^{14}\, ms$$

Or approximately **9,894.9 years.** Hey, at least it's better than Subtraction3.java...

Activity 2. D&C by Division

- 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.

Indeed, they do. Division1.java and Division3.java are linear, as the times would suggest. Division2.java, on the other hand, is O(n*log n), which is also coherent with the data when measured.

- Implement a Division4.java class with a complexity O(n^2) (with a<b^k) and then fill in a table showing the time (in milliseconds) for n=1000, 2000, 4000, 8000, … (up to OoT).
- Implement a Division5.java class with a complexity O(n^2) (with a>b^k) and then fill in a table showing the time (in milliseconds) for n=1000, 2000, 4000, 8000, … (up to OoT).

	Time (ms)
n	Division4.java
100	LoR
200	LoR
400	66
800	253
1600	1042
3200	4059
6400	16387
12800	OoT
25600	OoT
51200	OoT

	Time (ms)
n	Division5.java
100	LoR
200	98
400	390
800	1510
1600	6046
3200	23673
6400	OoT
12800	OoT
25600	OoT
51200	OoT

Activity 3. Two basic examples

Algorithmics	Student information	Date	Number of session
	UO: UO300535	27-02-2025	3
	Surname: Cabo Stroup		
	Name: José David		

	Fibonacci Times (ms)					Fibonacci Times (ms)			
n	**fib1**	**fib2**	**fib3**	**fib4**	**n**	**fib1**	**fib2**	**fib3**	**fib4**
10	0,000091	0,000127	0,000209	0,00261	**35**	0,000187	0,000398	0,000602	433
11	0,000089	0,000138	0,000216	0,00417	**36**	0,0002	0,000432	0,000651	696
12	0,000091	0,000148	0,000232	0,00679	**37**	0,000197	0,00044	0,000637	1129
13	0,000097	0,000158	0,000249	0,01148	**38**	0,0002	0,000429	0,000646	1857
14	0,000099	0,000168	0,000264	0,01846	**39**	0,000213	0,000446	0,000669	3000
15	0,000107	0,000178	0,000278	0,03078	**40**	0,000215	0,000449	0,000683	4920
16	0,000109	0,000188	0,000293	0,04917	**41**	0,000221	0,00046	0,000703	8030
17	0,000111	0,000199	0,000312	0,07865	**42**	0,000224	0,000476	0,000719	12737
18	0,000117	0,000209	0,000346	0,12842	**43**	0,000235	0,000487	0,000733	20344
19	0,00012	0,000221	0,000371	0,2027	**44**	0,000232	0,000493	0,000755	33088
20	0,000128	0,000233	0,000361	0,318	**45**	0,000227	0,000508	0,000753	54959
21	0,00013	0,000242	0,000389	0,516	**46**	0,000234	0,000518	0,000774	OoT
22	0,000132	0,000255	0,000401	0,829	**47**	0,000242	0,000531	0,000788	OoT
23	0,000137	0,000266	0,000409	1,364	**48**	0,000238	0,000542	0,000808	OoT
24	0,00014	0,000278	0,000425	2,196	**49**	0,000245	0,000577	0,00083	OoT
25	0,000158	0,000289	0,000434	3,561	**50**	0,000266	0,000572	0,000843	OoT
26	0,000148	0,000299	0,000451	5,754	**51**	0,000253	0,000589	0,000849	OoT
27	0,000152	0,00031	0,000505	9,204	**52**	0,000257	0,000586	0,000867	OoT
28	0,000157	0,00032	0,000486	14,901	**53**	0,00026	0,00059	0,000875	OoT
29	0,000162	0,00033	0,000501	24,103	**54**	0,000266	0,000601	0,00091	OoT
30	0,000162	0,000342	0,000524	38,866	**55**	0,000269	0,000606	0,000945	OoT
31	0,000172	0,000357	0,000532	63,765	**56**	0,000273	0,000616	0,000934	OoT
32	0,000173	0,000364	0,00055	102	**57**	0,000277	0,000627	0,000947	OoT
33	0,000178	0,000383	0,000577	167	**58**	0,00028	0,000643	0,000953	OoT
34	0,00018	0,000392	0,000588	266	**59**	0,000287	0,000651	0,000971	OoT

	VectorSum Times (ms)		
n	**sum1**	**sum2**	**sum3**
3	0,0000395	0,000077	0,000088
6	0,0000623	0,000117	0,00016
12	0,0000856	0,000234	0,000345
24	0,0001314	0,000427	0,000704
48	0,0002226	0,000811	0,001448
96	0,0004046	0,001598	0,002894
192	0,0007699	0,003182	0,005747
384	0,0015166	0,006476	0,011674
768	0,00296	0,012749	0,023393
1536	0,00586	0,025404	0,0473
3072	0,01175	0,0496	0,0977
6144	0,02389	0,0992	0,1881
12288	0,04725	OVERFLOW	0,3753
24576	0,09443	OVERFLOW	0,7499
49152	0,18808	OVERFLOW	1,5232
98304	0,37758	OVERFLOW	3,0049

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

For Fibonacci, the algorithms are of decreasing efficiency: the first three are O(n), whereas the last one is O(1.6^n). The VectorSum algorithms, on the other hand, are all linear, with a clear preference for the first one. The second one might be better than the third one if it didn't overflow, but alas, I guess we'll never know.