Algorithmics	Student information	Date	Number of session
	UO: 300829	27/02/2025	3
	Surname: Cid Lazcano		
	Name: Izan		

Activity 1. Divide and conquer by substraction

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.

- **Substraction1:** In the first case the times are so fast that we can't see the growth, I modified the class to be able to pass it arguments as we did in session1.1 and session1.2 and measure lower times. Then, we can see that the results match the theoretical complexity (O(n)) as they grow by a factor of n when the problem size is increased by n.

- **Substraction2:** We can clearly see how the times follow a complexity of O(n^2) as the time values get multiplied by 4 when increasing the size of the problem by 2.

- **Substraction3:** Finally, with substraction3 it is noticeable how the times increase much faster than in the two previous cases accordingly with what we should expect from the O(2^n) complexity.

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?
Both classes will stop giving times for relatively small values of n. This happens because of the important use of stack memory which causes stack overflows.

How many years would it take to complete the Subtraction3 execution for n=80? Reason the answer.
Due to the function's exponential complexity, the execution times grow so fast that they become impossible to work with very soon.

Algorithmics	Student information	Date	Number of session
	UO: 300829	27/02/2025	3
	Surname: Cid Lazcano		
	Name: Izan		

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).

n	Substraction4
100	LoR
200	LoR
400	200
800	1326
1600	9444
3200	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).

n	Substraction5
30	815
32	2361
34	6521
36	20001
38	57707
40	OoT

How many years would it take to complete the Substraction5 execution for n=80? Reason the answer.

Again, the times would be untreatable for this exponential complexity and the times would also depend on the hardware it's executed on.

Algorithmics	Student information	Date	Number of session
	UO: 300829	27/02/2025	3
	Surname: Cid Lazcano		
	Name: Izan		

Activity 3. Divide and conquer 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.

- **Division1:** In the first case, just like it happened in Substraction1, the times are so fast that we can't see the growth, again, I modified the class to be able to pass it arguments as we did before. Then, we can see that the results match the theoretical complexity (O(n)) as they grow by a factor of n when the problem size is increased by n.

- **Division2:** Since we can appreciate that the times measured double when n doubles and the ratio **Execution Time / n log(n)** stays more or less constant we can say that the times match the complexity of the algorithm.

- **Division3:** Finally, for Division3 we can check if the times match the expected just by taking a look at the ratio **Execution Time / n** and since it remains nearly constant we can state that they do match the complexity.

Implement a Division4.java class with a complexity O(n2) (with abk) 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(n2) (with a>bk) and then fill in a table showing the time (in milliseconds) for n=1000, 2000, 4000, 8000, … (up to OoT).

n	Division4
1000	LoR
2000	LoR
4000	187
8000	654
16000	2397
32000	11090

n	Division5
1000	LoR
2000	213
4000	771
8000	2762
16000	10943
32000	43545

Algorithmics	Student information	Date	Number of session
	UO: 300829	27/02/2025	3
	Surname: Cid Lazcano		
	Name: Izan		

Activity 4. Two basic examples

After analysing 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.

- **Fibonacci (Iterative and Recursive1):** The fib1, fib2, and fib3 methods perform a fixed amount of work per step, so as n increases, the time grows linearly (O(n)).
- **Fibonacci (Recursive2):** The fib4 method makes two recursive calls per call, causing an exponential growth in execution time (approximately O(2^n)).
- **Vector Sum:** All three methods in VectorSum1 (sum1, sum2, and sum3) process each element once, so their execution time grows in direct proportion to n (O(n)).

n	sum1	sum2	sum3
1000	LoR	40*10^-3	58*10^-3
2000	LoR	80*10^-3	122*10^-3
4000	LoR	190*10^-3	291*10^-3
8000	57*10^-3	317*10^-3	476*10^-3
16000	129*10^-3	Overflow	962*10^-3
32000	261*10^-3	Overflow	1872*10^-3
64000	478*10^-3	Overflow	3713*10^-3
128000	912*10^-3	Overflow	7297*10^-3
256000	1810*10^-3	Overflow	14865*10^-3
512000	3645*10^-3	Overflow	29292*10^-3
1024000	7527*10^-3	Overflow	OoT
2048000	14869*10^-3	Overflow	OoT
4096000	29716*10^-3	Overflow	OoT

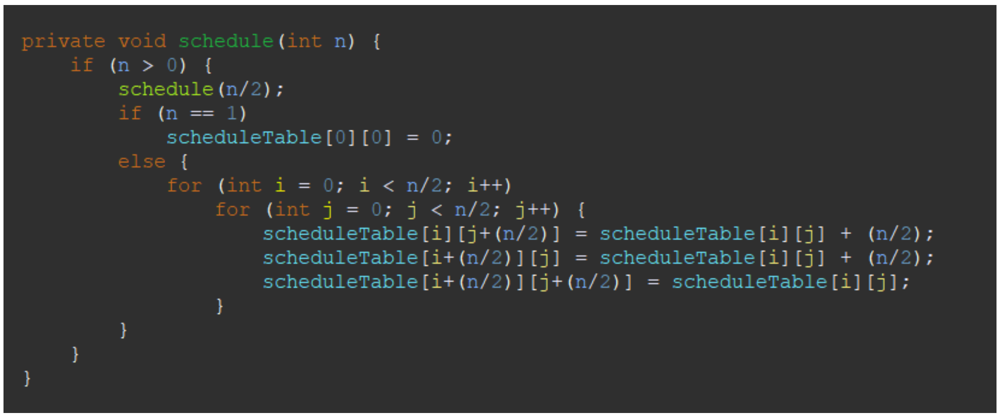
n	fib1	fib2	fib3	fib4
1000	LoR	LoR	33*10^-3	OoT
2000	LoR	LoR	66*10^-3	OoT
4000	LoR	50*10^-3	144*10^-3	OoT
8000	73*10^-3	106*10^-3	330*10^-3	OoT
16000	166*10^-3	247*10^-3	Overflow	OoT
32000	316*10^-3	427*10^-3	Overflow	OoT
64000	558*10^-3	947*10^-3	Overflow	OoT
128000	1108*10^-3	2093*10^-3	Overflow	OoT
256000	2183*10^-3	4266*10^-3	Overflow	OoT
512000	4432*10^-3	12209*10^-3	Overflow	OoT
1024000	8935*10^-3	24791*10^-3	Overflow	OoT
2048000	21105*10^-3	OoT	Overflow	OoT
4096000	OoT	OoT	Overflow	OoT

Algorithmics	Student information	Date	Number of session
	UO: 300829	27/02/2025	3
	Surname: Cid Lazcano		
	Name: Izan		

Activity 3. Petanque championship organization

What complexity does the algorithm created have? It is very important to know it to be sure that our solution is the best possible.

The algorithm splits the problem in half each time, calling itself with n/2. At each level, two nested loops process a part of the matrix, taking O(n²) time.



```
private void schedule(int n) {
    if (n > 0) {
        schedule(n/2);
        if (n == 1)
            scheduleTable[0][0] = 0;
        else {
            for (int i = 0; i < n/2; i++)
                for (int j = 0; j < n/2; j++) {
                    scheduleTable[i][j+(n/2)] = scheduleTable[i][j] + (n/2);
                    scheduleTable[i+(n/2)][j] = scheduleTable[i][j] + (n/2);
                    scheduleTable[i+(n/2)][j+(n/2)] = scheduleTable[i][j];
                }
        }
    }
}
```

We are asked to design a way to verify the time that our algorithm would take to classify the participants starting from a sample size n = 2 until the memory of the computer we have "hold". Obviously, we must measure the times for each size without considering text outputs per console that distort the results.

n	T Calendar
2	LoR
4	LoR
8	LoR
16	LoR
32	LoR
64	LoR
128	LoR
256	LoR
512	205
1024	788
2048	2961
4096	13671
8192	50938

Check whether the times obtained in the previous section meet or not the theoretical complexity of the designed algorithm in the first step.

They do match the theoretical complexity O(n^2), we can see how the values grow by a factor of 4 when the size of the problem is increased by 2.