Algorithmics	Student information	Date	Number of session
	UO: 295180	17/02/24	1
	Surname: Orviz Viesca		
	Name: Mario		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. Measuring Execution Times I

Calculate how many more years we can continue using the way of counting. Knowing that:

- *System.currentTimeMillis()* measures the time in a type long variable
- It measures the time from the first of January of 1970
- Long has **64 bits.**

Using java, we can obtain the maximum date that the max value of long corresponds to with:

```
long max = Long.MAX_VALUE;
Date d = new Date(max);
System.out.println(d); // Maximum date reachable -> Sun Aug 17 08:12:55 CET
292278994
System.out.println(d.year - 2024) // Years left -> 292275070
```

And we obtain:
Sun Aug 17 08:12:55 CET 292278994

Then, that's the maximum date, so from now we can just subtract 2024
Then we have ***292275070*** years left

Activity 2. Measuring Execution Times II

- What does it mean that the time measured is 0?

  When the time measured is 0, it means that the execution time is so low that we cant have any relevant measures of the algorithm. Then, we'll need to increase n in order to get some relevant information

- From what size of problem (n) do we start to get reliable times?

  To get a reliable time, we need to obtain a time bigger than 50 ms. For example, for the n = 10000000, the algorithm wastes 83ms

Algorithmics	Student information	Date	Number of session
	UO: 295180	17/02/24	1
	Surname: Orviz Viesca		
	Name: Mario		

Activity 3. Taking Small Execution Times (<50ms)

- What happens with the time if the problem size is multiplied by 2? What happens with the time if the problem size is multiplied by a value k other than 2? (try it, for example, for k=3 and k=4 and check the times obtained)

  -> The time will increase in proportion to the factor applied to it, but it will still have the same complexity.

- Explain whether the times obtained are those expected from the linear complexity O(n)

From what we saw in Vector4.java measuring the times for sum, create the following three java classes:

- **Vector5.java** to measure times for **maximum**.
- **Vector6.java** to measure times for **matches1**.
- **Vector7.java** to measure times for **matches2**.

The data of this new classes measures are in the table below. Those measures for Tmatches1 are mostly Oot (Out of time), because they exceed 60 seconds. That happens because the algorithm used in this case has a $O(n^2)$ complexity, so its executions time grows heavily. It can be seen that the lowest time value for Tmatches1 is greater than the biggest one of Tsum.

n	Tsum	Tmaximum	Tmatches1	Tmatches2
10000	0.115 ms	0.119 ms	725 ms	0.142 ms
20000	0.282 ms	0.326 ms	3443 ms	0.312 ms
40000	0.53 ms	0.492 ms	11791 ms	0.627 ms
80000	1.07 ms	1.23 ms	44737 ms	1.12 ms
160000	2.14 ms	2.48 ms	187085 ms	2.23 ms
320000	4.28 ms	4.11 ms	Oot	3.7 ms
640000	8.6 ms	10.4 ms	Oot	10.1 ms
1280000	16.6 ms	21 ms	Oot	19 ms
2560000	32.3 ms	42.2 ms	Oot	26.3 ms
5120000	59 ms	83.7 ms	Oot	48.1 ms
10240000	93 ms	131.8 ms	Oot	135 ms
20480000	203 ms	241.1 ms	Oot	234 ms
40960000	267 ms	468 ms	Oot	438 ms
81920000	707 ms	916.4 ms	Oot	896 ms

Algorithmics	Student information	Date	Number of session
	UO: 295180	17/02/24	1
	Surname: Orviz Viesca		
	Name: Mario		

The previous measurements were taken in the class computer, with a processor Intel® Core™ I7-4790 CPU @ 3.60 GHz and memory 8,0 GB DDR3.