


Algorithmics	Student information	Date	Number of session
	UO: 270534	2/10/2021	2
	Surname: Iglesias Préstamo	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Ángel		



Activity 1. [Measuring execution times]

- **How many years can we continue using this way of counting?** According to the Java Documentation, the return type of the execution of the *currentTimeMillis()* method is *long*. This means, the maximum number (positive) to be represented is 9,223,372,036,854,775,807; as we the size of the long type is 8 *bytes* = 8 * 8 *bit* = 64 *bit* → $2^{64} = 18,446,744,073,709,551,616$; from now on we will call this number N. You may say: OK, all you've done is fine, but that's not the number I'm asking for; true. Remember long numbers are represented in two's complement, what leads us to the fact that, indeed, N is the range of different values that we can represent. In order us to get the upper bound, is as easy as using its formula: $2^{N-1} - 1 = 9,223,372,036,854,775,807$. If we perform some mathematical conversions, but first, finding the difference between the highest value and the current – when writing this script – one: we will get the number of years we can keep on using this way of counting. What's interesting here is the fact that there's a quite easy solution for us to solve this problem. By running the following code (Java):

```
System.out.println(new Date(Long.MAX_VALUE)).
```

Which will return:

```
Sun Aug 17 17:12:55 EST 292278999.
```

As you can notice, this way, we can see that in 292 million years this way of counting will eventually overflow.

- **What does it mean that the time measured is 0?** I understand the computer is so fast that the value of milliseconds is too low: under 1 millisecond; this is, the precision of the *currentTimeMillis()* method in Java is higher than the actual value measured.
- **From what size of problem (n) do we start to get reliable times?** In my case, with my computer, I must give as a parameter a value of n around 140,000,000.

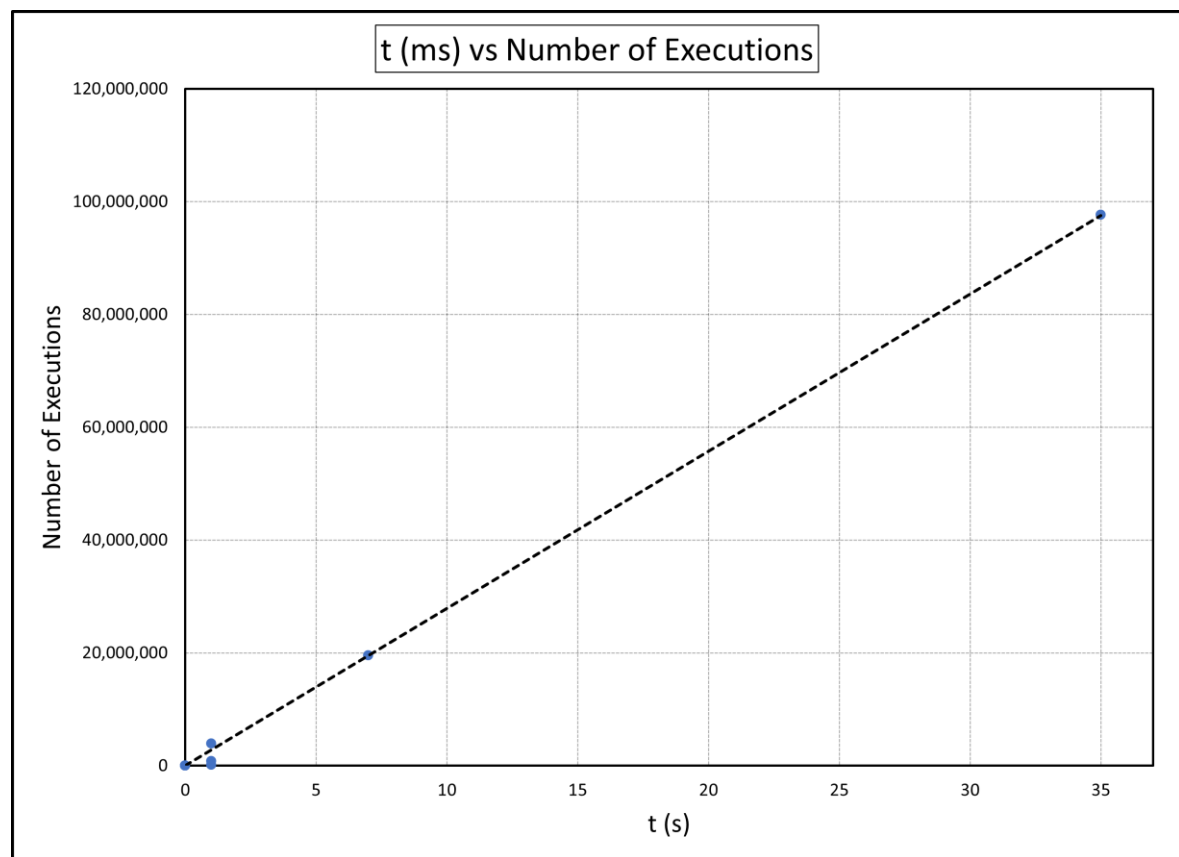
Activity 2. [Grow of the problem size]

Task 1.

- What happens with time if the size of the problem is multiplied by 5? It also gets incremented by 5.

$$f(n) = O(n^1) \rightarrow t_2 = k^1 \cdot t_1 = k \cdot t_1 = 5 \cdot t_1$$

- Are the times obtained those that were expected from linear complexity $O(n)$? Yes, they are. As shown in the previous equation. Notice that k is the factor by which the program is incremented; in the previous example: 5.
- Use a spreadsheet to draw a graph with Excel. On the X axis we can put the time and on the Y axis the size of the problem.



Task 2. Even if this task is not requested, I obtained the times for Vector3.java.

<i>n</i>	<i>fillIn(t)</i>	<i>sum(t)</i>	<i>maximum(t)</i>
10	0	2	0
30	0	0	0
90	0	0	0

<i>n</i>	<i>fillIn(t)</i>	<i>sum(t)</i>	<i>maximum(t)</i>
270	0	1	0
810	0	0	0
2,430	0	0	0
7,290	0	0	0
21,870	0	1	0
65,610	0	2	0
196,830	1	3	1
590,490	0	6	0
1,771,470	1	18	0
5,314,410	2	54	2
15,943,230	6	156	5
47,829,690	16	468	17
143,489,070	49	1,374	51
430,467,210	148	4,051	146

* Notice that sometimes, you may find values that are greater than their successors, for example, you may see that for a value of *n* equal to 196,830 the execution time is 1 millisecond; however, for a value of *n* equal to 590,490, the execution time is 0 milliseconds. That may be caused by the fact that those measurements are still not so low that the results are not conclusive, reasonable. Those may be caused by the garbage collector being run during one of the executions, or a process being executed at the background: too many reasons for us to consider those acceptable. More in more, the documentation of Java states that there may be some troubles related to the ratio at which the OS updates the time of the machine.

Activity 3. [Taking small execution times]

Task 1.

<i>n</i>	<i>fillIn(t)</i>	<i>sum(t)</i>	<i>maximum(t)</i>
10	22	4	6
30	37	1	4
90	97	3	5
270	260	8	11
810	795	26	37
2,430	2,383	155	92
7,290	7,024	472	229
21,870	20,933	1,410	701
65,610	65,778	4,245	2,133
196,830	196,432	12,732	6,291
590,490	599,752	38,408	18,481
1,771,470	1,830,533	117,018	73,118

n	$fillIn(t)$	$sum(t)$	$maximum(t)$
5,314,410	5,591,880	350,667	238,733
15,943,230	16,567,089	1,050,549	809,650
47,829,690	49,672,445	3,151,879	2,546,780
143,489,070	147,989,765	9,454,941	7,056,789
430,467,210	444,819,520	28,768,704	21,678,452

* All the results are in hundreds of micros => nTimes = 100,000. Notice that some of the measurements aren't well given as the time for us to execute the program was going to be huge, indeed, it was.

Task 2.

- **What are the main components of the computer in which you did the work (process, memory)?** Mainly memory and CPU. According to the values I got looking at the task manager – during a test execution, not the one in the script. Both were over 60% of usage; when the program finished its execution, everything went back to idle.
- **Do the values obtained meet the expectations? For that, you should calculate and indicate the theoretical values (a couple of examples per column) of the time complexity. Briefly explain the results.** Notice that each, and every time, I'm going to be using the following expression, according to the complexity of this algorithm:

$$f(n) = O(n^c) \rightarrow t_2 = k^c \cdot t_1$$

$$f(n) = O(n^1) \rightarrow t_2 = k^1 \cdot t_1 = k \cdot t_1 \text{ (in this case)}$$

$fillIn(t)$	$sum(t)$	$maximum(t)$
$t_1 = 97, n_1 = 90 \text{ and } n_2 = 21,870$ $t_3 = 260, n_3 = 270 \text{ and } n_4 = 7,290$ <ul style="list-style-type: none"> $t_2 = \left(\frac{21,870}{90}\right) \cdot 97 = 23,571$ $t_4 = \left(\frac{7,290}{270}\right) \cdot 260 = 7,020$ 	$t_1 = 472, n_1 = 7,290 \text{ and } n_2 = 196,830$ $t_3 = 155, n_3 = 2,430 \text{ and } n_4 = 7,290$ <ul style="list-style-type: none"> $t_2 = \left(\frac{196,830}{7,290}\right) \cdot 472 = 12,755$ $t_4 = \left(\frac{7,290}{2,430}\right) \cdot 155 = 465$ 	$t_1 = 229, n_1 = 7,290 \text{ and } n_2 = 196,830$ $t_3 = 11, n_3 = 270 \text{ and } n_4 = 7,290$ <ul style="list-style-type: none"> $t_2 = \left(\frac{196,830}{7,290}\right) \cdot 229 = 6,183$ $t_4 = \left(\frac{7,290}{270}\right) \cdot 11 = 297$

Values	$fillIn(t)$	$sum(t)$	$maximum(t)$
Actual ₁	20,933	12,732	6,291
Expected ₁	23,571	12,755	6,183
Actual ₂	7,024	472	229
Expected ₂	7,020	465	297

Activity 4. [Operations on matrices]

Task 1.

n	$sumDiagonal1(t)$	$sumDiagonal2(t)$
10	13	6
30	35	3
90	269	6
270	2,175	26
810	16,775	109
2,430	144,071	2,781
7,290	1,386,453	12,988
21,870	12,477,890	74,886

* Notice that results are in hundreds of micros, as I want this to be executed in a reasonable time. I get an error related to the fact that java is unable to allocate more memory for the array.

Task 2.

- **What are the main components of the computer in which you did the work (process, memory)?** As before, Memory and processor.
- **Do the values obtained meet the expectations? For that, you should calculate and indicate the theoretical values (a couple of examples per column) of the time complexity. Briefly explain the results.** I have probably some mistake regarding the theoretical values of the $sumDiagonal2(t)$.

$sumDiagonal1(t)$	$sumDiagonal2(t)$
$t_1 = 16,775, n_1 = 810 \text{ and } n_2 = 7,290$ $t_3 = 144,071, n_3 = 2,430 \text{ and } n_4 = 7,290$	$t_1 = 109, n_1 = 810 \text{ and } n_2 = 21,870$ $t_3 = 26, n_3 = 270 \text{ and } n_4 = 7,290$
<ul style="list-style-type: none"> • $t_2 = \left(\frac{7,290}{810}\right)^2 \cdot 16,775 = 1,358,755$ • $t_4 = \left(\frac{7,290}{2,430}\right)^2 \cdot 144,071 = 1,296,639$ 	<ul style="list-style-type: none"> • $t_2 = \left(\frac{21,870}{810}\right) \cdot 109 = 2,943$ • $t_4 = \left(\frac{7,290}{270}\right) \cdot 26 = 702$

Values	$sumDiagonal1(t)$	$sumDiagonal2(t)$
Actual ₁	1,386,453	1,386,453
Expected ₁	1,358,755	1,296,639
Actual ₂	74,886	12,988

Values	<i>sumDiagonal1(t)</i>	<i>sumDiagonal2(t)</i>
Expected ₂	2,943	702

Activity 5. [Benchmarking]

- **Why you get differences in execution time between the two programs?** Even if Python 3 does not perform badly, it's clearly proved that it takes interpreted languages more time to run than those that are compiled, as Java.
- **Regardless of the specific times, is there any analogy in the behavior of the two implementations?** Those functions from Python are translated into methods in the world of Java. The rest of the code is just the whole same thing but in one case using Python's syntax, and on the other, using Java's one.