Algorithmics	Student information	Date	Number of session
	UO: 283428		1.1
	Surname: Triana Fernández		
	Name: Carlos		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. Measuring execution times

1. how many more years can we continue using this way of counting?

Since the method currentTimeMillis() returns the current millisecond stored in a long type variable, we can know how many years can be stored in a whole long number.

The long data type is a 64-bit two's complement integer, so the greatest positive value stored is 2^63 -1. As 2^63 milliseconds are 292 277 266 years, if we subtract the years have already passed, we can continue counting 292.277.214 years more.

2. What does it mean that the time measured is 0?

Although there may not have passed a millisecond, there is not enough precision to say how much time has passed. When the elapsed time is that low, internal process from the system might have interfered with the process we wan to measure.

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

With a value of n = 100000000 we obtain a time of 48 seconds, enough to be reliable.

Activity 2. Grow of the problem size

1. What happens with time if the size of the problem is multiplied by 5?

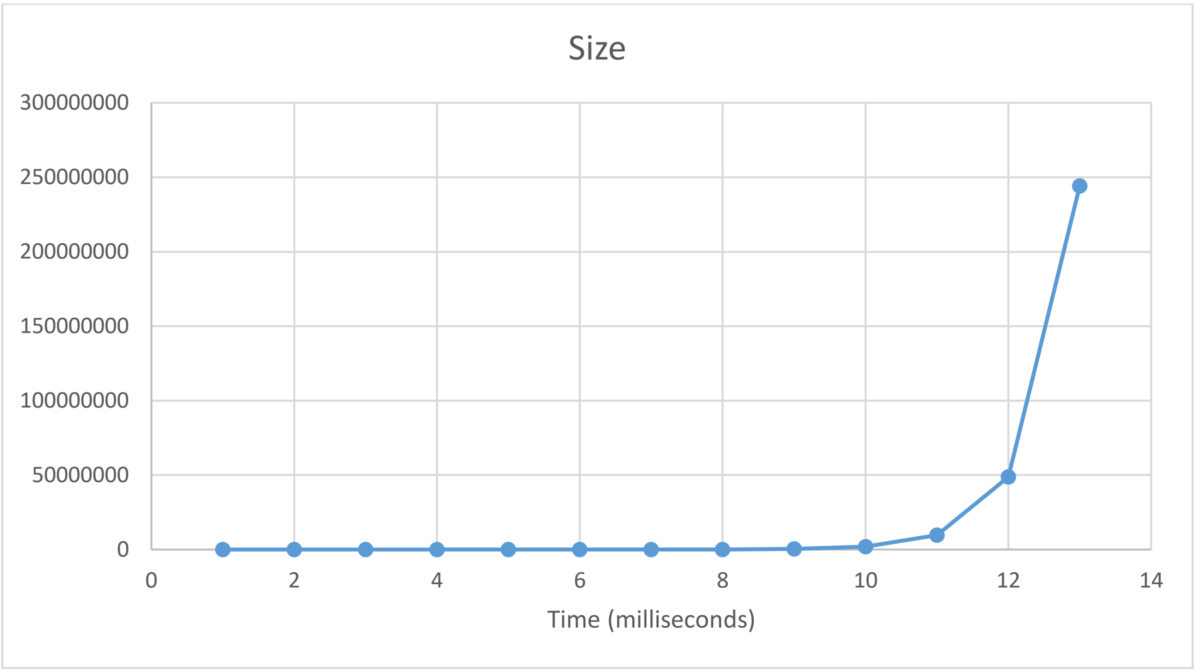
As the complexity of this method is linear(n), the complexity will be multiplied by 5 each iteration. In the same way, the time of execution should be multiplied by 5. We can observe how the times evolve: 3, 18, 92; each one are the result of the increasing complexity: 5 times 3 is 18 and 5 times 18 it is 90.

Algorithmics	Student information	Date	Number of session
	UO: 283428		1.1
	Surname: Triana Fernández		
	Name: Carlos		

2. Are the times obtained those that were expected from linear complexity O(n)?

Although the results are as expected, they are too low to be reliable. They are as expected because the times follow the progression of a linear complexity.

3. 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.



Algorithmics	Student information	Date	Number of session
	UO: 283428		1.1
	Surname: Triana Fernández		
	Name: Carlos		

Activity 3 & 4. Taking small execution times

n	fillin(t)	sum(t)	maximun(t)	sumDiagonal1(t)	sumDiagonal2(t)
10	0	0	0	0	0
30	0	0	0	0	0
90	0,1	0	0,1	0	0
270	0,3	0	0	0,1	0
810	1,1	0,2	0,2	0,4	0
2430	8,4	0,5	0,6	1	0
7290	67,7	1,6	1,9	4	0,1
21870	597,8	15,1	12,5	7	0,3
65610	5458	136,1	103,8	13	2
196830	48573,5	1177,7	981,4	108	2
590490	nan	10885,4	9590,4	nan	nan

Measurements taken on: Intel(R) Core(TM) i7-1065G7 CPU @ 1.30GHz   1.50 GHz
16,0 GB (15,6 GB usable)

1. What are the main components of the computer in which you did the work (process, memory)?

The methods “fillin”, “sum” and “maximum”, performs simple operations on vectors, therefore the CPU is not under heavy work during the execution of the algorithm. The read and write operations of the methods are the most important.

Thus, the method sum has a longer execution time than maximum because has more write operations.

Algorithmics	Student information	Date	Number of session
	UO: 283428		1.1
	Surname: Triana Fernández		
	Name: Carlos		

2. 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.

The time complexity of the methods sum and maximum is linear. However, the function "fillin" is quadratic because we have to take in mind the complexity O(n) from the method "nextInt()".

"Fillin": If runtime for n = 7290 is 67.7 the following time should be 3^2 times the previous one, 603 which is close to 597,8. In the same sense, the next value 5.373 is closed to 5458.

"Maximum": The complexity of the method is O(2n + 2) in the best case and O(5n + 2) in the worst case.

"Sum": its complexity Is always the same, O(3n + 2).

The complexities of the "maximum" and "sum" functions vary from the ones of "fillin" method. The "fillin" method grows exponentially as n^2, and the methods "sum" and "maximum" as, approximately, n*3. In the particular case of n being 3 we can see similarities in the progression as n^2 will be the same as n*3.

Activity 4. Operations on matrices

1. What are the main components of the computer in which you did the work (process, memory)?

As in the previous methods, the memory.

2. 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.

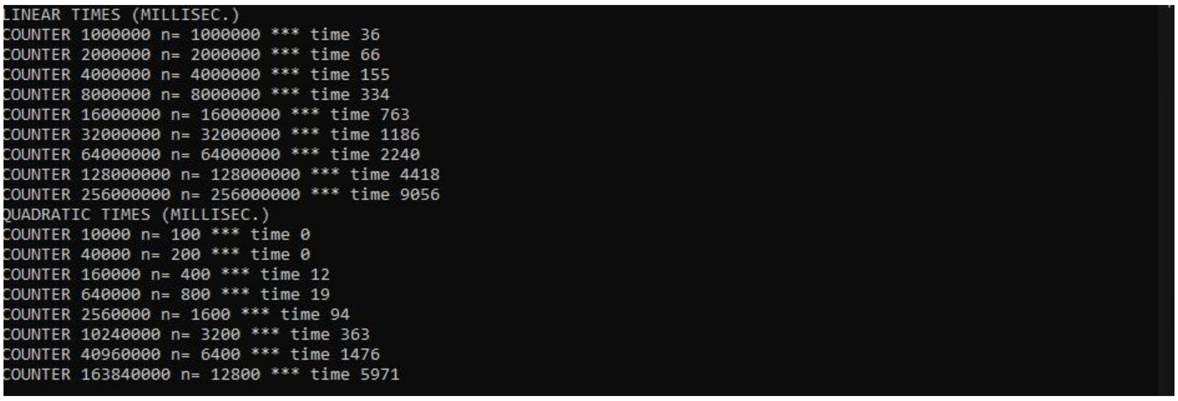
The values obtained follow the sequence of a quadratic and a linear function.

	Student information	Date	Number of session
Algorithmics	UO: 283428		1.1
	Surname: Triana Fernández		
	Name: Carlos		

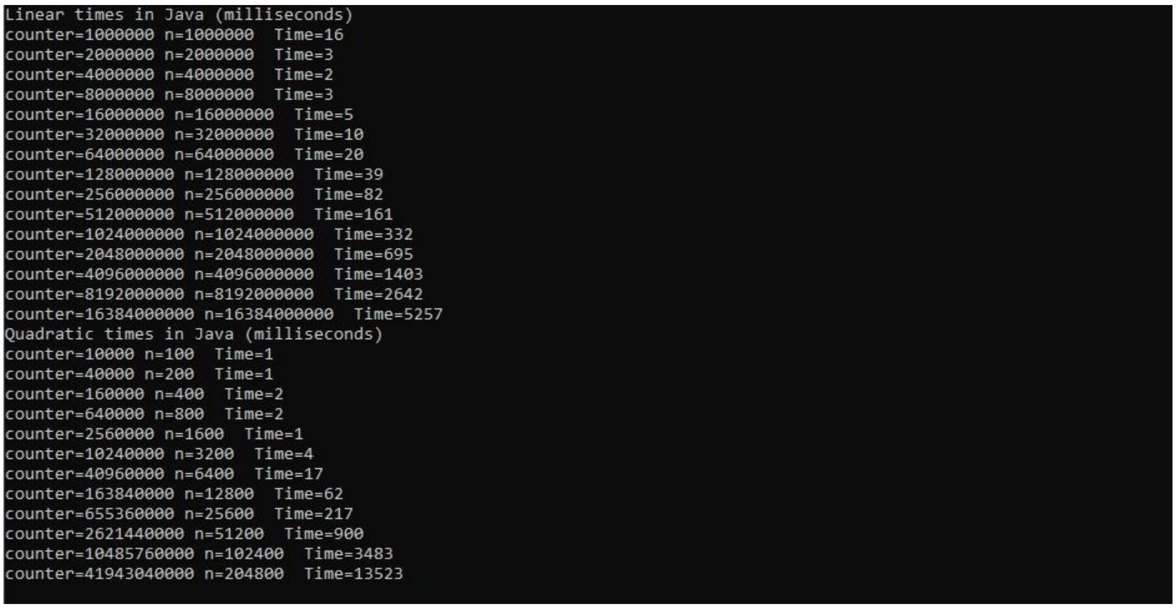
Although the results are not reliable enough, we can see how the progression is different between the two functions.

sumDiagonal1: execution time when n = 65610 is 13, the following time should be around 117, and it is 108.

Activity 5. Benchmarking



```
LINEAR TIMES (MILLISEC.)
COUNTER 1000000 n= 1000000 *** time 36
COUNTER 2000000 n= 2000000 *** time 66
COUNTER 4000000 n= 4000000 *** time 155
COUNTER 8000000 n= 8000000 *** time 334
COUNTER 16000000 n= 16000000 *** time 763
COUNTER 32000000 n= 32000000 *** time 1186
COUNTER 64000000 n= 64000000 *** time 2240
COUNTER 128000000 n= 128000000 *** time 4418
COUNTER 256000000 n= 256000000 *** time 9056
QUADRATIC TIMES (MILLISEC.)
COUNTER 10000 n= 100 *** time 0
COUNTER 40000 n= 200 *** time 0
COUNTER 160000 n= 400 *** time 12
COUNTER 640000 n= 800 *** time 19
COUNTER 2560000 n= 1600 *** time 94
COUNTER 10240000 n= 3200 *** time 363
COUNTER 40960000 n= 6400 *** time 1476
COUNTER 163840000 n= 12800 *** time 5971
```



```
Linear times in Java (milliseconds)
counter=1000000 n=1000000  Time=16
counter=2000000 n=2000000  Time=3
counter=4000000 n=4000000  Time=2
counter=8000000 n=8000000  Time=3
counter=16000000 n=16000000  Time=5
counter=32000000 n=32000000  Time=10
counter=64000000 n=64000000  Time=20
counter=128000000 n=128000000  Time=39
counter=256000000 n=256000000  Time=82
counter=512000000 n=512000000  Time=161
counter=1024000000 n=1024000000  Time=332
counter=2048000000 n=2048000000  Time=695
counter=4096000000 n=4096000000  Time=1403
counter=8192000000 n=8192000000  Time=2642
counter=16384000000 n=16384000000  Time=5257
Quadratic times in Java (milliseconds)
counter=10000 n=100  Time=1
counter=40000 n=200  Time=1
counter=160000 n=400  Time=2
counter=640000 n=800  Time=2
counter=2560000 n=1600  Time=1
counter=10240000 n=3200  Time=4
counter=40960000 n=6400  Time=17
counter=163840000 n=12800  Time=62
counter=655360000 n=25600  Time=217
counter=2621440000 n=51200  Time=900
counter=10485760000 n=102400  Time=3483
counter=41943040000 n=204800  Time=13523
```

1. Why you get differences in execution time between the two programs?

Algorithmics	Student information	Date	Number of session
	UO: 283428		1.1
	Surname: Triana Fernández		
	Name: Carlos		

The main difference between both languages, Java and Python, is that Java is a compiled language that, usually, ensures a better performance than Python. The pre-compilation of the Java code avoids wasting execution time.

This differences allows the execution of the program during more repetitions than Python.

2. Regardless of the specific times, is there any analogy in the behavior of the two implementations?

As both implementations have same complexities, they have a similar progression on their results.

The first program has a linear complexity, it is seen in the evolution of the elapsed time; each time is approximately two times the previous one.

The complexity of the second program is quadratic. The evolution is two square times the previous time. O(n^2) means that the variation of n will also be affected to the square of the complexity: n = 2 O(n^2), n = 2n O(2n^2) = 2^2 * O(n^2);