


Algorithmics	Student information	Date	Number of session
	UO: 296503	15/02/2025	1.1
	Surname: Mulet	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Alonso		



Activity 1. Measuring execution times

How many years can we continue using `System.currentTimeMillis()`

The return type of this method is long, that means that we have 64bits to represent it.

$$2^{64} - 1 = 9.223.372.036.854.775.807$$

This is the maximum number of ms we can represent. At the moment I'm writing this document, the return of `System.currentTimeMillis()` is 1.739.208.680.863.

Then by a simple calculation we can get the available ms we still have until this approach stops working: $9.223.372.036.854.775.807 - 1.739.208.680.863 = 1.844674233 \cdot 10^{19}$ ms

We have to convert these milliseconds into years, I have used [this web](#) and got the following result: 584.553.993,9979 years

Vector 2

Our problem complexity is $O(n)$, first it fills an array of a size passed as argument by the user and then adds all the elements in that array.

As it has a good complexity, we need a big problem size to get measuring times.

In my computer, I start getting reliable times with a problem size $\geq 15.000.000$

Algorithmics	Student information	Date	Number of session
	UO: 296503	15/02/2025	1.1
	Surname: Mulet		
	Name: Alonso		

Activity 2. Taking small execution times

What happens with time if the problem size is multiplied by 2?

With a number of 1.000.000 repetitions (note that we have to divide the time by the number of repetitions, that is I am working with nano seconds, which are 10^{-6} milliseconds) I got the following results:

n	time(ns)
10	88
50	270
250	1085
1250	5196
6250	26139
31250	OoT
156250	OoT
781250	OoT
3906250	OoT

If the problem size is multiplied by two:

n	times(ns)
10	81
20	123
40	211
80	366
160	701
320	1345
640	2598
1280	5157
2560	10490

Algorithmics	Student information	Date	Number of session
	UO: 296503	15/02/2025	1.1
	Surname: Mulet		
	Name: Alonso		

5120	21621
10240	41763

Times follow the Linear complexity, everytime n grows as 2, time grows more or less by 2 also. For example, $n=20$ $t=123$ // $n=40$ $t=211$ which is almost the double for the time.

What happens with 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)

Increased by 3:

n	time(ns)
10	85
30	161
90	402
270	1109
810	3175
2430	9570
7290	28876
21870	OoT

Increased by 4:

n	time(ns)
10	82
40	200
160	679
640	2525
2560	10057
10240	40619

Algorithmics	Student information	Date	Number of session
	UO: 296503	15/02/2025	1.1
	Surname: Mulet		
	Name: Alonso		

40960	OoT
-------	-----

We can see how the times grow linearly following the complexity.

For instance, in the three problem size with $n = 30$ we get 161 as time and then $n = 90$ gets a time of 402, which is a number close to 161×3 . Same for all the next numbers and same for $n = 4$, we can easily see the linear growth.

Activity 3. Vector4 vs Vector5

Now I am working with a number of 1000 repetitions (microseconds).

n	Tsum	Tmaximum
10000	38	61
20000	77	126
40000	153	232
80000	314	473
160000	628	936
320000	1260	1876
640000	2519	3809
1280000	5127	7706
2560000	10301	15433
5120000	21414	31810
10240000	43747	OoT
20480000	OoT	OoT
40960000	OoT	OoT
81920000	OoT	OoT

Algorithmics	Student information	Date	Number of session
	UO: 296503	15/02/2025	1.1
	Surname: Mulet		
	Name: Alonso		

Both algorithms have the same complexity $O(n)$, the problem with maximum is that we actually need to fill another array of the same length so it takes a bit more of time.

For $n=1000$ is not a problem, but there is a significant increase in the time (38 for sum and 61 for maximum) so by increasing the size, even if they have the same complexity, maximum times get bigger.

Activity 4. Vector6 vs Vector7

Again, I am still using 1000 repetitions, so microseconds are kept as time unit.

n	Tmatches1	Tmatches2
10000	OoT	58
20000	OoT	115
40000	OoT	240
80000	OoT	477
160000	OoT	961
320000	OoT	1941
640000	OoT	3854
1280000	OoT	7729
2560000	OoT	15851
5120000	OoT	32438
10240000	OoT	OoT
20480000	OoT	OoT
40960000	OoT	OoT
81920000	OoT	OoT

The main difference here is that algorithm matches1 has a $O(n^2)$ complexity while matches2 has linear complexity.

Algorithmics	Student information	Date	Number of session
	UO: 296503	15/02/2025	1.1
	Surname: Mulet		
	Name: Alonso		

To get some reliable times, I have established the number of repetitions just to 1, keep in mind that the unit of time used for this measurement is milliseconds.

n	Tmatches1
10000	518
20000	2081
40000	8349
80000	33714
160000	OoT
320000	OoT
640000	OoT
1280000	OoT
2560000	OoT
5120000	OoT
10240000	OoT
20480000	OoT
40960000	OoT
81920000	OoT

And even by working with just one repetition, times are still much slower. This shows how important complexity is. Times reflect the growth, for instance with $n = 10000$ we get a time of 518 milliseconds, by increasing the size by two ($n = 20000$) the time increased more or less by 4 (2^2) getting a time of 2081 milliseconds.

If we had used the same repetitions as in matches2, the time for $n = 80000$ would have been approximately 33714000 microseconds, which is 70679 times bigger than matches2.