Algorithmics	Student information	Date	Number of session
	UO: 300829	13/02/2025	1
	Surname: Cid Lazcano		
	Name: Izan		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. Measuring execution times – Question 1

Calculate how many more years we can continue using this way of counting (currentTimeMillis()). Explain what you did to calculate it.

Since the method returns the time in milliseconds from January 1st, 1970, as an integer of type long, the max value in milliseconds that it could return is $2^{63} - 1 = 9.22 \times 10^{18}$. So, what we must do to get how much time this method will be working from now on:

Max.Time – current time = time remaining

The current time in milliseconds is approximately 1.73×10^{12}, so we would still get 9.22×10^{18} milliseconds remaining (292,364,282 years approx.).

Activity 2. Measuring execution times – Question 2

Why does the measured time sometimes come out as 0?

Because for small input sizes the execution is so fast that the difference between the second currentTimeMillis() and the first is less that 1 millisecond, so the execution time is lower than that.

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

This would depend on the hardware of the computer used to perform the operation but, in the one used during this session, I started getting times over 1ms from size = 100,000. However, we should also take into account the time used by internal processes of the system. So, if we take the threshold of 50ms, int this computer I start getting reliable times at size = 10,000,000.

Activity 2. Taking small execution times (<50ms)

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

The execution time should also double when n is doubled to match the expected O(n) complexity.

Algorithmics	Student information	Date	Number of session
	UO: 300829	13/02/2025	1
	Surname: Cid Lazcano		
	Name: Izan		

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)

If n increases by a factor of k, then the execution times should also increase by the same factor. The time scales linearly with n.

Table1 (Times in milliseconds without optimization):

n	Tsum	Tmaximum
10000	0,072	0,104
20000	0,167	0,226
40000	0,292	0,431
80000	0,638	0,829
160000	1,144	1,669
320000	2,341	3,275
640000	4,567	6,594
1280000	9,044	13,271
2560000	18,281	28,064
5120000	36,693	77,7
10240000	73,8	109,1
20480000	145,8	212,2
40960000	286,2	429,6
81920000	584,7	954,4

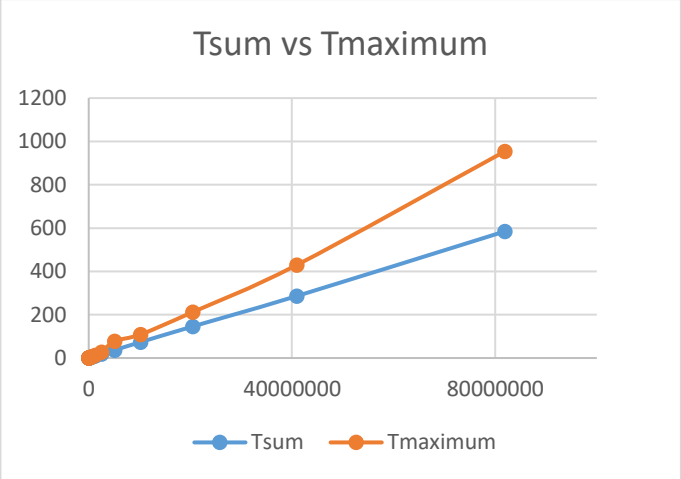
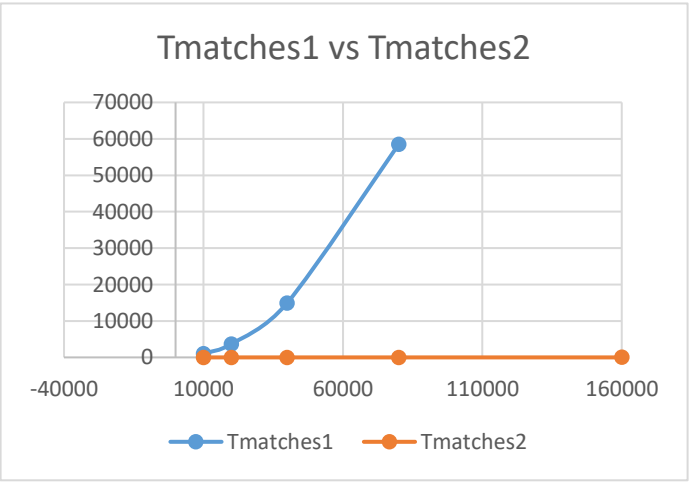


Table2 (Times in milliseconds without optimization):

n	Tmatches1	Tmatches2
10000	995	0,112
20000	3678	0,252
40000	14918	0,452
80000	58485	0,883
160000	OoT	1,853
320000	OoT	3,404
640000	OoT	6,799
1280000	OoT	13,796
2560000	OoT	27,604
5120000	OoT	56,33
10240000	OoT	113,1
20480000	OoT	225,1
40960000	OoT	439,6
81920000	OoT	897,8



Algorithmics	Student information	Date	Number of session
	UO: 300829	13/02/2025	1
	Surname: Cid Lazcano		
	Name: Izan		

- **Computer specs:**
  - 13th Gen Intel(R) Core(TM) i7 1360P 2.20 GHz
  - 16GB Ram

Conclusions:

Tsum and Tmaximum O(n) complexities are confirmed by how the times values double as n doubles, showing a linear growth. We have some variations derived from subprocesses, memory management, etc.

Tmatches1 clearly shows a quadratic growth that gets to Out of Time values pretty early, confirming its $O(n^2)$ complexity.
Tmatches2 shows a linear growth just like Tsum and Tmaximum did before, we can clearly see how the execution times double as the size of the problem does.