Algorithmics	Student information	Date	Number of session
	UO: 300092		
	Surname: Herce Campiña		
	Name: Andrés		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. Measuring execution times

Java's currentTimeMillis() returns a 64-bit signed integer, which can hold values up to 2^{63} -1 or roughly 9.22 x 10^{18} milliseconds. Since there are about 3.16 x 10^{10} milliseconds in a year, this counter will continue to work for approximately 292 million years from January 1, 1970, before it overflows.

Activity 2. Measuring thresholds

Sometimes, the measured time comes out as 0 because the operation finishes so quickly that it doesn't even take a full millisecond. This happens when the task is too small to be captured by the clock's resolution. We start getting reliable times when the problem size (n) is large enough so that the operation takes over 50 milliseconds.

Activity 3. Measuring times with different algorithms

Specifications of the computer: 13th Gen Inter(R) Core i5-1335U, 16 GB RAM

Vector 4

nSize	Time(ms)
10000	46 * 10^{-3}
20000	76 * 10^{-3}
40000	153 * 10^{-3}
80000	296 * 10^{-3}
16000	608 * 10^{-3}
32000	1193 * 10^{-3}
64000	2401 * 10^{-3}
1280000	4812 * 10^{-3}

2560000	9953 * 10^{-3}
5120000	20589 * 10^{-3}
10240000	39977 * 10^{-2}
20480000	7804 * 10^{-2}
40960000	15663* 10^{-2}
81920000	31813 * 10^{-2}

Vector 5

nSize	Time(ms)
10000	58 * 10^{-4}
20000	112 * 10^{-4}
40000	217 * 10^{-4}
80000	434 * 10^{-4}
16000	902 * 10^{-4}
32000	1789 * 10^{-4}
64000	3461 * 10^{-4}
1280000	6986 * 10^{-4}
2560000	14254 * 10^{-4}
5120000	28687 * 10^{-4}
10240000	5615* 10^{-3}
20480000	11381 * 10^{-3}
40960000	23173* 10^{-3}
81920000	46956 * 10^{-3}

Vector 6

Algorithmics	Student information	Date	Number of session
	UO: 300092		
	Surname: Herce Campiña		
	Name: Andrés		

nSize	Time(ms)
10000	50 * 10^{-3}
20000	76 * 10^{-3}
40000	153 * 10^{-3}
80000	296 * 10^{-3}
16000	608 * 10^{-3}
32000	1193 * 10^{-3}
64000	2401 * 10^{-3}
1280000	4812 * 10^{-3}
2560000	9953 * 10^{-3}
5120000	20589 * 10^{-3}
10240000	39977 * 10^{-2}
20480000	7804 * 10^{-2}
40960000	15663* 10^{-2}
81920000	31813 * 10^{-2}

Matches1

nSize	Time(ms)
10000	51 * 10^{-1}
20000	207 * 10^{-1}
40000	822 * 10^{-1}
80000	3241 * 10^{-1}
16000	12703 * 10^{-1}
32000	51447 * 10^{-1}

64000	20088 * 10^{-1}
1280000	OoT
2560000	OoT
5120000	OoT
10240000	OoT
20480000	OoT
40960000	OoT
81920000	OoT

Matches2

nSize	Time(ms)
10000	58 * 10^{-4}
20000	117 * 10^{-4}
40000	238 * 10^{-4}
80000	463 * 10^{-4}
16000	938 * 10^{-4}
32000	1894* 10^{-4}
64000	3786 * 10^{-4}
1280000	7617 * 10^{-4}
2560000	15094 * 10^{-4}
5120000	29912 * 10^{-4}
10240000	5755 * 10^{-3}
20480000	11932 * 10^{-3}
40960000	25075 * 10^{-3}
81920000	49099 * 10^{-3}

To sum up, every algorithm meets its expectation.

The complexity of each method aligns with its expected performance. The maximum, matches2, and sum methods run in linear time, making them efficient even for large inputs. In contrast, matches1 runs in quadratic time due to its nested loops, leading to significantly slower execution for large n. As expected, matches2 provides the same result as matches1 but in much less time, demonstrating the inefficiency of redundant iterations. The sum method performs as expected, efficiently computing the sum of all elements in the array in linear time.