Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. [Make a table reflecting the execution times of the PythonA1.py module for the exposed values of n (10000, 20000, 40000, 80000, 160000, 320000, 640000). If for any n it takes more than 60 seconds you can indicate OoT ("Out of Time"), both in this section and in subsequent sections.]

n	time(ms)
10000	1520
20000	6057
40000	24417
80000	Oot
160000	Oot
320000	Oot
640000	Oot

Activity 2. [Make a table that reflects, at least for two computers to which you have access, the execution times of the PythonA1.py module for the exposed values of n (10000,20000, …, 640000).

Clearly indicate which CPU and RAM memory you are using in each test.]

n	Computer 1 time (ms)	Computer 2 time (ms)
10000	1520	3322
20000	6057	26073
40000	24417	Oot
80000	Oot	Oot
160000	Oot	Oot
320000	Oot	Oot
640000	Oot	Oot

Computer 1 : Intel core i5-12600KF, 32GB DDR5

Computer 2: Intel core i5-8300U, 8GB DDR3

Activity 3. [Program a class named JavaA1.java, which uses the same A1 algorithm to find out if a number is a primer number, as in PythonA1.py. Then, a table must be made reflecting the execution times of JavaA1.java for the same values of n (10000, 20000, ..., 640000). Finally, compare these times with those obtained in Python (in a previous section) for that same algorithm A1.]

n	time(ms)
10000	10
20000	29
40000	110
80000	401
160000	1478
320000	5559
640000	20888

Times in Java

n	time(ms)
10000	1520
20000	6057
40000	24417
80000	Oot
160000	Oot
320000	Oot
640000	Oot

Times in python

We can see that Java is much faster than python.

Activity 4. [Make a table reflecting the execution times of the modules PythonA1.py, PythonA2.py and PythonA3.py, for the same values of n (10000, 20000, …, 640000). Codify those same algorithms A2 and A3 in Java, in two classes named respectively JavaA2.java and JavaA3.java. Make a table that reflects the execution times of the classes JavaA1.java, JavaA2.java and JavaA3.java, for the values of n (10000, 20000, ..., 640000) WITHOUT OPTIMIZATION of the Java program. Make a table that reflects the execution times of the classes JavaA1.java, JavaA2.java and JavaA3.java, for the values of n (10000, 20000, ..., 640000) WITH

OPTIMIZATION of the Java program. Finally, draw final conclusions by comparing the times previously obtained: with Python, with Java WITHOUT OPTIMIZATION and with Java WITH OPTIMIZATION. Optionally, an A4 algorithm can be implemented in Java, which should improve the previous algorithms.]

Times for Python A1, A2, A3

n	time A1 (ms)	time A2 (ms)	time A3 (ms)
10000	1520	178	89
20000	6057	684	334
40000	24417	2519	1235
80000	Oot	9282	4568
160000	Oot	35355	17230
320000	Oot	Oot	Oot
640000	Oot	Oot	Oot

Times for Java A1, A2, A3 without optimization:

WITHOUT OPT	n	ms(J1)	ms(J2)	ms(J3)
	10000	35	35	21
	20000	125	125	75
	40000	457	463	271
	80000	1936	1734	1007
	160000	6325	6430	3695
	320000	23957	24256	13871
	640000	Oot	Oot	53150

Times for Java A1, A2, A3 with optimization:

WITH OPT	n	ms(J1)	ms(J2)	ms(J3)
	10000	9	9	5
	20000	28	29	15
	40000	108	106	56
	80000	391	394	196
	160000	1458	1469	716
	320000	5493	5462	2730
	640000	20851	20523	10296

Having studied all the algorithms we can assume that A3 is the best algorithm as it takes the lowest time considering the other two. It works reducing the problem size of the problem to its half, knowing that dividing a number by other one that is higher than its half the resulting quotient will be 0.

From the times taken in Java we can see that both, with optimization and without, times of A1 and A2 are very similar, this is because Java does not take many time taking classes outside the main class.

We can easily appreciate that times with optimization in Java are much faster than without it, with optimization we don´t get any out of time measure. The biggest time we get with optimization in any of the three programs is 20 seconds while in the ones without it are higher than 60 seconds.

In conclusion, Java is much faster than python.