| Algorithmics | Student information | Date | Number of session |
|---|---|---|---|
| | UO: 301022 | 13/02/2025 | 2 |
| | Surname: Canga | | |
| | Name: Martín | | |

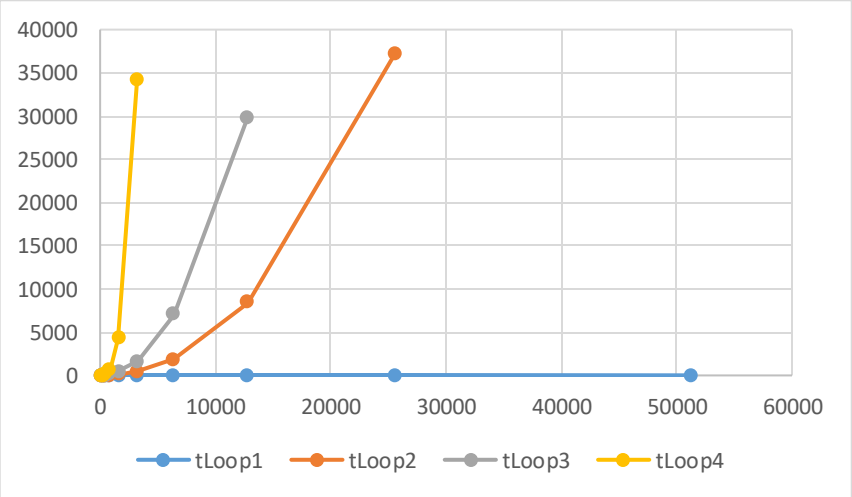
# Activity 1. [Iterative models, different complexity]

Java loops with different complexity

| N | tLoop1 | tLoop2 | tLoop3 | tLoop4 |
|---|---|---|---|---|
| 100 | 0.0088 | 0.37 | 1.28 | 1.38 |
| 200 | 0.019 | 1.23 | 4.88 | 9.49 |
| 400 | 0.042 | 5.64 | 21.65 | 71.75 |
| 800 | 0.0984 | 26.38 | 91.54 | 546.00 |
| 1600 | 0.2094 | 100.40 | 398.00 | 4284.00 |
| 3200 | 0.439 | 466.00 | 1625.00 | 34075.00 |
| 6400 | 0.9368 | 1875.00 | 7022.00 | OoT |
| 12800 | 2.1994 | 8461.00 | 29861.00 | OoT |
| 25600 | 4.991 | 37275.00 | OoT | OoT |
| 51200 | 9.3598 | OoT | OoT | OoT |

All the loops seem to match their respective complexities:

- Loop1 = $O(n \times log(n))$
- Loop2 = $O(n^2 \times log(n))$
- Loop3 = $O(n^2 \times log(n))$
- Loop4 = $O(n^3)$



| Algorithmics | Student information | Date | Number of session |
|---|---|---|---|
| | UO: 301022 | 13/02/2025 | 2 |
| | Surname: Canga | | |
| | Name: Martín | | |

# Activity 2. [Developing algorithms]

Java O(n^2 * log^2(n)), O(n^3 * log(n)) and O(n^4) respective loops without optimization:

| N | tLoop5 | tLoop6 | tLoop7 |
|---|---|---|---|
| 100 | 2.07 | 10.50 | 39.00 |
| 200 | 22.13 | 83.20 | 499.00 |
| 400 | 62.40 | 742.00 | 7182.00 |
| 800 | 218.44 | 6991.00 | OoT |
| 1600 | 1663.95 | OoT | OoT |
| 3200 | 5344.00 | OoT | OoT |
| 6400 | 24054.00 | OoT | OoT |

We can see that the times match their respective growth rate.

# Activity 3. [Comparing algorithms]

**Comparing algorithms without optimization**

| N | tLoop1 | tLoop2 | T1/T2 |
|---|---|---|---|
| 100 | 0.0088 | 0.37 | 0.023 |
| 200 | 0.019 | 1.23 | 0.015 |
| 400 | 0.042 | 5.64 | 0.007 |
| 800 | 0.0984 | 26.38 | 0.003 |
| 1600 | 0.2094 | 100.40 | 0.002 |
| 3200 | 0.439 | 466.00 | 0.0009 |
| 6400 | 0.9368 | 1875.00 | 0.0004 |
| 12800 | 2.1994 | 8461.00 | 0.0002 |

| | | | |
|---|---|---|---|
| 25600 | 4.991 | 37275.00 | 0.0001 |
| 51200 | 9.3598 | OoT | OoT |

If we divide the complexities, we'll get the rate at which T1/T2 will differ

- Loop1 = $O(n \times log(n))$
- Loop2 = $O(n^2 \times log(n))$

In this case is 1/n. Since we know that the closer, we are to zero the better T1 is with respect to T2, the efficiency of T1 will follow a linear trend getting better as n tends to infinity.

| N | tLoop3 | tLoop2 | T3/T2 |
|---|---|---|---|
| 100 | 1.28 | 0.37 | 3.45 |
| 200 | 4.88 | 1.23 | 3.96 |
| 400 | 21.65 | 5.64 | 3.83 |
| 800 | 91.54 | 26.38 | 3.47 |
| 1600 | 398.00 | 100.40 | 3.96 |
| 3200 | 1625.00 | 466.00 | 3.48 |
| 6400 | 7022.00 | 1875.00 | 3.74 |
| 12800 | 29861.00 | 8461.00 | 3.52 |
| 25600 | OoT | 37275.00 | OoT |
| 51200 | OoT | OoT | OoT |

In this case we can see that relationships are always the same as we increase n. This is due to the complexity of the two algorithms being the same. Although both have the same complexity, we can argue that Loop2 is more efficient because it completes any N iteration in a lower time. This is because we are simplifying and the base of the logarithm of Loop2 is greater than Loop3.

| Algorithmics | Student information | Date | Number of session |
|---|---|---|---|
| | UO: 301022 | 13/02/2025 | 2 |
| | Surname: Canga | | |
| | Name: Martín | | |

# Activity 4. [Algorithms in different environments]

| N | tLoop4 (t41) python | tLoop4 (t42) Java NonOpt. | tLoop4 (t43) Java Opt. | t42/t41 | t43/t42 |
|---|---|---|---|---|---|
| 100 | 10 | 1.38 | 0.14 | 0.138 | 0.101 |
| 200 | 64 | 9.49 | 0.23 | 0.148 | 0.024 |
| 400 | 625 | 71.75 | 1.37 | 0.114 | 0.019 |
| 800 | 10212 | 546.00 | 9.42 | 0.05 | 0.017 |
| 1600 | OoT | 4284.00 | 32.36 | OoT | 0.007 |
| 3200 | OoT | 34075.00 | 345.90 | OoT | 0.010 |

We can see that the proportion of each execution is the same for all N values. One thing that stands out is that java is consistently faster than python no matter if this one use optimization techniques or not. Another thing to consider is the optimization of java that changes the bytecode of the original files and optimizes it to reduce potential bottlenecks, that's why we see that difference between the two java executions.