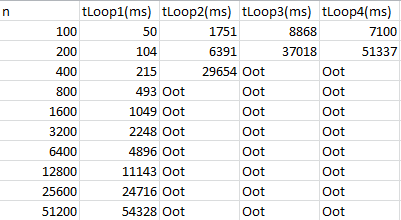


Activity 1. Iterative models



Analyzing the algorithms, we see that:

Loop1 1: O(nlogn)

Loop2: O(n^2logn)

Loop3: O (n^2logn)

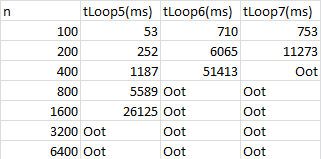
Loop4: O(n^3)

As we can see, loop1 is the fastest algorithm. The second fastest is the loop 2 which, in spite of having the same complexity as the loop3, the algorithm seems to be better designed, obtaining lower times. We can easily appreciate that working with algorithms with high complexities is very impractical in terms of getting good performance.

It is used a size of 100000 to get the times in a computer with a processor i7-12600KF and 32GB DDR5

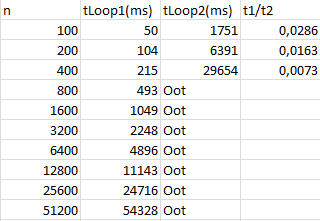
Activity 2. Creation of iterative models of a given time complexity.

The size is adjusted in order to get reliable times, now is 10



As we supposed, loop5 has the worst performance of the new algorithms as has the worst complexity, and even with a size much lower, we are getting too high times after n is 1600, this represents how bad this algorithms performs.

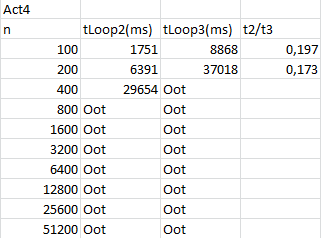
Activity 3. Two algorithms with different complexity.



As the quotient obtained between t1 and t2 obtained is lower than 1 we can appreciate that Loop1 is much better than Loop2, as for the same size argument we are getting Oot times in loop 2 after n = 400 while in the loop 1 all times are less than a minute.

This is expected since the complexity of Loop1 is O(nlogn) while Loop2 is O(n^2logn).

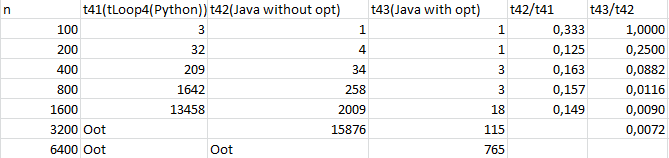
Activity 4. Two algorithms with same complexity.



In spite of having the same complexity, Loop2 seems to be faster, this may be caused by the way we are iterating in the loops, in the loop 2 we are dividing by 3 until we reach n = 0 and in the loop 3 we are multiplying by 2 until we get n, then could be expected that Loop2 works better as it has less iterations.

Activity 5. Same algorithm in different development environments.

In this activity, I used a size of 1 in order to get the same conditions as in the Loop4.py



It is appreciated that python works worse than java (at least for this type of exercise).

In this sample we don´t get a really noticeable difference between the times in Java with optimization and without it because of the size, but we can easily see that as n increases the times without optimization increase faster.

Furthermore, despite the low size, we are getting big times so we have to avoid this types of complexities (O(n^3))