Activity 1. Iterative models

n	tLoop1(ms)	tLoop2(ms)	tLoop3(ms)	tLoop4(ms)
100	50	1751	8868	7100
200	104	6391	37018	51337
400	215	29654	Oot	Oot
800	493	Oot	Oot	Oot
1600	1049	Oot	Oot	Oot
3200	2248	Oot	Oot	Oot
6400	4896	Oot	Oot	Oot
12800	11143	Oot	Oot	Oot
25600	24716	Oot	Oot	Oot
51200	54328	Oot	Oot	Oot

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

n	tLoop5(ms)	tLoop6(ms)	tLoop7(ms)
100	53	710	753
200	252	6065	11273
400	1187	51413	Oot
800	5589	Oot	Oot
1600	26125	Oot	Oot
3200	Oot	Oot	Oot
6400	Oot	Oot	Oot

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.

n	tLoop1(ms)	tLoop2(ms)	t1/t2
100	50	1751	0,0286
200	104	6391	0,0163
400	215	29654	0,0073
800	493	Oot	
1600	1049	Oot	
3200	2248	Oot	
6400	4896	Oot	
12800	11143	Oot	
25600	24716	Oot	
51200	54328	Oot	

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.

Act4			
n	tLoop2(ms)	tLoop3(ms)	t2/t3
100	1751	8868	0,197
200	6391	37018	0,173
400	29654	Oot	
800	Oot	Oot	
1600	Oot	Oot	
3200	Oot	Oot	
6400	Oot	Oot	
12800	Oot	Oot	
25600	Oot	Oot	
51200	Oot	Oot	

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

n	t41(tLoop4(Python))	t42(Java without opt)	t43(Java with opt)	t42/t41	t43/t42
100	3	1	1	0,333	1,0000
200	32	4	1	0,125	0,2500
400	209	34	3	0,163	0,0882
800	1642	258	3	0,157	0,0116
1600	13458	2009	18	0,149	0,0090
3200	Oot	15876	115		0,0072
6400	Oot	Oot	765		

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))