


Algorithmics	Student information	Date	Number of session
	UO: 270534	03/02/2021	5.2
	Surname: Iglesias Préstamo		
	Name: Ángel		



Activity 1. [Counting inversions]

First, what I have to say is that the returning values of the inversions have been changed to the primitive type Long, as we are willing to represent numbers so big that integer type is not enough.

Notice that It has been included a column called Parallel which is the result of performing the mergesort solution, but using the parallel techniques implemented in Java; more in more, I use the Fork/Join Framework. Notice that those results can be easily understood, for values of n so small, the time consumed creating the structure for us to parallelize the code; even if I am not an expert in concurrent programming, create several threads, joining them, and so on, takes a reasonable amount of time: thus, we must tune the THRESHOLD so that for values so small, the execution should be performed sequentially. However, I'm impressed with the time we can save by just using more power from the hardware we have. I believe the parallel version can roughly be improved, we can move to the so loved C++, which may be quicker. To conclude, notice that the parallel algorithm for the highest value is 3,500 times faster.

File	$t\ O(n^2)$	$t\ O(n \log(n))$	Parallel	$t\ O(n^2)/t\ O(n \log(n))$	n inversions	n
Ranking1.txt	92	19	48	4.84	14,074,466	7,500
Ranking2.txt	344	11	15	31.27	56,256,142	15,000
Ranking3.txt	802	13	11	61.69	225,312,650	30,000
Ranking4.txt	3,024	21	13	144.00	903,869,574	60,000
Ranking5.txt	16,080	51	37	315.29	3,613,758,061	120,000
Ranking6.txt	94,070	124	64	758.63	14,444,260,441	240,000
Ranking7.txt	513,207	217	141	2,365.01	57,561,381,803	480,000

Algorithmics	Student information	Date	Number of session
	UO: 270534	03/02/2021	5.2
	Surname: Iglesias Préstamo		
	Name: Ángel		

<i>Brute Force(t)</i>	<i>MergeSort(t)</i>
$t_1 = 344; n_1 = 15,000 \text{ and } n_2 = 120,000$ $t_3 = 3,024; n_3 = 60,000 \text{ and } n_4 = 240,000$ <ul style="list-style-type: none"> $t_2 = \left(\frac{120,000}{15,000}\right)^2 \cdot 344 = 22,016$ $t_4 = \left(\frac{240,000}{60,000}\right)^2 \cdot 3,024 = 48,000$ 	$t_1 = 51; n_1 = 120,000 \text{ and } n_2 = 240,000$ $t_3 = 124; n_3 = 240,000 \text{ and } n_4 = 480,000$ <ul style="list-style-type: none"> $t_2 = \frac{\log k + \log n_2}{\log n_1} \cdot t_1 = \frac{\log_2 2 + \log_2 240,000}{\log_2 120,000} \cdot 51 = 57$ $t_4 = \frac{\log k + \log n_2}{\log n_1} \cdot t_1 = \frac{\log_2 2 + \log_2 480,000}{\log_2 240,000} \cdot 124 = 138$

Values	<i>Brute Force(t)</i>	<i>MergeSort(t)</i>
Actual ₁	16,080	124
Expected ₁	22,016	57
Actual ₂	94,070	217
Expected ₂	48,000	138

I have to say that for me to look for the sizes, I just look at the number of lines of the ranking files. More in more I have to say that it is true that some of the values I got from the improved versions cannot be said to be “right” as they are so small; because I wanted to compare it to the brute force algorithm, as I needed the same units for both of them.

Before doing the calculations, I wasn't pretty sure that the complexities would fit perfectly with the ones I would get, as the code is a bit too complex for it to have complexities a bit different from the approximations. I mean, we consider $O(3 \cdot n)$ to be $O(n)$. And those slight changes can make the expected results to differ. However, it is clear that both algorithms perform in the way they should. The growth rate of the mergesort version outperforms the brute force one. Which was the thing that we were expecting. This is because the brute force implementation's growth rate is increased with the size, not in a linear way, but in some sort of quadratic shape. More in more, as can be proved, by applying the D&C techniques I explained in the previous section, the improved version has

the logarithmic component slowing a bit such rate: $O(n \cdot \log(n))$. Which can be understood as the improved version, way improved indeed, of the quadratic complexity.