


Algorithmics	Student information	Date	Number of session
	UO:269546	24-02-21	2
	Surname: Fernández Arias	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Sara		



## Activity 1. Time measurements for sorting algorithms.

### Insertion measurements

n	sorted(t)	inverse(t)	random(t)
10000	6	489	482
20000	3	1257	1251
40000	0	4472	4352
80000	0	19266	17514
160000	2	76488	69912
320000	4	362058	311626
640000	9	1904140	1278588

Using diffSizes=10 and nTimes=10

Provided that this method has a  $O(n^2)$  complexity for both worst and average cases and a  $O(n)$  for it's best case , the results make sense.

The best case, is when there's nothing to sort. The results obtained are really fast as expected. But not exactly expected linear-wise.

The results obtained for the worst and an average case(random) , are notoriously similar, and also not exactly as expected (but it's possible to perceive a linear tendency).

The demonstration would be:

$$\text{Using } n_2 = k \cdot n_1 \quad k = \frac{n_2}{n_1} \quad T_2 = \frac{f(n_2)}{f(n_1)} \cdot T_1 \quad \text{and provided that } f(n) = o(n^c) \quad T_2 = k^c \cdot T_1$$

Identifying n1 with 10000, and n2 with 2000, and k=2 ,for the different columns:

For the sorted column: being T1=6ms. T2=2^2\*6=24 ms.

For the inverse column: being T1=489ms T2=2^2\*489=1.956ms.

For the random column: T1=482ms T2=2^2\*482=1928ms.

Algorithmics	Student information	Date	Number of session
	UO:269546	24-02-21	2
	Surname: Fernández Arias		
	Name: Sara		

## Selection measurements

n	sorted(t)	inverse(t)	random(t)
10000	65	138	99
20000	182	468	281
40000	641	1378	1094
80000	2530	4523	4155
160000	10381	15709	16984
320000	37612	61967	69657
640000	143987	259956	262991

Using diffSizes=10 and nTimes=10

Provided that this method has a  $O(n^2)$  complexity for all cases: best, worst and average, the result aren't as expected . Having the same complexity, the worst case values are almost the double of the best and average ones .

The demonstration would be:

Using  $n_2 = k \cdot n_1$   $k = \frac{n_2}{n_1}$   $T_2 = \frac{f(n_2)}{f(n_1)} \cdot T_1$  and provided that  $f(n) = o(n^c)$   $T_2 = k^c \cdot T_1$

Identifying  $n_1$  with 10000, and  $n_2$  with 2000, and  $k=2$  ,for the different columns:

For the sorted column: being  $T_1=65$ ms.  $T_2=2^2 \cdot 65=260$  ms.

For the inverse column: being  $T_1=138$ ms  $T_2=2^2 \cdot 138=552$ ms.

For the random column:  $T_1=99$ ms  $T_2=2^2 \cdot 99=396$ ms.

Numerically , is not exactly as expected neither, but the great difference between the worst case and the average an best one is appreciated still.

Algorithmics	Student information	Date	Number of session
	UO:269546	24-02-21	2
	Surname: Fernández Arias		
	Name: Sara		

## Bubble measurements

n	sorted(t)	inverse(t)	random(t)
10000	38	71	157
20000	142	251	626
40000	272	770	2568
80000	2169	5062	10817
160000	11504	21332	46526
320000	41766	87252	194401
640000	179943	345809	832173
1280000	13010533	1381477	

Using diffSizes=7 and nTimes=10

Provided that this method has a  $O(n^2)$  complexity for all cases: best, worst and average, the results aren't so obvious. There's a bit of difference between the columns. It makes sense, since the worst the scenario is, the greater the measures are.

Actually, they are almost doubled:  $38 \cdot 2 = 76 \sim 71$ ;  $76 \cdot 2 = 152 \sim 157$ .

The demonstration would be:

Using  $n_2 = k \cdot n_1$   $k = \frac{n_2}{n_1}$   $T_2 = \frac{f(n_2)}{f(n_1)} \cdot T_1$  and provided that  $f(n) = o(n^c)$   $T_2 = k^c \cdot T_1$

Identifying  $n_1$  with 10000, and  $n_2$  with 20000, and  $k=2$ , for the different columns:

For the sorted column: being  $T_1=38\text{ms}$ .  $T_2=2^2 \cdot 38=152\text{ ms}$ .

For the inverse column: being  $T_1=71\text{ms}$   $T_2=2^2 \cdot 71=284\text{ms}$ .

For the random column:  $T_1=157\text{ms}$   $T_2=2^2 \cdot 157=628\text{ms}$ .

In this case, the **results are as expected!**

Algorithmics	Student information	Date	Number of session
	UO:269546	24-02-21	2
	Surname: Fernández Arias		
	Name: Sara		

### QuickSort measurements

sorted(t)	inverse(t)	random(t)
50	165	183
65	316	364
132	660	659
268	1398	1460
481	3097	3381
1042	6488	6992
2095	13710	13672
4220	27772	28625
8746	58163	61520
18572	120697	128848
39536	244951	267285
80599	510631	551993

Provided that this method has a  $O(n^2)$  for it's worst case, and a  $O(\log n)$  for it's best and average cases.

The demonstration would be:

Using  $n_2 = k \cdot n_1$   $k = \frac{n_2}{n_1}$   $T_2 = \frac{f(n_2)}{f(n_1)} \cdot T_1$  and provided that

$$f = O(\log n) \rightarrow T_2 = \frac{\log k + \log n_1}{\log n_1} \cdot T_1$$

Identifying  $n_1$  with 10000, and  $n_2$  with 2000, and  $k=2$ , for the different columns:

For the sorted column: being  $T_1=50\text{ms}$ .  $T_2=2 \cdot (\log 2 + \log 10000) / \log 10000 \cdot 50 = 107,52\text{ms}$

For the inverse column: being  $T_1=165\text{ms}$ .  $T_2=2 \cdot (\log 2 + \log 10000) / \log 10000 \cdot 165 = 354\text{ms}$

For the random column:  $T_1=$ .  $T_2=2 \cdot (\log 2 + \log 10000) / \log 10000 \cdot 165 = 393,54\text{ms}$

In this case, the **results are as expected!**

Algorithmics	Student information	Date	Number of session
	UO:269546	24-02-21	2
	Surname: Fernández Arias		
	Name: Sara		

## Activity 2. QuicksortFateful.

The pivot is chosen by taking the leftmost element of the vector. The idea will work if the elements to be sorted are closer to the pivot, else, if the array is inversely ordered, the process will be much less efficient.

It will get worse the greater the vector does, since the effort due to the number of comparisons is increased.

If the vector is quite ordered, the number of exchanges might be nice. But the less ordered the vector is, the less efficient this algorithm will be.

That's a bottle neck, since the purpose of this algorithm is the sorting of disordered vectors.