Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersan		

Escuela de Ingeniería Informática
Universidad de Oviedo

Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Activity 1. Bubble algorithm

n	t ordered	t reverse	t random
10000	778	1904	1415
20000	3071	7777	5589
40000	9882	30889	21791
80000	40071	OoT	OoT
160000	OoT	OoT	OoT

The bubble sort algorithm follows an $O(n^2)$ time complexity, although in some cases it can be O(n), like when it uses sentinel. The ordered algorithm fulfils it. Since we are duplicating the size of the program in each measurement, we can see that the results are coming out 4 times the previous result ($2^2 = 4$). The graph on the right shows that they follow the same complexity.

Activity 2. Selection algorithm

n	t ordered	t reverse	t random
10000	698	765	653
20000	2468	2765	2533
40000	9814	10734	10089
80000	39092	43165	41237
160000	OoT	OoT	OoT

The selection algorithm follows an $O(n^2)$ time complexity in all case scenarios. If we check the same way we did for the previous table, we can see that it fits the pattern. The graph on the right proves this information.

Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersan		

Activity 3. Insertion algorithm

n	t ordered	t reverse	t random
10000	LoR	763	398
20000	LoR	3259	1901
40000	LoR	12614	6308
80000	LoR	51381	26355
160000	LoR	OoT	OoT
320000	LoR	OoT	OoT
640000	LoR	OoT	OoT
1280000	LoR	OoT	OoT
2560000	58	OoT	OoT
5120000	109	OoT	OoT
10240000	297	OoT	OoT
20480000	503	OoT	OoT
40960000	910	OoT	OoT
81920000	1893	OoT	OoT

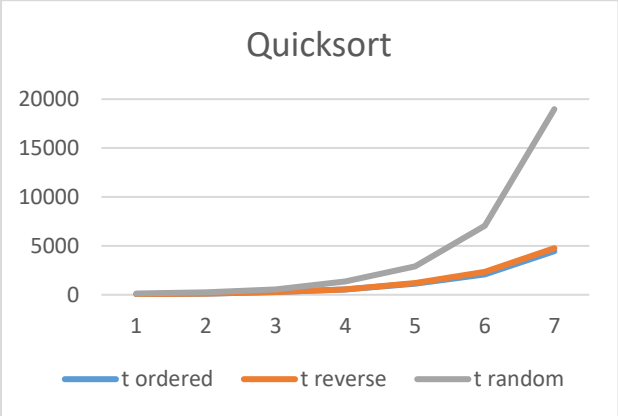
The insertion algorithm also follows an $O(n^2)$ for all case scenarios except for its best-case scenario (the ordered one) which follows an O(n) complexity. In the ordered algorithm we get a lot of low reliability measurements, but as soon as we get better results, we see that it follows its complexity by duplicating the previous result. So, the other algorithms, as we are duplicating the size of the problem, the measurements should be 4 times bigger than the previous one. In the reverse algorithm, there doesn't seem to be any problems. The random algorithm isn't maybe exactly ideal, but it more or less follows the idea.

Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersan		

Activity 4. Quicksort algorithm

n	t ordered	t reverse	t random
250000	65	71	130
500000	130	137	266
1000000	272	282	562
2000000	531	550	1368
4000000	1130	1192	2904
8000000	2091	2350	7046
16000000	4459	4761	18976

Quicksort follows an O(n*log(n)) complexity in its best and average case scenario, which are the cases of the ordered and the reverse. However, in it's worse case scenario, it follows an $O(n^2)$ complexity. This happens when sorting a random list of elements. In the chart shown below, you can see how the random takes much longer than the other 2. You can also see that ordered and reversed are pretty much proportional, due to them having the same time complexity.



Now that we know how long it takes to sort 16 million items initially ordered randomly using the quicksort algorithm, let's have a look at how long it would take when applying the other algorithms that we have been testing. The calculations are made on the following page. As we can see, bubble, selection and insertion are not made to sort such massive amount of items, so for those cases, we will stick with quicksort.

Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersan		

Bubble → Random ($O(n^2)$)

$t_1 = 21791\,ms \quad n_1 = 40000 \quad t_2 = ? \quad n_2 = 16\,000\,000$

$k = \frac{16 \cdot 10^6}{40 \cdot 10^3} = 400 \rightarrow t_2 = k^2 \cdot t_1 = 400^2 \cdot 21791 = 3.48656 \cdot 10^9\,ms$

$3.48656 \cdot 10^9\,ms \cdot \frac{1\,s}{10^3\,ms} \cdot \frac{1\,h}{3600\,s} \cdot \frac{1\,day}{24\,h} = \boxed{40.35\ days}$

Selection → Random ($O(n^2)$)

$t_1 = 41237\,ms \quad n_1 = 80000 \quad t_2 = ? \quad n_2 = 16\,000\,000$

$k = \frac{16 \cdot 10^6}{80 \cdot 10^3} = 200 \rightarrow t_2 = k^2 \cdot t_1 = 200^2 \cdot 41237 = 1.64948 \cdot 10^9\,ms$

$1.64948 \cdot 10^9\,ms \cdot \frac{1\,s}{10^3\,ms} \cdot \frac{1\,h}{3600\,s} \cdot \frac{1\,day}{24\,h} = \boxed{19.09\ days}$

Insertion → Random ($O(n^2)$)

$t_1 = 26355\,ms \quad n_1 = 80000 \quad t_2 = ? \quad n_2 = 16\,000\,000$

$k = \frac{16 \cdot 10^6}{80 \cdot 10^3} = 200 \rightarrow t_2 = k^2 \cdot t_1 = 200^2 \cdot 26355 = 1.0542 \cdot 10^9\,ms$

$1.0542 \cdot 10^9\,ms \cdot \frac{1\,s}{10^3\,ms} \cdot \frac{1\,h}{3600\,s} \cdot \frac{1\,day}{24\,h} = \boxed{12.2\ days}$

Algorithmics	Student information	Date	Number of session
	UO: 293615		
	Surname: Lavelle		
	Name: Gersan		

Activity 5. Quicksort + Insertion algorithm

n	t random
Quicksort	18976
Q+I(k=5)	2405
Q+I(k=10)	2355
Q+I(k=20)	2323
Q+I(k=30)	2360
Q+I(k=50)	2252
Q+I(k=100)	2018
Q+I(k=200)	1650
Q+I(k=500)	1791
Q+I(k=1000)	2444

The first one is the same measurement as before. The idea of this is that if k is greater than the length of the subvector we are sorting, we use insertion, and otherwise we use quicksort. The k will be constant during the whole of the recursive algorithm, so the greater the k is, the sooner we will switch to insertion from quicksort. That explains why k=1000 took a bit longer compared to the previous one, because it spent much more in insertion.