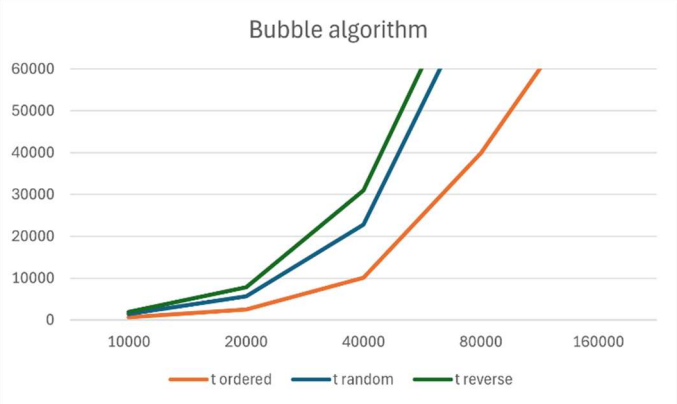
Algorithmics	Student information	Date	Number of session
	UO: 300827	24/02/2025	2
	Surname: Leiras Núñez		
	Name: Sofía		

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	659	1966	1460
2 * 10000	2508	7824	5669
2^2 * 10000	10140	30996	22831
2^3 * 10000	39958	OoT	OoT
2^4 * 10000	OoT	OoT	OoT

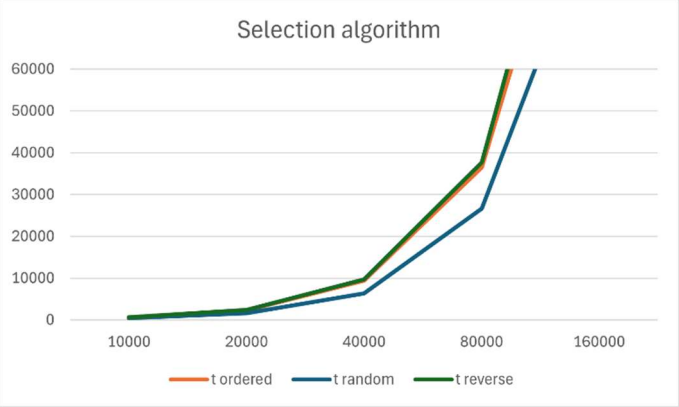


The results align with Bubble Sort's $O(n^2)$ complexity, as execution time grows quadratically with input size. Doubling n results in (more or less) multiplying the time times 2, and larger inputs quickly become infeasible (OoT). The best case is when the vector is sorted and no swaps are needed, with the values of the vector ordered at random, the time is worse because more swaps are needed and, finally, with the reverse order of the values, it takes the most time as every element must be swapped.

Activity 2. Selection algorithm

n	t ordered	t reverse	t random
10000	603	609	440
2 * 10000	2272	2378	1644
2^2 * 10000	9456	9722	6343
2^3 * 10000	36467	37652	26611
2^4 * 10000	OoT	OoT	OoT

Algorithmics	Student information	Date	Number of session
	UO: 300827	24/02/2025	2
	Surname: Leiras Núñez		
	Name: Sofía		

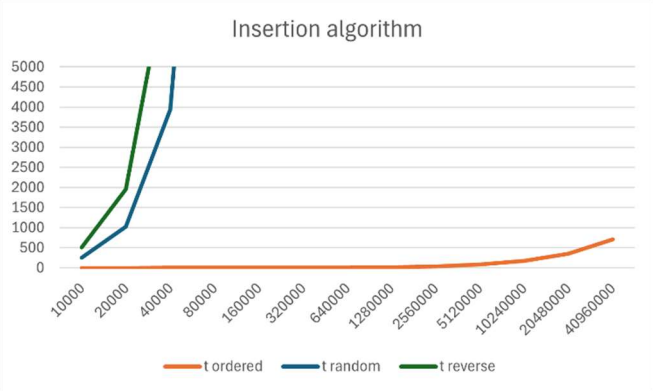


The results align with the expected $O(n^2)$ time complexity of Selection Sort. The execution time grows quadratically as n increases, with each doubling of n leading to approximately 4× the previous time. Unlike Bubble Sort, Selection Sort's performance is independent of initial order, as it always makes the same number of comparisons. This explains why the times for ordered, reverse, and random cases are very similar.

Activity 3. Insertion algorithm

n	t ordered	t reverse	t random
10000	LoR	510	257
2 * 10000	LoR	1958	1027
2^2 * 10000	LoR	7829	3936
2^3 * 10000	LoR	32877	16836
2^4 * 10000	LoR	OoT	OoT
2^5 * 10000	LoR	OoT	OoT
2^6 * 10000	LoR	OoT	OoT
2^7 * 10000	LoR	OoT	OoT
2^8 * 10000	LoR	OoT	OoT
2^9 * 10000	86	OoT	OoT
2^{10} * 10000	172	OoT	OoT
2^{11} * 10000	351	OoT	OoT
2^{12} * 10000	703	OoT	OoT

Algorithmics	Student information	Date	Number of session
	UO: 300827	24/02/2025	2
	Surname: Leiras Núñez		
	Name: Sofía		



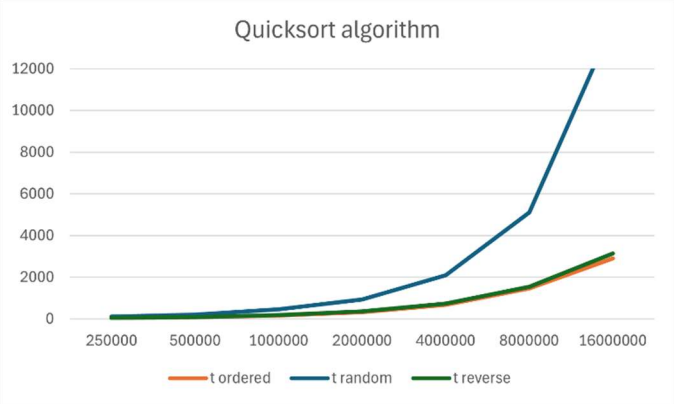
The times obtained align with the expected time complexity of the insertion sort algorithm, which is $O(n^2)$ in the worst case and O(n) in the best case. The ordered case is the best-case scenario for insertion sort, as no swaps are needed and it only makes n-1 comparisons, leading to an O(n) complexity, which is refelected in the results. The reverse order represents the worst-case scenario, as every element must be shifted to the beginning, so the number of comparisons and swaps is larger, leading to $O(n^2)$ complexity. The random inputs represent the average-case complexity, which is $O(n^2)$, but the number of swaps is slightly lower than the previous case.

Activity 4. Quicksort algorithm

n	t ordered	t reverse	t random
250000	LoR	LoR	100
2 * 250000	77	84	201
2^2 * 250000	160	177	461
2^3 * 250000	324	365	926
2^4 * 250000	680	733	2084
2^5 * 250000	1454	1536	5103
2^6 * 250000	2899	3145	14176

The results mostly align with Quicksort's expected O(n log n) time complexity. Thanks to the median-of-three pivot, both ordered and reverse cases avoid the worst-case $O(n^2)$

Algorithmics	Student information	Date	Number of session
	UO: 300827	24/02/2025	2
	Surname: Leiras Núñez		
	Name: Sofía		



behavior and perform similarly (with a O(n log n) complexity). Besides, as expected, the random case grows faster for large values of n as more swaps and partitions are needed.

Calculate how many days would the Bubble, Insertion and Selection algorithms take to sort 16 million items initially in random order:

We will take into account the first value measured for random ordered values in each algorithm, that is n = 10000. So, we have to compare the value 10000 with the value 16000000: 16000000 = 1600 * 10000

Bubble: the Bubble algorithm has a $O(n^2)$ complexity with random ordered values (in fact, for every case), so taking into account that the time with n = 10000 is t = 1460ms, the time for n = 16000000 will be: $1600^2 * 1460 = 3737600000$ms.

$$3737600000 \text{ ms} \cdot \frac{1\ s}{1000\ ms} \cdot \frac{1\ min}{60\ s} \cdot \frac{1\ h}{60\ min} \cdot \frac{1\ day}{24\ h} \approx 43.259\ days$$

Insertion: the Insertion algorithm also has $O(n^2)$ complexity with random ordered values, so taking into account that the time with n = 10000 is t = 257ms, the time for n = 16000000 will be: $1600^2 * 257 = 657920000$ms.

$$657920000 \text{ ms} \cdot \frac{1\ s}{1000\ ms} \cdot \frac{1\ min}{60\ s} \cdot \frac{1\ h}{60\ min} \cdot \frac{1\ day}{24\ h} \approx 7.6148\ days$$

Selection: the Selection algorithm also has $O(n^2)$ complexity with random ordered values, so taking into account that the time with n = 10000 is t = 440ms, the time for n = 16000000 will be: $1600^2 * 440 = 1126400000$ms.

$$1126400000 \text{ ms} \cdot \frac{1\ s}{1000\ ms} \cdot \frac{1\ min}{60\ s} \cdot \frac{1\ h}{60\ min} \cdot \frac{1\ day}{24\ h} \approx 13.037\ days$$

Algorithmics	Student information	Date	Number of session
	UO: 300827	24/02/2025	2
	Surname: Leiras Núñez		
	Name: Sofía		

Activity 5. Quicksort + Insertion algorithm

n	t random
Quicksort	12550
Quicksort + Insertion (k = 5)	13012
Quicksort + Insertion (k = 10)	13636
Quicksort + Insertion (k = 20)	14227
Quicksort + Insertion (k = 30)	13184
Quicksort + Insertion (k = 50)	11088
Quicksort + Insertion (k = 100)	6235
Quicksort + Insertion (k = 200)	3627
Quicksort + Insertion (k = 500)	2055
Quicksort + Insertion (k = 1000)	1893

The table shows that using Quicksort with Insertion Sort works better when the switch to insertion sort happens on larger subarrays (with bigger k). When k is too small, the algorithm switches to insertion sort very often, which is not helpful because insertion sort is only fast when the subarrays are bigger and almost sorted. With small k, the overhead of constantly switching between methods slows everything down. The best results happen with larger k (around 500-1000), where Quicksort handles the big parts and Insertion Sort efficiently finishes the small ones.