


Algorithmics	Student information	Date	Number of session
	UO: 301949	20/2/2025	2
	Surname: Sánchez Menéndez	 Escuela de Ingeniería Informática Universidad de Oviedo	
	Name: Adrián		



Activity 1. Bubble Algorithm

n	t ordered	t reverse	t random
10,000	431	1473	1013
20,000	1707	5798	4013
40,000	6844	22497	15654
80,000	27739	OoT	62211
160,000	OoT	OoT	OoT

The Bubble Algorithm, with complexity $O(n^2)$, works by swapping from the back of the array to the first every pair of elements to move the lowest one to the first position and then to the second, third,...

If the elements are ordered, the algorithm will iterate through the array n times but not perform a single swap, getting the best time.

If the elements are reversed, the algorithm will swap all of them in every single iteration, getting the worst time.

If the elements are random, the algorithm will iterate through the array and swap them whenever the it has to, but as the elements are random, it will happen approximately half of the time, which we can check by looking at the times, as the times it takes to sort the list is more or less in the middle of the best and worst case scenario.

Activity 2. Selection Algorithm

n	t ordered	t reverse	t random
10,000	397	432	394
20,000	1521	1667	1590
40,000	5915	6437	5981

Algorithmics	Student information	Date	Number of session
	UO: 301949	20/2/2025	2
	Surname: Sánchez Menéndez		
	Name: Adrián		

80,000	23723	25792	23882
160,000	OoT	OoT	OoT

The Selection Algorithm, with complexity $O(n^2)$, stores the value of the first element of the array and iterates through the whole array looking for a smaller value. Then, once it has traversed the whole array it will swap the first value and the minimum. It will repeat this process but now starting from the second, third,... elements.

If the elements are ordered, the algorithm will iterate through the array n times but not perform a single swap, getting the best time.

If the elements are reversed, the algorithm will find the best swap which casually allocates the highest value in its place, so it will only perform $n/2$ swaps.

If the elements are random, the algorithm will iterate through the array and swap once every iteration, placing the smaller element in its place each time.

As can be seen, the performances of the three of them are quite similar because there are not many swaps to be performed.

Activity 3. Insertion Algorithm

n	t ordered	t reverse	t random
10,000	LoR	537	289
20,000	LoR	2158	1086
40,000	LoR	8270	4235
80,000	LoR	33614	20111
160,000	LoR	OoT	74876
320,000	LoR	OoT	OoT
640,000	LoR	OoT	OoT
1,280,000	LoR	OoT	OoT
2,560,000	LoR	OoT	OoT

Algorithmics	Student information	Date	Number of session
	UO: 301949	20/2/2025	2
	Surname: Sánchez Menéndez		
	Name: Adrián		

5,120,000	95	OoT	OoT
10,240,000	184	OoT	OoT
20,480,000	377	OoT	OoT
40,960,000	714	OoT	OoT
81,920,000	1514	OoT	OoT

The Insertion Algorithm, with complexity $O(n^2)$, selects as pivot the second element of the list and compares it to the previous ones until it finds one smaller or reaches the beginning of the list. When doing so, if it finds a greater value, it will not swap them, but overwrite the value of the pivot in the list by the greater one. At the end of the iterations, it will rewrite the value from the pivot to the list, essentially creating a smaller but ordered list. The following iterations will take as pivots the third, fourth,... elements, creating small ordered lists of size 3, 4,.. at the beginning of the list.

If the elements are ordered, the algorithm will select the pivot and compare it to the previous one and find that it is already smaller, so it will select a new pivot and so on until the end of the list. This particular case has complexity $O(n)$, because it just iterates once over the whole list.

If the elements are reversed, the algorithm will find that the pivot must be moved to the first position, so it will move all the elements until the pivot one position to the right and then place the pivot in the first position and so on for every iteration, which is the worst performance.

If the elements are random, the algorithm will perform several pivot movements, but it is unlikely that it will do as many as in the reverse ordered list.

Activity 4. Quicksort Algorithm

n	t ordered	t reverse	t random
250,000	LoR	LoR	100

Algorithmics	Student information	Date	Number of session
	UO: 301949	20/2/2025	2
	Surname: Sánchez Menéndez		
	Name: Adrián		

500,000	78	87	206
1,000,000	158	184	442
2,000,000	346	376	960
4,000,000	713	761	2090
8,000,000	1439	1540	4883
16,000,000	2885	3194	12461

The Quicksort Algorithm finds the median of the first, last and center value of the list and uses that value as a pivot to split the list into two lists. These three values will be reordered, having the smallest one in the first position, the highest one in the center and the pivot in the last position. Then, two iterations will start, one from the left and the other from the right. The one from the left will stop whenever it finds an element greater than the pivot and the one from the right, whenever it finds a value smaller than the pivot. When both iterations are stopped, the values will be swapped and the iterations will keep advancing from the positions they were stopped, repeating this process until the iterations cross, in which the pivot will be swapped to that position with the element pointed by the left iterator. This process ensures that the pivot is placed in its position in the list and that all the values to his left are lower than it and the ones on the right are higher.

This process then repeats itself with the two new list until the list has size lower or equal to three, for which the median will order the positions.

Ideally, the median will be a value closer to the median of all the values of the list, which will result in splitting the size of the list to be sorted by 2, hence the complexity $O(n \log n)$. If the elements are ordered, the algorithm will compute the median, allocate the pivot in the last position and iterate through the whole list until it reaches the middle which will swap the pivot and the highest value, leaving the list as it was, but splitting it half to keep performing Quicksort.

If the elements are reversed, the algorithm will compute the median, allocate the pivot in the last position and iterate through the whole list swapping each one of the elements and,

Algorithmics	Student information	Date	Number of session
	UO: 301949	20/2/2025	2
	Surname: Sánchez Menéndez		
	Name: Adrián		

by chance, placing them in its final position, leaving the list ordered but still having to apply Quicksort to the two smaller lists.

If the elements are random, the algorithm will act normally, performing iterations as it is supposed to do not having a particular case with more or less operations than normal.

For a size of 16,000,000 the following algorithms will require:

Bubble Algorithm:

$$\frac{62211 \text{ ms}}{80,000 \text{ size}} \times k = 20 \times \frac{1 \text{ s}}{1,000 \text{ ms}} \times \frac{1 \text{ min}}{60 \text{ s}} \times \frac{1 \text{ h}}{60 \text{ min}} \times \frac{1 \text{ d}}{24 \text{ h}} = 0.288 \text{ d}$$

Selection Algorithm:

$$\frac{23882 \text{ ms}}{80,000 \text{ size}} \times k = 20 \times \frac{1 \text{ s}}{1,000 \text{ ms}} \times \frac{1 \text{ min}}{60 \text{ s}} \times \frac{1 \text{ h}}{60 \text{ min}} \times \frac{1 \text{ d}}{24 \text{ h}} = 0.111 \text{ d}$$

Insertion Algorithm

$$\frac{74876 \text{ ms}}{160,000 \text{ size}} \times k = 10 \times \frac{1 \text{ s}}{1,000 \text{ ms}} \times \frac{1 \text{ min}}{60 \text{ s}} \times \frac{1 \text{ h}}{60 \text{ min}} \times \frac{1 \text{ d}}{24 \text{ h}} = 0.087 \text{ d}$$

Activity 5. Quicksort + Insertion

n	t random
Quicksort	12461
Quicksort + Insertion (k = 5)	702
Quicksort + Insertion (k = 10)	661
Quicksort + Insertion (k = 20)	583
Quicksort + Insertion (k = 30)	592
Quicksort + Insertion (k = 50)	625
Quicksort + Insertion (k = 100)	733

Algorithmics	Student information	Date	Number of session
	UO: 301949	20/2/2025	2
	Surname: Sánchez Menéndez		
	Name: Adrián		

Quicksort + Insertion (k = 200)	1019
Quicksort + Insertion (k = 500)	1920
Quicksort + Insertion (k = 1,000)	3471

From this table we can conclude that Insertion algorithm greatly improves the performance of the Quicksort algorithm but the size of the vector for the Insertion is also important, because having a vector too small would nearly affect the original Quicksort algorithm and having a vector too big would reduce the performance of the Quicksort algorithm. The optimal size for the vector is around size 20.