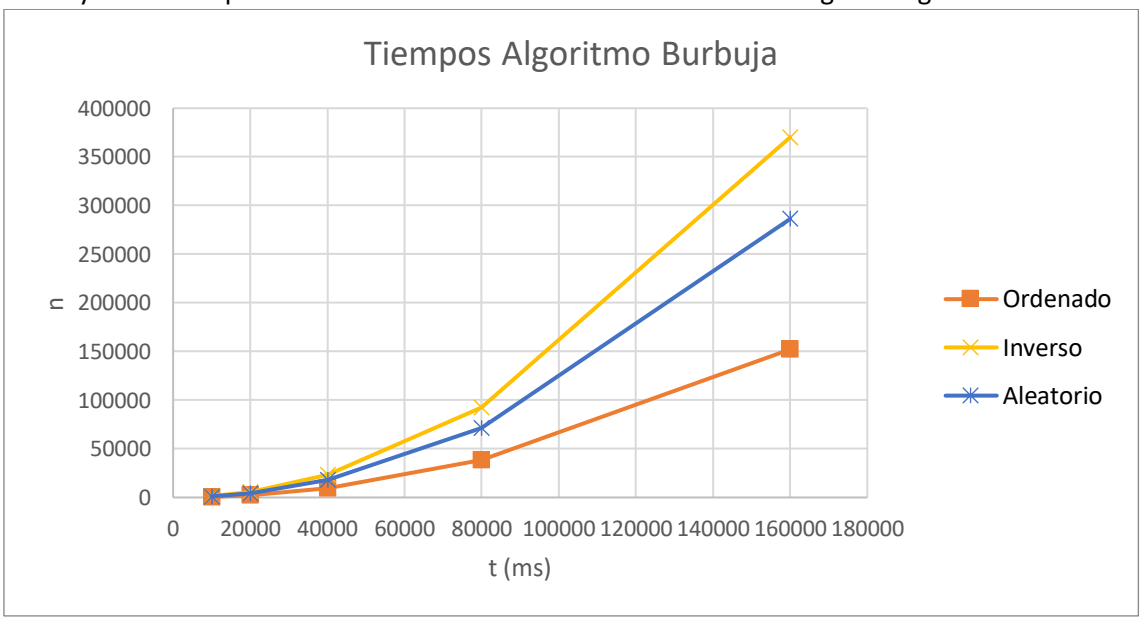
PRÁCTICA 2 ALGORITMIA

Durante esta práctica se abordará el problema de ordenación, es decir, dados n elementos en cualquier orden aplicarles un algoritmo que consiga ordenarlos por una clave predeterminada. En este caso particular, los elementos serán enteros, pero hay que tener en cuenta que podría ser aplicable sobre cualquier tipo de dato que permita una distinción de orden.

1. A continuación, se van a tomar los tiempos del algoritmo de ordenación denominado **burbuja**. Este algoritmo tiene una complejidad o(n^2), por lo que se esperan tiempos bastante grandes.
   La siguiente tabla representa los tiempos obtenidos:

n	t ordenado	t inverso	t aleatorio
10000	602	1468	1142
20000	2378	5776	4494
40000	9634	23156	18052
80000	38369	92620	71521
160000	152142	370159	286242

   Para ayudar a comprender las diferencias entre ellos se muestra la siguiente gráfica:



   Se puede ver que cuando los elementos ya están ordenados el tiempo de ejecución del algoritmo es menor, pero la complejidad es la misma que en los otros dos casos, O(n^2). Se puede comprobar en el crecimiento, pues las tres crecen igual de rápido. También se puede observar que tarda más en aplicarse sobre un conjunto de elementos en orden inverso. Por tanto, todos los tiempos concuerdan con lo esperado de este algoritmo.

2. Ahora se tomarán los tiempos del algoritmo de ordenación denominado **selección**. Este algoritmo tiene una complejidad O(n^2) en todos los casos, por lo que se esperan tiempos bastante grandes.
   La siguiente tabla representa los tiempos obtenidos:

n	t ordenado	t inverso	t aleatorio
10000	437	449	419
20000	1858	1750	1600
40000	6947	7100	6578
80000	38135	34625	26841
160000	148290	114304	108729

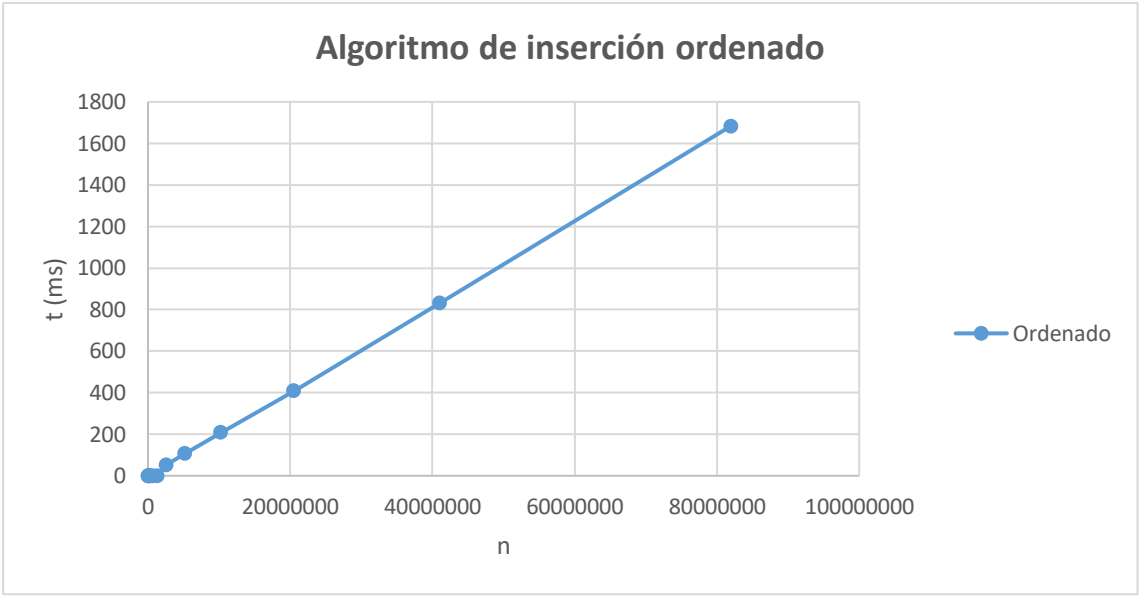
Y se muestra la siguiente gráfica para ayudar a entender los datos obtenidos:

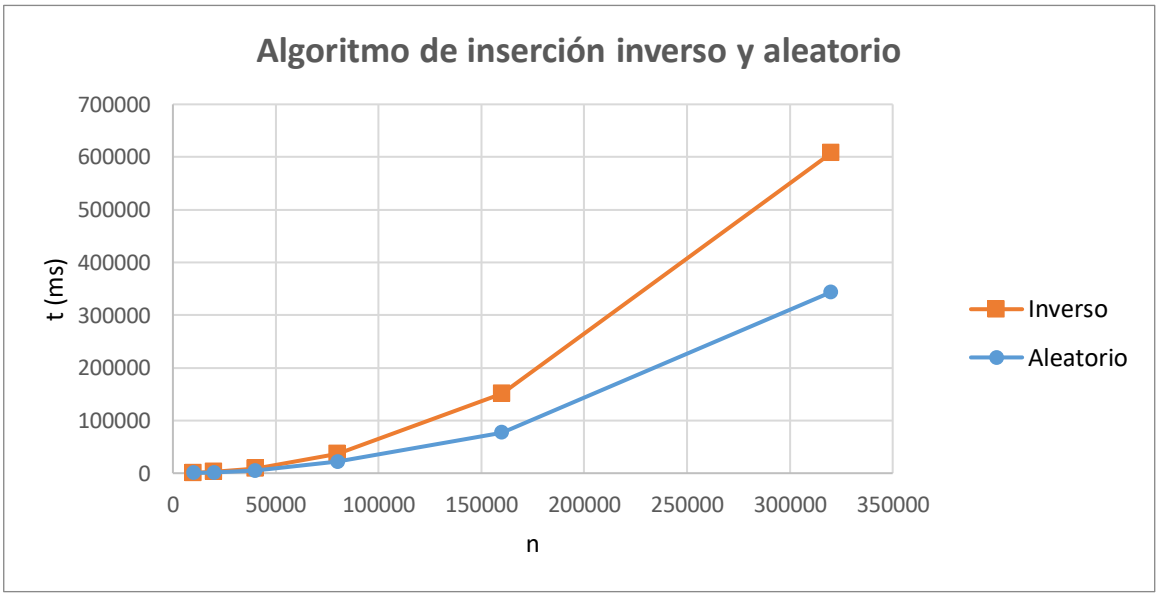
Se puede ver que los tiempos son algo menores que los obtenidos con el algoritmo anterior, pero la gráfica crece en el mismo orden, pues las dos tienen complejidad $O(n^2)$. Se puede ver que, en cuanto a tiempos de ejecución, son menores cuando los elementos están en orden aleatorio. En definitiva, los tiempos concuerdan con lo esperado.

3. Para continuar con la práctica, se tomarán los tiempos del algoritmo de ordenación denominado **inserción**. Este algoritmo tiene una complejidad $O(n^2)$ de forma general, pero en el caso de que los elementos ya estén ordenados el algoritmo tendrá una complejidad lineal, $O(n)$.
   Esto se comprueba con los siguientes datos:

n	t ordenado	t inverso	t aleatorio
10000	FdF	597	313
20000	FdF	2407	1150
40000	FdF	9158	4816
80000	FdF	36796	21631
160000	FdF	151288	77051
320000	FdF	607222	343531
640000	FdF	FdT	FdT
1280000	FdF	FdT	FdT
2560000	52	FdT	FdT
5120000	105	FdT	FdT
10240000	208	FdT	FdT
20480000	407	FdT	FdT
40960000	830	FdT	FdT
81920000	1683	FdT	FdT

En este caso, una gráfica con todos los casos en ella no representa bien la información, pues la diferencia de complejidad entre el caso en que los elementos estén ordenados y los demás ocasiona que las gráficas sean muy diferentes. Por esto, se representa una gráfica para el primer caso y otra gráfica para los dos siguientes:

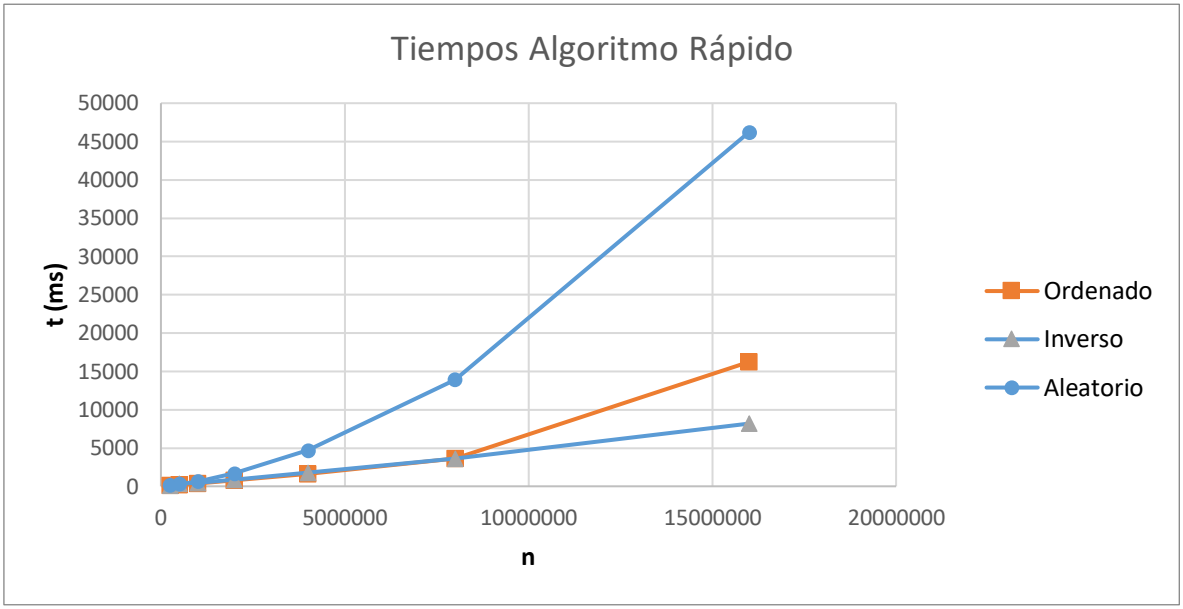




Como se puede ver, la primera gráfica es lineal, mientras que en la segunda las funciones crecen según su complejidad cuadrática. Por tanto, los datos son los esperados.

4. El último algoritmo del que vamos a medir tiempos es el algoritmo de ordenación denominado **rápido** (**Quicksort**). De este algoritmo se prevén tiempos bastante mejores que en los anteriores, debido a que tiene una complejidad O(n log n). Sin embargo, en el caso de que los elementos estén ordenados y se escoja como pivote el primer elemento, la complejidad será $O(n^2)$. Lo mismo ocurre para el caso análogo.

n	t ordenado	t inverso	t aleatorio
250000	101	138	140
500000	186	341	299
1000000	405	473	644
2000000	796	830	1677
4000000	1602	1747	4708
8000000	3622	3603	13906
16000000	16231	8186	46219



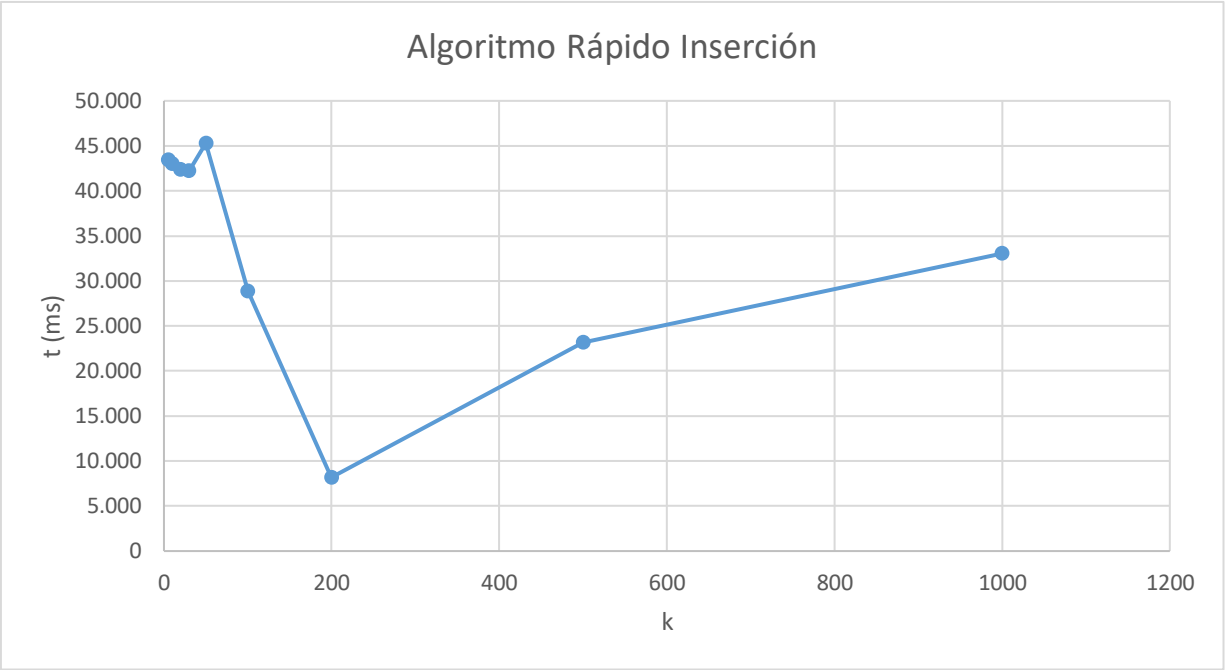
En este caso, se coge como pivote el elemento central, por lo que no se da el caso peor explicado anteriormente y la complejidad de los tres casos es O(n log n). En este caso particular, el caso con los elementos colocados aleatoriamente tarda más en ejecutarse que los dos anteriores. Los tiempos obtenidos son los esperados para este algoritmo.

Teniendo en cuenta los datos obtenidos con este algoritmo, se va a calcular cuántos días tardan en ejecutarse para n = 16000000 los algoritmos anteriores:

- **Burbuja**: tardaría 1827,2 ms, que son 0,021148148 días (1827,2 segundos).
- **Selección**: tardaría 670400 ms, que son 0,00775926 días (670,40 segundos).
- **Inserción**: tardaría 5008 ms, que son 0,057962963 días (5008 segundos).

5. Para finalizar, se va a reimplementar el algoritmo de inserción para que solo ordene las posiciones que se le indiquen. A continuación, se crea una nueva clase denominada **rapidoInserción** que utilice el algoritmo de rápido hasta que el número de elementos sea igual al parámetro k. Este nuevo algoritmo creado es más rápido que Quicksort, pues estos subvectores explicados pueden estar medianamente ordenados y en este caso el método de inserción tiene una complejidad O(n).

Algoritmo	k	t (aleatorio)
Rápido	0	46.219
RápidoInserción	5	43.452
	10	43.016
	20	42.414
	30	42.233
	50	45.292
	100	28.873
	200	8.177
	500	23.185
	1000	33.061



Como se puede ver, todos los tiempos obtenidos con este nuevo algoritmo son mejores que los obtenidos con el algoritmo rápido. Además, se observa que el mejor tiempo obtenido se obtiene para k = 200. Los tiempos obtenidos con el algoritmo rapidoInsercion se esperaban más rápidos que los de Quicksort, ya que el primero solo ordena una parte del vector mientras que el segundo ordena todo el vector. Sin embargo, podría esperarse que para tamaños de vector más grandes este nuevo algoritmo sea peor que Quicksort, pues este suele ser más eficiente gracias a sus sistema de particiones.

Se puede ver que todos los tiempos obtenidos son menores que los obtenidos con Quicksort, por lo que cumple con lo esperado.