ALGORITMOS DE ORDENACION Y SU ESTUDIO COMPARATIVO

Raúl Fernández España – UO278036

2. EL ALGORITMO DE ORDENACIÓN MEJOR: "QUICKSORT"

En la clase RapidoFatal.java se escoge el primer elemento de la lista como el pivote por lo que la partición sería con todos los elementos a la derecha y ningún elemento a la izquierda. Por lo que aumentaría la complejidad temporal del algoritmo, siendo en este caso O(n^2).

Esto nos afecta gravemente a casi todos los tiempos excepto en aquellos que la lista ya que se encuentra ordenada de por sí.

3. TRABAJO PEDIDO

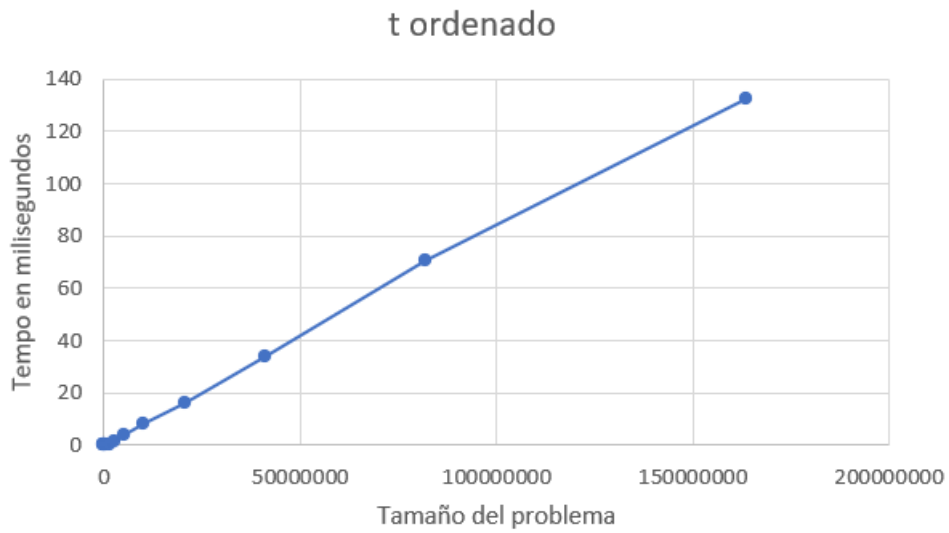
A continuación, se presentan una serie de tablas con 3 filas de tiempos, el primero de ellos hace referencia al tiempo del algoritmo cuando la lista ya se encuentra ordenada, la segunda cuando se encuentra ya ordenada, pero de forma inversa, y la última fila cuando se encuentra ordenada aleatoriamente.

De esta forma podremos analizar como se comporta cada unos de los cuatro algoritmos, (para Inserción, Selección, Burbuja y Quicksort con pivote utilizando mediana a tres) y observar cuanto coste temporal necesitan en base a como se encuentre la lista a ordenar.

Todos los datos han sido cogidos desde un ordenador con **16 GB de RAM y un procesador Intel i7 8750-H**. Aquellos valores de las tablas que aparezcan representados con una 'x', suponen tiempos superiores a media hora

Tiempos Inserción:

N	t ordenado	nVeces	t inverso	t aleatorio	nVeces
10000	0,00226667	1500000	60,347	36,45	150
20000	0,00461111	1500000	244,786	121,91	150
40000	0,00917778	1500000	264,3421	141,22	150
80000	0,01945556	1500000	1115,174	549,78	10
160000	0,03905556	1500000	4472,673	2219,45	10
320000	0,07887778	1500000	19336,89	9795,23	10
640000	0,1618	1500000	84995	42478	1
1280000	0,541	1500000	370483	183862	1
2560000	1,602222	15000	1764498	830250	1
5120000	4,11333	15000	x	x	x
10240000	8,41778	150	x	x	x
20480000	16,496666	150	x	x	x
40960000	34,246666	150	x	x	x
81920000	70,9844	150	x	x	x
163840000	132,74889	150	x	x	x

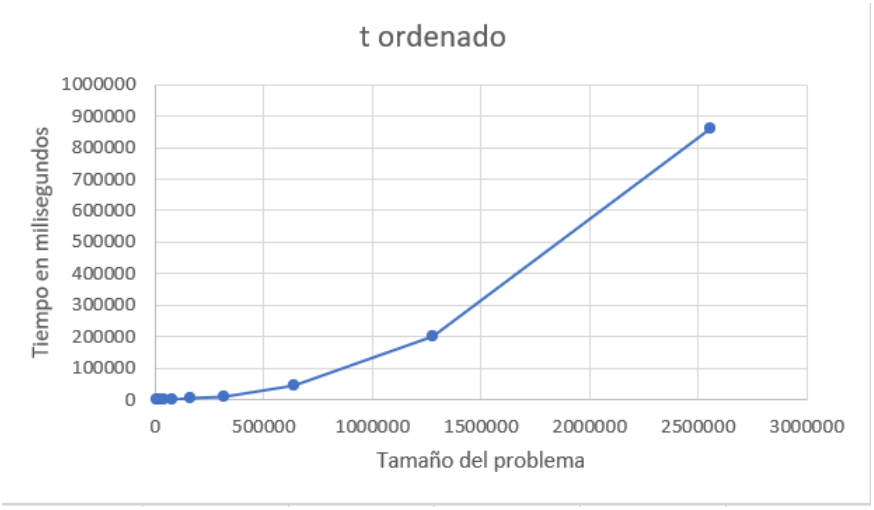


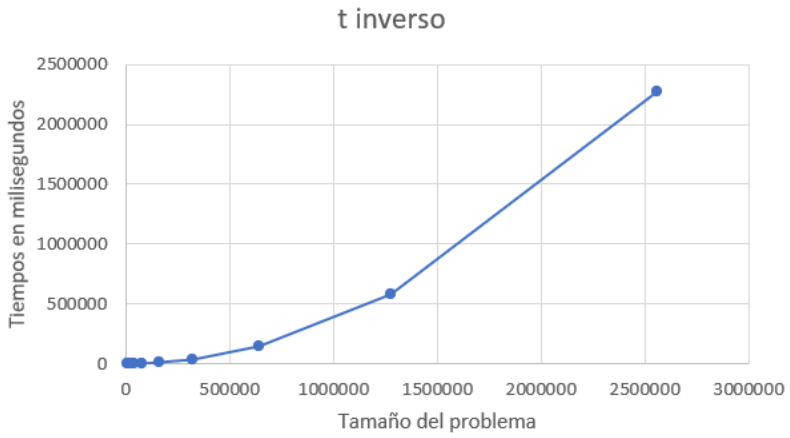
Como se puede observar en las gráficas, se consigue una complejidad lineal O(n) cuando ya se encuentra ordenada, por lo que podemos tomar tiempos fácilmente hasta que se produce un fallo de “heap overflow”. En cambio, para el caso de la lista inversa o aleatoria la complejidad es O(n^2) por lo que podemos tomar tiempos inferiores a media hora hasta tamaños de lista de 2 millones y medio.

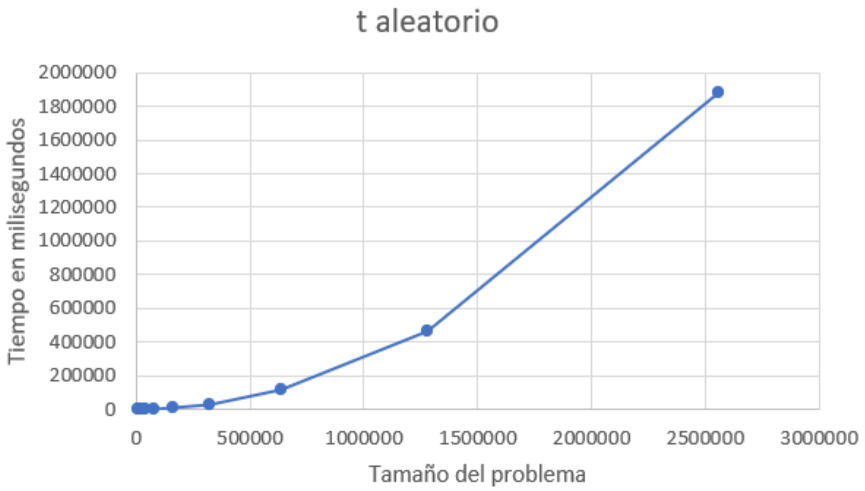
Por tanto podemos observar un mayor coste de tiempos para los casos inversos y aleatorio.

Tiempo Selección:

N	t ordenado	nVeces	t inverso	t aleatorio	nVeces
10000	11,75	150	37,12	39,65	10
20000	35,5	150	127,56	108,23	10
40000	147	10	535	421	1
80000	614,25	10	2135	1657	1
160000	2494,75	10	8479	6536	1
320000	10665	1	34303	26929	1
640000	45307	1	142395	113038	1
1280000	198805	1	584202	465767	1
2560000	862670	1	2271961	1881161	1



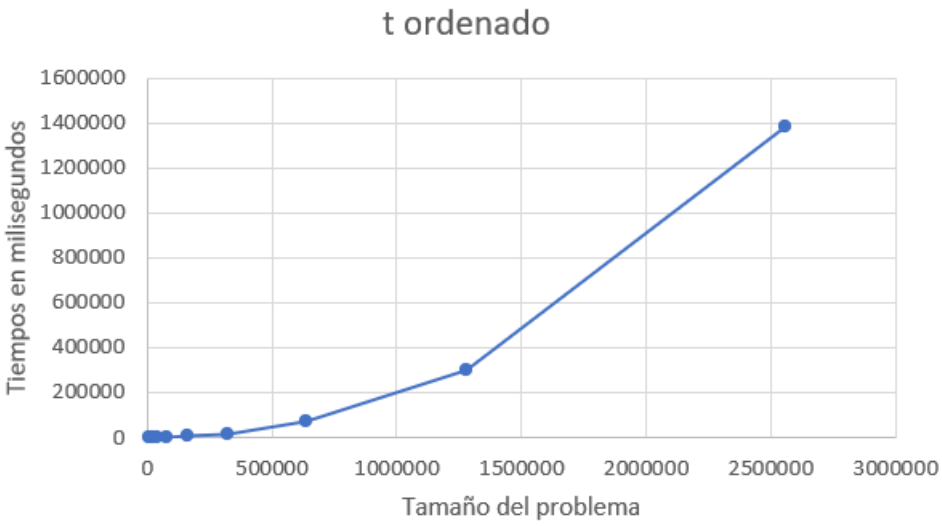


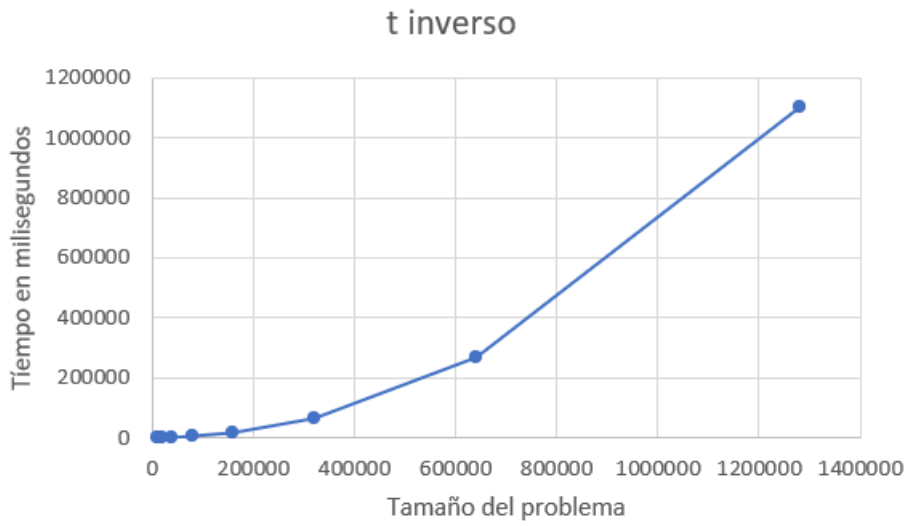


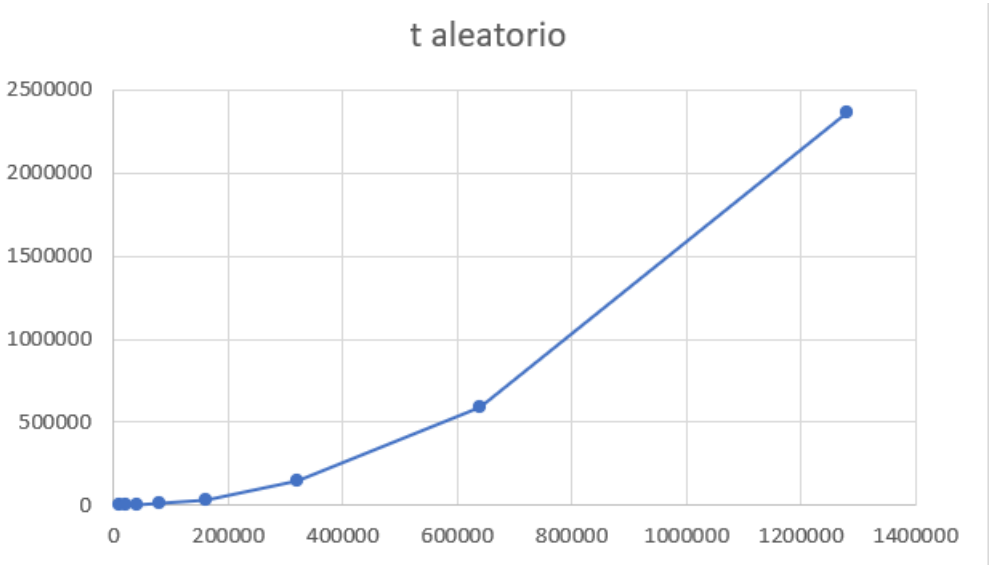
Como se puede observar en las gráficas de este algoritmo, la complejidad es de O(n^2) para todos los casos. Por lo que podemos observar en los costes temporales debido a la complejidad del algoritmo, nos es indistinto utilizar uno u otro indistintamente de como este ordenada la lista a ordenar.

Tiempo Burbuja:

N	t ordenado	nVeces	t inverso	nVeces	t aleatorio	nVeces
10000	21,34	10	64	1	113	1
20000	57	1	237	1	524	1
40000	219	1	927	1	2094	1
80000	939	1	3895	1	8748	1
160000	3762	1	15720	1	35612	1
320000	16261	1	64753	1	144433	1
640000	70246	1	269703	1	587826	1
1280000	297091	1	1102394	1	2362154	1
2560000	1383478	1	x	x	x	


t ordenado
Tiempos en milisegundos
1600000
1400000
1200000
1000000
800000
600000
400000
200000
0
0
500000
1000000
1500000
2000000
2500000
3000000
Tamaño del problema


t inverso
Tíempo en milisegundos
1200000
1000000
800000
600000
400000
200000
0
0
200000
400000
600000
800000
1000000
1200000
1400000
Tamaño del problema

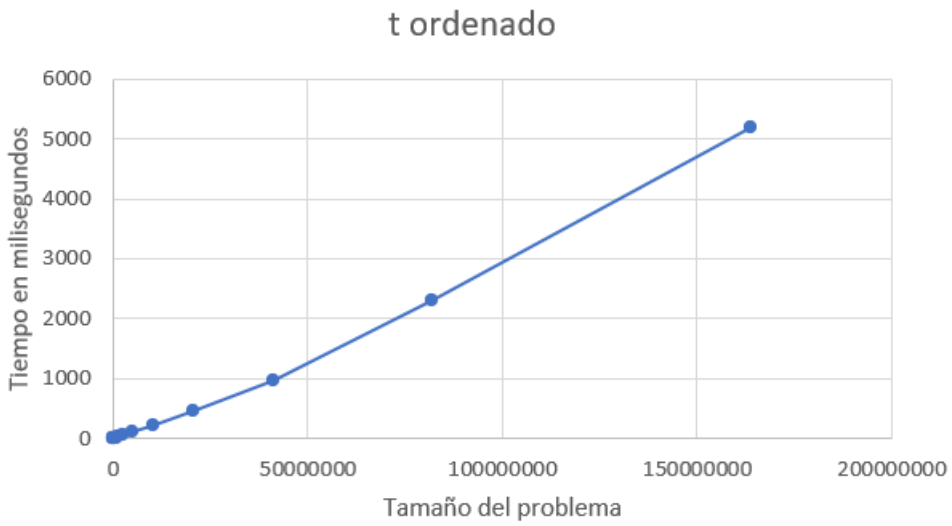


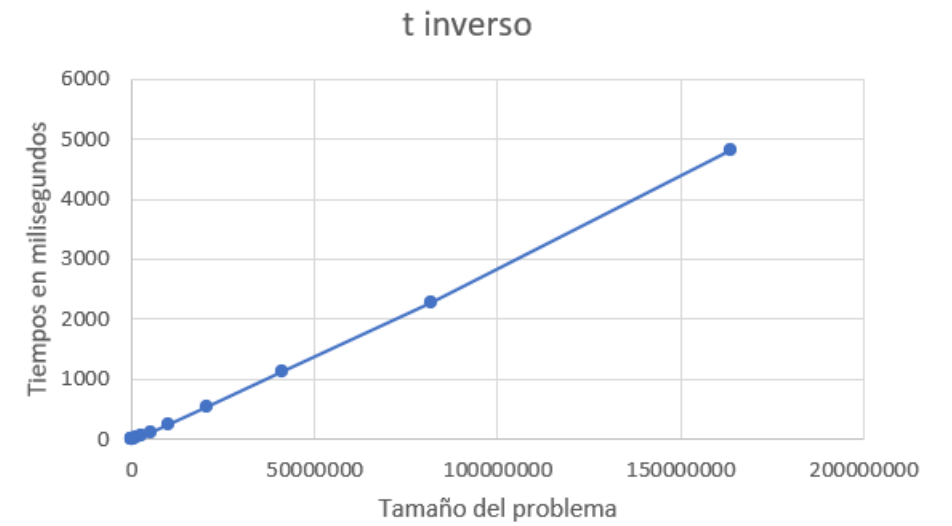
Al igual que el algoritmo de selección, el método de burbuja presenta una complejidad de O(n^2), podemos observar que la complejidad no varia entre los diferentes casos de ordenación de la lista. Aunque si observamos atentamente los tiempos, aunque la complejidad sea la misma los tiempos de ejecución varían ligeramente dependiendo del caso en el que nos encontremos.

Por lo que podemos tomar valores inferiores a medio hora cuando la lista es inferior a 1 millón 280 mil para el caso de lista inversa o aleatoria; y lista de hasta 2 millones y medio de elementos cuando ya se encuentra ordenada.

Tiempo Quicksort:

N	t ordenado	nVeces	t inverso	nVeces	t aleatorio	nVeces
10000	0,1223	1500000	0,142	1500000	0,141	1500000
20000	0,2616	1500000	0,277	1500000	0,272	1500000
40000	0,5502	1500000	0,601	1500000	0,587	1500000
80000	1,1712	1500000	1,236	1500000	1,308	1500000
160000	2,565	1500	2,725	1500	2,756	1500
320000	5,366	1500	5,679	1500	6,308	1500
640000	11,2556	1500	12,096	1500	14,73	1500
1280000	24,78	1500	25,013	1500	36,96	1500
2560000	50,9	150	57,9	150	96,35	150
5120000	107,2	150	120	150	256,97	150
10240000	222	10	250,3	10	748,333	10
20480000	462,7	10	525,8	10	2147,6667	10
40960000	968,6	10	1117,3	10	6644	10
81920000	2296	1	2281,3	10	26358,666	10
163840000	5186	1	4828,5	10	85784	1


t ordenado
Tiempo en milisegundos
6000
5000
4000
3000
2000
1000
0
0
50000000
100000000
150000000
200000000
Tamaño del problema


t inverso
Tiempos en milisegundos
6000
5000
4000
3000
2000
1000
0
0
50000000
100000000
150000000
200000000
Tamaño del problema


t aleatorio
Tiempos en milisegundos
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
0
50000000
100000000
150000000
200000000
Tamaño del problema

El algoritmo Quicksort presenta una complejidad O(n.log(n)) para t aleatorio y t ordenado y O(n^2) para t inverso por lo que podemos tomar tiempos hasta que se produce un fallo de “heap overflow”.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es O**(n·log n)**.

Como hemos podido analizar en el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de **O(n²)**. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas pero el pivote es más determinante para determinar la complejidad del algoritmo