PRÁCTICA 2: ORDENACIÓN

Información del sistema:

Procesador:	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.8GHz
Memoria:	16 GB

Toma de tiempos del algoritmo de **Inserción directa**: Este programa sirve para ordenar n elementos con un algoritmo de complejidad O(n^2). Si la lista de elementos estuviese ordenada tendría complejidad O(n).

N	nVeces	t ordenado miliseg	t ordenado/nVeces
10000	100000	180	0,0018
20000		300	0,0030
40000		618	0,0062
80000		1193	0,0119
160000		2387	0,0239
320000		4877	0,0488
640000		27583	0,2758
1280000		60807	0,6081
2560000		129589	1,2959
5120000		280268	2,8027
10240000		570010	5,7001
20480000	10	72	7,2
40960000		111	11,1
81920000		219	21,9
163840000		451	45,1
327680000		906	90,6
Java heap space			

N	nVeces	t inverso miliseg	t inverso/nVeces
10000	100	63	0,63
20000		214	2,14
40000		689	6,89
80000		2729	27,29
160000		24394	243,94
320000		55937	559,37
640000		174716	1747,16
1280000	Tarda más de 10min con nVeces= 1		

N	nVeces	t aleatorio miliseg	t aleatorio/nVeces
10000	10000	114	0,0114
20000		178	0,0178
40000		414	0,0414
80000		1102	0,1102
160000		7464	0,7464
320000		30323	3,0323
640000		116417	11,6417

1280000		459373	45,9373
2560000	Tarda más de 10min con nVeces= 1		

Toma de tiempos del algoritmo de **Selección**: Este programa sirve para ordenar n elementos con un algoritmo de complejidad O(n^2) en todos los casos.

N	nVeces	t ordenado miliseg	t ordenado/nVeces
10000	10	83	8,3
20000		306	30,6
40000		1225	122,5
80000		4869	486,9
160000		49440	4944,0
320000		140870	14087,0
640000	1	64732	64732
1280000		112961	112961
2560000	Tarda más de 10min con nVeces= 1		

N	nVeces	t inverso miliseg	t inverso/nVeces
10000	10	246	24,6
20000		827	82,7
40000		6278	627,8
80000		26562	2656,2
160000		67469	6746,9
320000		334028	33402,8
640000	1	115083	115083
1280000		478460	478460
2560000	Tarda más de 10min con nVeces= 1		

N	nVeces	t aleatorio miliseg	t aleatorio/nVeces
10000	10	211	21,1
20000		686	68,6
40000		2708	270,8
80000		26451	2645,1
160000		63699	6369,9
320000		179886	17988,6
640000	1	73804	73804
1280000		271274	271274
2560000	Tarda más de 10min con nVeces= 1		

Toma de tiempos del algoritmo **Burbuja**: Este programa sirve para ordenar n elementos con un algoritmo de complejidad O(n^2) en todos los casos.

N	nVeces	t ordenado miliseg	t ordenado/nVeces
10000	10	134	13,4
20000		362	36,2
40000		1363	136,3
80000		5478	547,8
160000		66222	6622,2
320000		269681	26968,1
640000	Tarda más de 10min con nVeces= 1		

N	nVeces	t inverso miliseg	t inverso/nVeces
10000	1	67	67
20000		244	244
40000		898	898
80000		7410	7410
160000		43628	43628
320000		175307	175307
640000	Tarda más de 10min con nVeces= 1		

N	nVeces	t aleatorio miliseg	t aleatorio/nVeces
10000	1	94	94
20000		386	386
40000		1586	1586
80000		17335	17335
160000		28333	28333
320000		162997	162997
640000	Tarda más de 10min con nVeces= 1		

Toma de tiempos del algoritmo **Quicksort (Mediana a tres)**: Este programa sirve para ordenar n elementos con el algoritmo de complejidad O(nlogn).

N	nVeces	t ordenado miliseg	t ordenado/nVeces
10000	1000	105	0,105
20000		205	0,205
40000		392	0,392
80000		891	0,891
160000		1717	1,717
320000		3819	3,819
640000		7305	7,305
1280000		16213	16,213
2560000		32345	32,345

5120000		72078	72,078
10240000		141991	141,991
20480000		319765	319,765
40960000	10	682	68,2
81920000		1646	164,6
163840000		3504	350,4
327680000		7048	704,8
Java heap space			

N	nVeces	t inverso miliseg	t inverso/nVeces
10000	1000	112	0,112
20000		212	0,212
40000		417	0,417
80000		966	0,966
160000		1898	1,898
320000		4816	4,816
640000		9012	9,012
1280000		21303	21,303
2560000		40732	40,732
5120000		87766	87,766
10240000		152638	152,638
20480000		360498	360,498
40960000	1	1256	1256
81920000		2862	2862
163840000		6315	6315
327680000		13213	13213
Java heap space			

N	nVeces	t aleatorio miliseg	t aleatorio/nVeces
10000	1000	114	0,114
20000		217	0,217
40000		425	0,425
80000		975	0,975
160000		2007	2,007
320000		5064	5,064
640000		10536	10,536
1280000		24010	24,010
2560000		52512	52,512
5120000		102775	102,775
10240000		205813	205,813
20480000		440210	440,210
40960000	1	3438	3438
81920000		7310	7310
163840000		17022	17022
327680000		47565	47565
Java heap space			

El método **Quicksort RapidoFatal** comienza con el pivote como el primer elemento de la lista. Su caso fatal es cuando se le pasa una lista de elementos ya ordenados pues su complejidad aumenta, O(n^2). También cuando el pivote es el mínimo o máximo elemento, pues una de las particiones resultará estar vacía.