Jaime González Díaz UO306054

PRACTICA 2

EJERCICIO 1: algoritmo burbuja

n	T ordenado	T invertido	T aleatorio
10000	782	1579	1575
20000	3101	6406	6376
40000	12580	43871	33782
80000	50497	175189	177151
160000	343380	FdT	FdT
320000	FdT	FdT	FdT
640000	FdT	FdT	FdT
1280000	FdT	FdT	FdT

Los tiempos tienen sentido ya que el algoritmo burbuja tiene complejidad cuadrática en todos los casos. Sin embargo es más rápido en el caso de que esté ordenado porque no realiza ningún cambio entre pares de elementos y eso en listas con tamaños n muy grandes supone una diferencia de tiempo considerable, ya que solo recorre un vector ordenado en dos iteraciones anidadas.

EJERCICIO 2: algoritmo selección

n	T ordenado	T invertido	T aleatorio
10000	730	736	746
20000	3505	3547	3560
40000	20175	20970	19941
80000	80587	80537	80142
160000	FdT	FdT	FdT
320000	FdT	FdT	FdT
640000	FdT	FdT	FdT
1280000	FdT	FdT	FdT

Los tiempos tienen sentido ya que el algoritmo siempre es en complejidad O(n^2) sea el caso que sea. Por eso todos tienen tiempos tan similares y de orden cuadrático.

EJERCICIO 3: algoritmo inserción

n	T ordenado	T invertido	T aleatorio
10000	1	631	626
20000	1	2569	2522
40000	1	16977	10106
80000	2	67970	40655
160000	3	277248	259251
320000	5	FdT	FdT
640000	11	FdT	FdT
1280000	21	FdT	FdT

Los resultados tienen sentido ya que el algoritmo inserción se caracteriza por tener una complejidad O(n^2) menos en el mejor caso donde la lista está ordenada, donde no entra en la segunda iteración anidada y se convierte en un algoritmo de complejidad lineal, dando por tanto resultados bajos de complejidad O(n).

EJERCICIO 4: algoritmo rapido

n	T ordenado	T invertido	T aleatorio
10000	5	8	7
20000	10	16	15
40000	21	32	29
80000	43	67	68
160000	87	132	138
320000	217	309	303
640000	403	664	670
1280000	914	1436	1454

1) Los tiempos obtenidos tienen sentido ya que el Quicksort se caracteriza en que la complejidad se mantiene O(nlogn) en todos los casos posibles gracias a su uso de las llamadas a recursividad, hace menos comparaciones si la lista está ordenada y por eso en dicho caso sí tarda menos.
2) Calculando con la formula T(n) de cada algoritmo:
    a. Burbuja: t2=((n2)^2/(n1)^2)*t1. Usando n1=80.000 y t1=177151ms, calculamos que t2=185.756.287.976ms=**215,64 días.**

    b. Selección: t2=((n2)^2/(n1)^2)*t1. Usando n1=80000 y t1=80142ms, calculamos que t2=84.035.000.792ms=**97,55 días.**

    c. Inserción: t2=((n2)^2/(n1)^2)*t1. Usando n1=160000 y t1=259251ms, calculamos que t2=67.961.103.144ms=**78,89 días**.

    d. Rápido: t2=(log(n2)/log(n1))*t1. Usando n1=1.280.000 y t1=1454ms, haciendo log(81920000) calculamos que T(n2)=120.573ms=**0,00014 días.**

EJERCICIO 5: algoritmo rápidoBajos vs inserciónBajos

(Se han usado 100000 repeticiones)

n	T rapido	T insercion	Ttrapido/Tinsercion
10	218	128	1,703125
15	345	207	1,66666667
20	503	347	1,44956772
25	676	451	1,49889135
30	837	699	1,19742489
35	1019	919	1,10881393
40	1241	1209	1,02646816
45	1435	1510	0,95033113
50	1565	1774	0,88218715
55	1791	2125	0,84282353
60	1993	2534	0,78650355
65	2141	2947	0,72650153
70	2441	3407	0,7164661
75	2606	3850	0,67688312
80	2837	4407	0,64374858
85	3029	4845	0,6251806
90	3245	5593	0,58018952
95	3444	6109	0,56375839
100	3591	6712	0,53501192

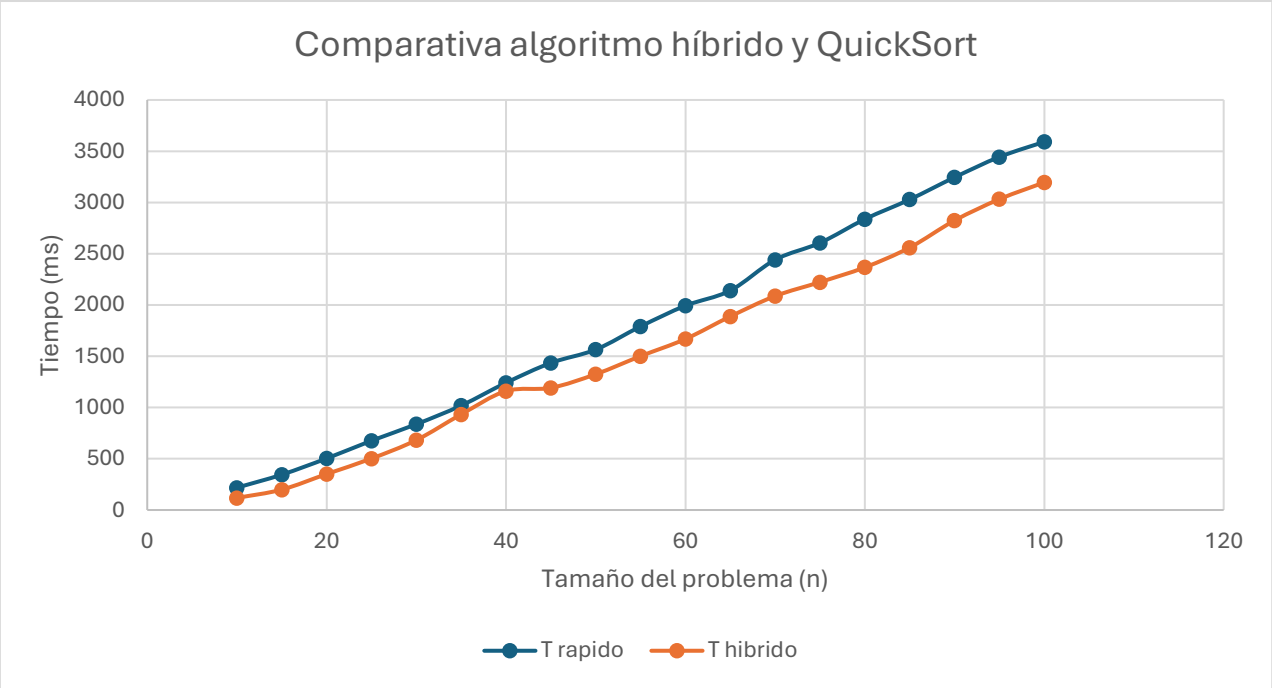
Los tiempos tienen sentido ya que el Quicksort realiza llamadas recursivas que en tamaños muy pequeños ralentizan el proceso, mientras que el Inserción realiza soluciones directas de forma iterativa. Sin embargo, a partir de un tamaño n (40) la llamada recursiva es más eficiente ya que el divide y vencerás en estos tamaños es más práctico a la hora de solucionar problemas para n más pequeños que una solución iterativa cuadrática

EJERCICIO OPCIONAL

n	T rapido	T hibrido	Ttrapido/Thibrido
10	218	118	1,847458
15	345	199	1,733668
20	503	351	1,433048
25	676	502	1,346614
30	837	682	1,227273
35	1019	933	1,092176
40	1241	1160	1,069828
45	1435	1192	1,203859
50	1565	1325	1,181132
55	1791	1501	1,193205
60	1993	1668	1,194844
65	2141	1888	1,134004
70	2441	2087	1,169621
75	2606	2223	1,17229
80	2837	2367	1,198564
85	3029	2560	1,183203
90	3245	2824	1,149079
95	3444	3032	1,135884
100	3591	3195	1,123944

Los tiempos Thibrido representan el código contenido en el archivo AlgoritmoOpcional.java, donde tras ver que el algoritmo de inserción es más rápido que el rápido hasta aproximadamente n=40, apliqué un algoritmo híbrido que utiliza durante los primeros 40 tamaños el algoritmo de inserción y a partir de ahí utiliza el rápido, implementando toda la lógica necesaria de ambos algoritmos en la misma clase, dando así tiempos más rápidos para todos los n desde 10 hasta 100, donde se acercan más los tiempos entre ambos algoritmos cuanto más aumenta el tamaño del problema al estar aplicando el mismo algoritmo de ordenación.

Incluso en los tiempos más altos mi algoritmo híbrido sigue siendo mas eficiente ya que durante las llamadas recursivas, cuando llega a una partición de tamaño inferior al cruce (en este caso, 40), deja de emplear Quicksort y emplea algoritmo de inserción por lo antes dicho, que es más rápido para estos tamaños.



(Sé que me dijiste que llamase al Rapido e Insercion desde sus clases pero me empeñé en querer rascar tiempos más pequeños)