Miguel Méndez Murias
Algoritmia - UO287687

PRÁCTICA 2

El algoritmo de ordenación mejor: Quicksort

En esta primera parte, nos encargamos de estudiar en profundidad el algoritmo de ordenación Quicksort, que es, demostrado matemáticamente, el mejor algoritmo de ordenación existente.
Este algoritmo se basa en la elección de un pivote, pero esta elección influye de manera muy importante en la complejidad. Para comprobarlo, realizamos mediciones del algoritmo escogiendo el elemento central como pivote (mejor) y eligiendo como pivote uno de los extremos (peor).
La elección del elemento central como pivote permite obtener una complejidad O(n*log n), tal y como se puede observar en los siguientes apartados de este documento. Sin embargo la elección de un extremo como pivote produce una complejidad $O(n^2)$, tal y como se puede observar a continuación:

- Quicksort con un extremo como pivote (caso fatal)

Tamaño del problema	Tiempo (µs)
100	4
200	5
400	20
800	75
1600	262
3200	1032
6400	3979

¿En que consiste el método de selección de pivote en el algoritmo Quicksort?

El algoritmo Quicksort es un método recursivo, y se elige un pivote para dividir en dos particiones la lista de elementos: en la partición izquierda se colocan los elementos menores que el pivote, mientras que en la partición derecha se colocan los elementos mayores que el pivote.
Teniendo en cuenta esto, se puede ver que si se escoge como pivote el mayor o el menor

de los elementos el algoritmo sería ineficiente, ya que una partición tendría todos los elementos menos el pivote y la otra estaría vacía, por lo que al realizar las llamadas recursivas se acabaría produciendo una complejidad $O(n^2)$, ya que no se divide el tamaño del problema. Sin embargo, si se escoge como pivote un elemento central, si que se dividiría el problema, ya que habría elementos mayores que él que estarían en la partición derecha y lo mismo pasaría con los menores y la partición izquierda. De esta forma, al realizar las llamadas recursiva se conseguir dividir el tamaño del problema, por lo que se conseguiría una complejidad $O(n * \log n)$.
Sabiendo esto, cuanto más centrado esté el pivote, más rápido será el algoritmo, por lo que la elección de un buen pivote es muy importante.

Trabajo pedido

En esta parte de la práctica, tuvimos que realizar mediciones de tiempo de distintos algoritmos de ordenación en tres casos diferentes:

- Con una lista de elementos ordenados
- Con una lista de elementos en orden inverso
- Con una lista de elementos en un orden aleatorio

Además, tras realizar las mediciones, tuvimos que comprobar que efectivamente se cumplía la complejidad propia de ese algoritmo para ese caso.

A continuación se muestran las tablas de tiempo de los distintos algoritmos:

Inserción directa

Tamaño del problema	Ordenado	Inversa	Aleatorio
10000	0,0019	13	8
20000	0,0031	35	20
40000	0,0051	136	62
80000	0,0105	546	245
160000	0,0222	2260	998
320000	0,0414	9281	3949
640000	0,0899	41770	17221
1280000	0,1686	169806	71179
2560000	0,3433	686301	281796
5120000	0,8723	-	-
10240000	2,4537	-	-
20480000	5,2844	-	-
40960000	11,0373	-	-
81920000	22,2583	-	-
163840000	43,3342	-	-
327680000	85,0924	-	-

Selección

Tamaño del problema	Ordenado	Inversa	Aleatorio
10000	15	35	76
20000	38	329	301

Miguel Méndez Murias
Algoritmia - UO287687

40000	149	1364	1207
80000	554	5291	4838
160000	2001	21485	19329
320000	8705	85649	77456
640000	50270	336456	309733
1280000	236758	1525710	1240144
2560000	696086	5102332	4961867

Burbuja

Tamaño del problema	Ordenado	Inversa	Aleatorio
10000	10	59	76
20000	30	220	361
40000	113	886	1616
80000	475	3514	6739
160000	1858	14062	27434
320000	7313	56490	110961
640000	29585	226886	450858
1280000	119166	907799	1824550
2560000	486653	3627435	7331438

Quicksort con pivote central

Tamaño del problema	Ordenado	Inversa	Aleatorio
10000	0,117	0,128	0,11
20000	0,225	0,231	0,224
40000	0,433	0,435	0,459
80000	0,954	0,927	0,975
160000	1,979	1,966	2,134
320000	3,921	4,489	4,689
640000	9,304	9,18	10,408
1280000	18,729	18,85	25,82
2560000	38,21	38,737	58,021

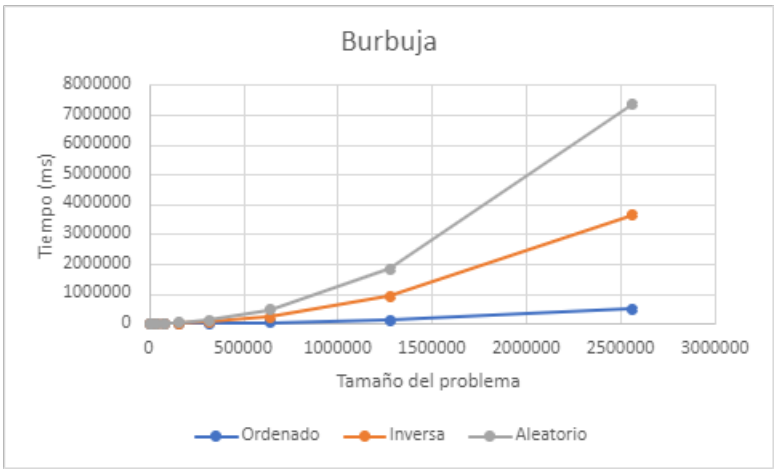
*En algunos casos el tamaño del problema máximo es menor que el permitido por el tamaño de un entero en java ya que el tiempo era demasiado elevado y ya disponíamos de suficientes datos como para garantizar que se cumplía la complejidad.

**Todos los tiempos están en milisegundos

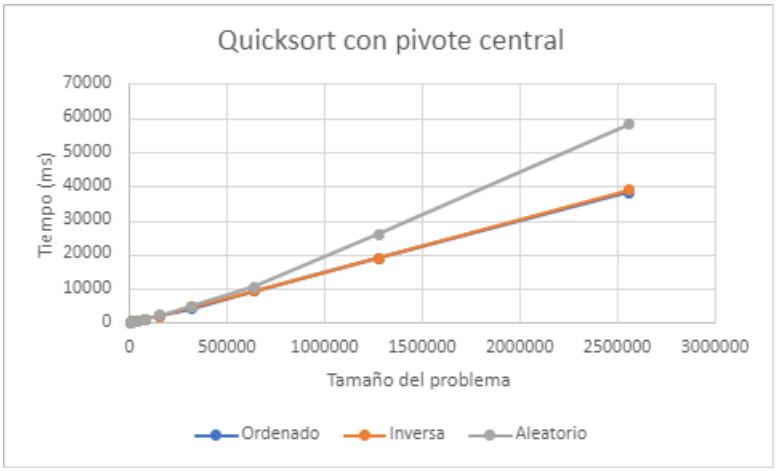
***Las mediciones se han realizado con un ordenador con un procesador Intel Core i7-12700H con 16GB de memoria RAM

Miguel Méndez Murias
Algoritmia - UO287687

Gráficas correspondientes



Miguel Méndez Murias
Algoritmia - UO287687



Resultados

Tras analizar los datos de tiempo obtenidos y las gráficas generadas a partir de ellos podemos garantizar que se cumplen las siguientes complejidades:

Algoritmo	**Caso**	**Complejidad**
Inserción directa	Ordenado	O(n)
Inserción directa	Inversa	O(n²)
Inserción directa	Aleatorio	O(n²)
Selección	Ordenado	O(n²)
Selección	Inversa	O(n²)
Selección	Aleatorio	O(n²)
Burbuja	Ordenado	O(n²)
Burbuja	Inversa	O(n²)
Burbuja	Aleatorio	O(n²)
Quicksort con pivote central	Ordenado	O(n*log n)
Quicksort con pivote central	Inversa	O(n*log n)
Quicksort con pivote central	Aleatorio	O(n*log n)