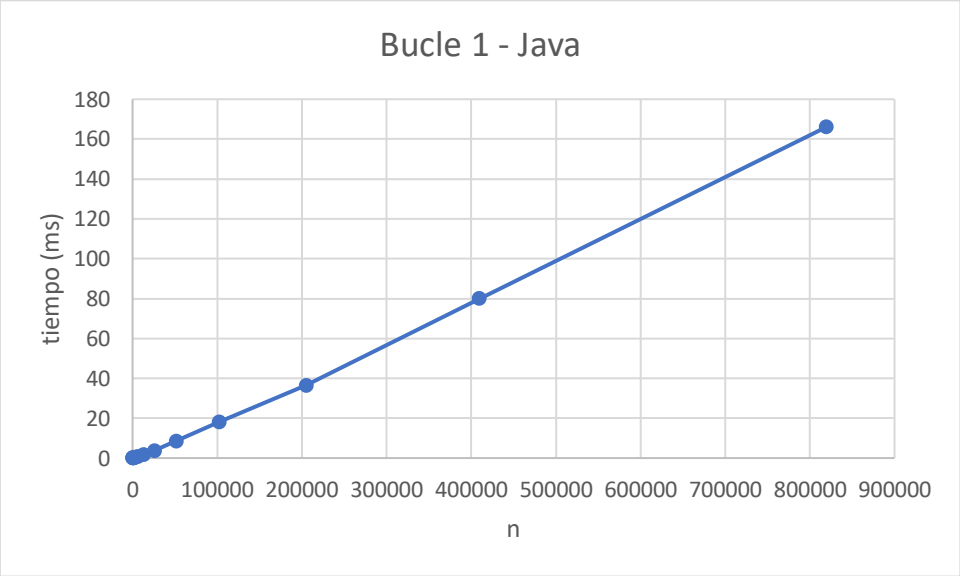
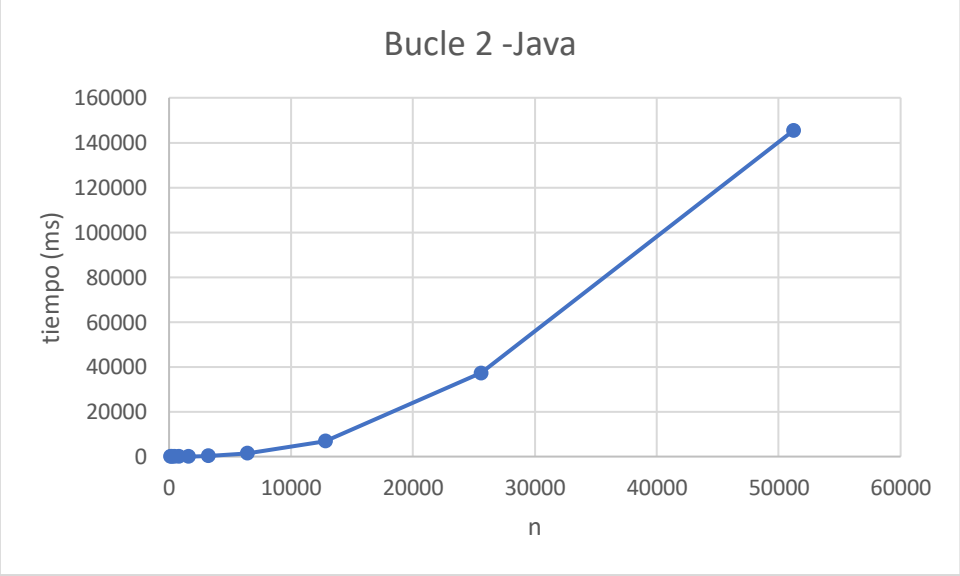
PRÁCTICA 1.2 ALGORITMIA

1. Tras medir los tiempos de ejecución de los 4 bucles dados en Java, se procede a decir sus complejidades y a la comparación de sus tiempos:

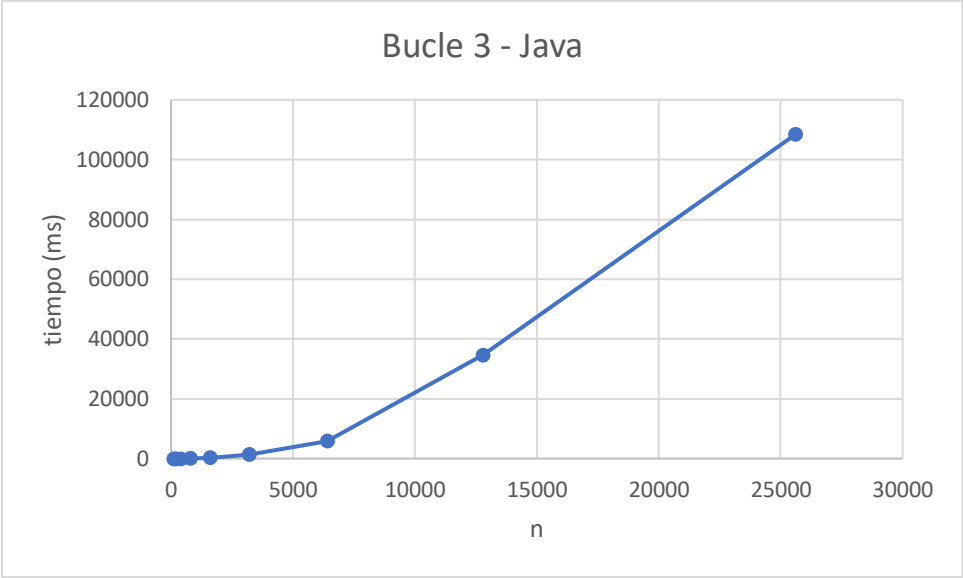
n	t Bucle 1	t Bucle 2	t Bucle 3	t Bucle 4
100	0.0085	0.268	0.86	1.17
200	0.0169	1.11	3.71	10
400	0.0355	4.7	16.6	68
800	0.087	22.3	74	537
1600	0.176	88	304	4359
3200	0.37	387	1356	34177
6400	0.76	1529	5858	476633
12800	1.71	6954	34648	FdT
25600	3.66	37367	108378	FdT
51200	8.5	145356	152148	FdT



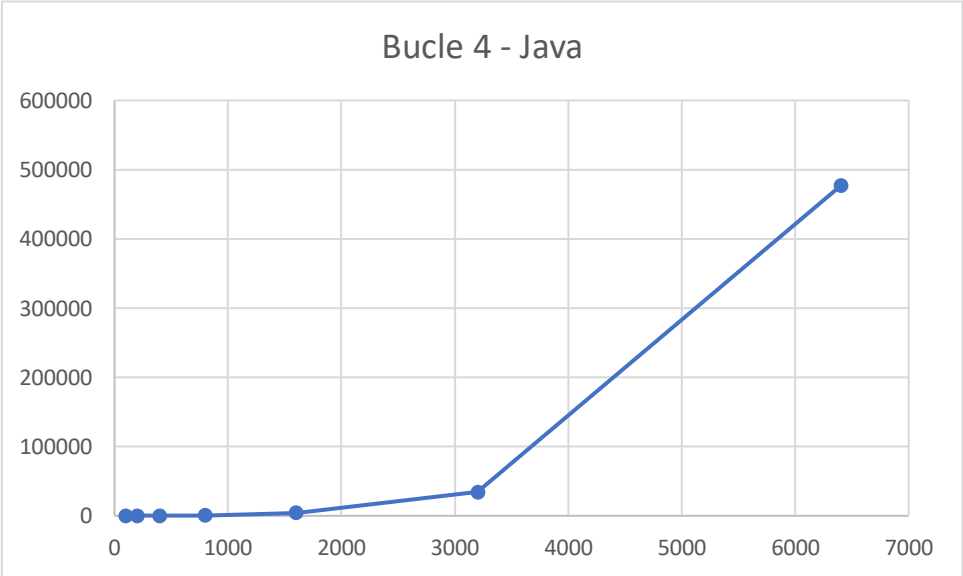
El bucle 1 tiene una complejidad nlog(n).



El bucle 2 tiene una complejidad $n^2\log(n)$.



El bucle 3 tiene una complejidad es $n^2\log(n)$



El bucle 4 tiene una complejidad de n^3.

2. Ahora se implementan las clases Bucle5, Bucle6 y Bucle7 en Java con las siguientes complejidades: $O(n^2\log_2 n)$, $O(n^3 \log n)$ y $O(n^4)$.

n	t Bucle 5	t Bucle 6	t Bucle 7
100	1.1	7.4	40.4
200	5.2	81	410
400	23.6	565	6050
800	112	4929	93835
1600	549	43361	FdT
3200	2536	FdT	FdT
6400	11691	FdT	FdT

Como se puede ver en la tabla, a mayor complejidad mayor tiempo de ejecución del algoritmo, siendo en este caso el que más tarda el bucle 7.

3. A continuación, se van a comparar dos algoritmos con **distinta complejidad**. En este caso, el bucle 1 tiene una complejidad nlog(n) y el bucle 2 tiene una complejidad n^2log(n).

n	t Bucle 1 (t1)	t Bucle 2 (t2)	t1/t2
100	0.0085	0.268	0.03171642
200	0.0169	1.11	0.01536364
400	0.0355	4.7	0.00755319
800	0.087	22.3	0.00390135
1600	0.176	88	0.002
3200	0.37	387	0.00095607
6400	0.76	1529	0.00049706
12800	1.71	6954	0.0002459
25600	3.66	37367	9,7947E-05
51200	8.5	145356	5,8477E-05

Se puede ver que independientemente del entorno de desarrollo, al dividir algoritmos de distinta complejidad el cociente siempre va a ir tendiendo a 0 cuando el de mayor complejidad se divida entre el algoritmo de menor complejidad. Si fuera al contrario, la división crecería sucesivamente y tendería a infinito.
Por esta razón, cuanto mayor sea el tamaño de un problema, mas conveniente es tener algoritmos con complejidades menores.

A continuación, se va a mostrar una tabla que compara dos algoritmos con la **misma complejidad**.

n	t Bucle 3 (t1)	t Bucle 2 (t2)	t3/t2
100	0.86	0.268	3,20895522
200	3.71	1.11	3,37272727
400	16.6	4.7	3,53191489
800	74	22.3	3,31838565
1600	304	88	3,45454545
3200	1356	387	3,50387597
6400	5858	1529	3,83126226
12800	34648	6954	4,98245614
25600	108378	37367	2,90036663
51200	152148	145356	1,04672666

En el caso de algoritmos de la misma complejidad sí que hay que implementar los dos algoritmos en el mismo entorno de desarrollo. Como se puede ver, todos los valores giran alrededor de una constante (en este caso el 3). Como esta constante es mayor de 1, es 3 veces mejor el algoritmo que hemos colocado en el denominador de la división, es decir, el bucle 2.

Ahora vamos a comparar el mismo algoritmo en **distintos entornos de desarrollo**.

n	Bucle 4 (Py) t41	Bucle 4 (Java_SIN) t42	Bucle 4 (Java_CON) t43	t42/t41	t43/t42
100	4	1.8	0.0211	0.45	0.01172222
200	31	11.1	0.098	0.35806452	0.00882883
400	281	68	0.469	0.24199288	0.00689706
800	2614	537	2.74	0.20543229	0.00510242
1600	23225	4359	20.6	0.18768568	0.00472585
3200	258397	34177	132	0.13226547	0.00386225
6400	FdT	476633	918	FdT	0.00192601

En la tabla anterior se puede ver que los tiempos en Python son superiores a los tomados en Java, con y sin optimización. Además, esto mismo lo demuestra el cociente t42/t41, en el que se puede ver que todos los valores oscilan en torno al 0. Esto es así debido a que el algoritmo situado en el numerador (en Java sin optimización) es más eficiente que el colocado en el denominador (en Python).
Por otra parte, en la comparación de los tiempos en Java con y sin optimización, se ve en la toma de tiempos que es bastante mejor en el caso con optimización. Esto se confirma en el cociente t43/t42, en el que se puede ver que todos los resultados obtenidos están muy próximos al 0. De nuevo, esto es así porque los tiempos del algoritmo colocado en el numerador (en Java con optimización) son bastante más bajos que los tiempos del algoritmo colocado en el denominador (en Java sin optimización).