

PRÁCTICA 1.1

TOMA DE TIEMPOS DE EJECUCIÓN

Se utiliza para ello el método `currentTimeMillis ()` de la clase `System` del paquete de Java `java`. Ese método `currentTimeMillis ()` devuelve un entero de tipo `long` (64 bits), que es el milisegundo actual que vive el ordenador (la cuenta a 0 se puso hace más 50 años: 1 de enero de 1970).

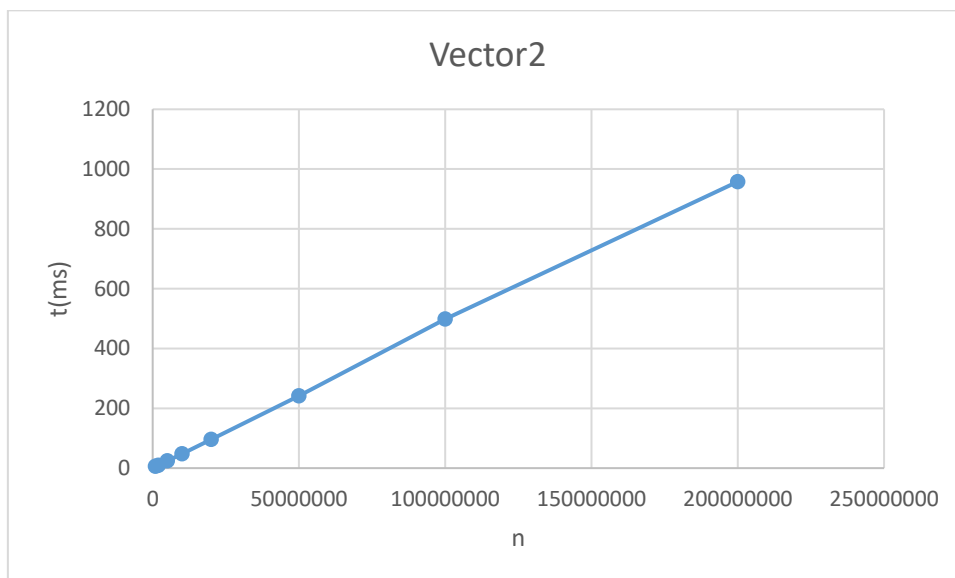
El valor máximo que puede representar un `long` es $2^{63} - 1$ ms. En años sería $2,9 \cdot 10^8$ años.

Por lo tanto, se podrá seguir usando esta forma de contar durante **$2,9 \cdot 10^8$ años**.

$5,8 \times 10^8$

Mediciones Vector2

n	Tiempo(ms)
1000000	6
2000000	9
5000000	24
10000000	48
20000000	96
50000000	241
100000000	498
200000000	958



¿Por qué a veces el tiempo medido sale 0?

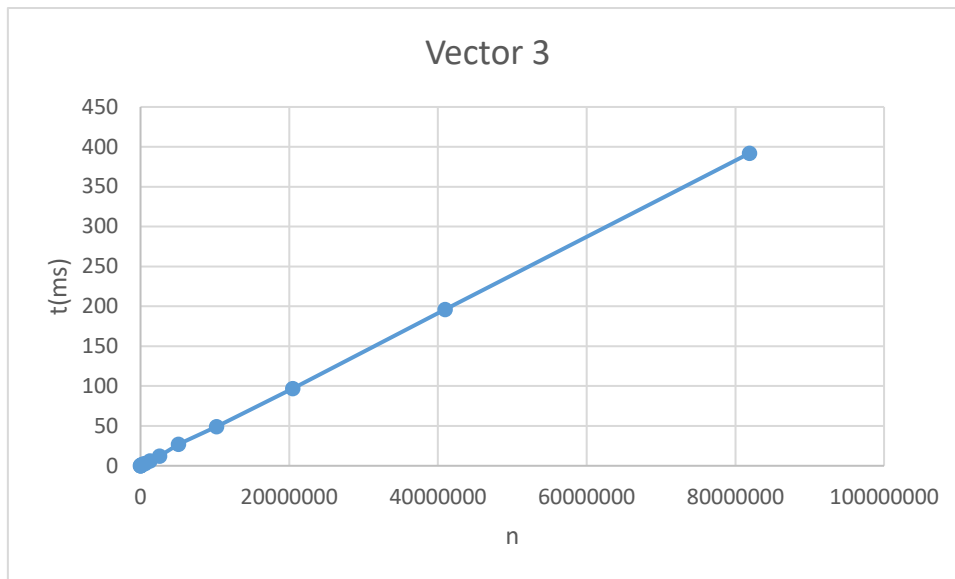
Porque el `n` no es suficientemente grande.

A partir de `n = 1000000` se empiezan a obtener tiempos fiables.

Mediciones Vector3

n	tTiempo
10000	0
20000	0
40000	0
80000	1
160000	1
320000	2
640000	3
1280000	6
2560000	12
5120000	27
10240000	49
20480000	97
40960000	196
81920000	392

n	Tiempo(ms)
10000	0
20000	0
40000	0
80000	1
160000	1
320000	2
640000	3
1280000	6
2560000	12
5120000	27
10240000	49
20480000	97
40960000	196
81920000	392



En el Vector3 las mediciones por debajo de 50 ms no son fiables.

Mediciones Vector 4

n	Tiempo(ms)
10000	0.11663
20000	0.22585
40000	0.45124
80000	0.88732
160000	1.7734
320000	3.43
640000	7.18
1280000	14.06
2560000	28.48
5120000	57.74
10240000	113.44
20480000	227.23
40960000	479.44
81920000	926.14

¿Qué pasa con el tiempo si el tamaño del problema se multiplica por 2?

Al duplicar el tamaño el tiempo de ejecución también se duplica (hay un cierto error asociado). Esto se debe a la complejidad lineal del algoritmo.

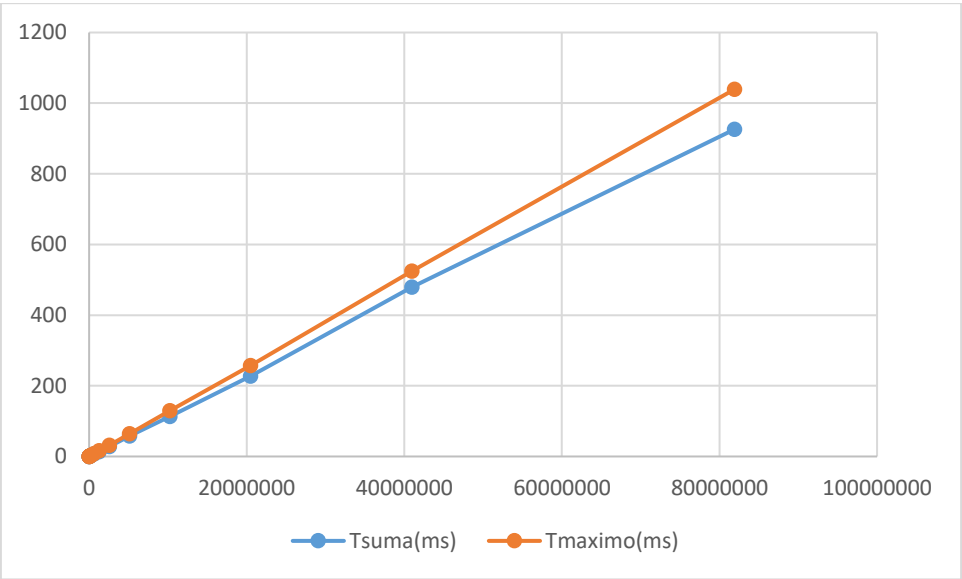
¿Qué pasa con el tiempo si el tamaño del problema se multiplica por otro k que no sea 2?

(Pruebe, por ejemplo, para k=3 y k=4 y compruebe los tiempos obtenidos.) Razone si los tiempos obtenidos son los que se esperaban de la complejidad lineal $O(n)$.

Si el tamaño del problema se multiplica por otro k, el tiempo obtenido también se multiplicaría por k. Los tiempos obtenidos son los esperados según la complejidad $O(n)$ ya que los tiempos crecen de forma proporcional al tamaño.

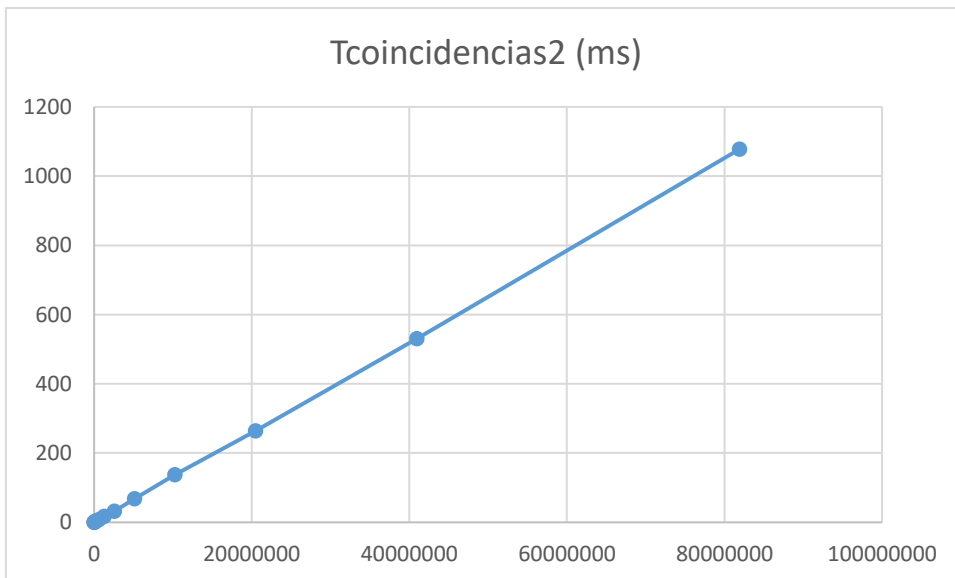
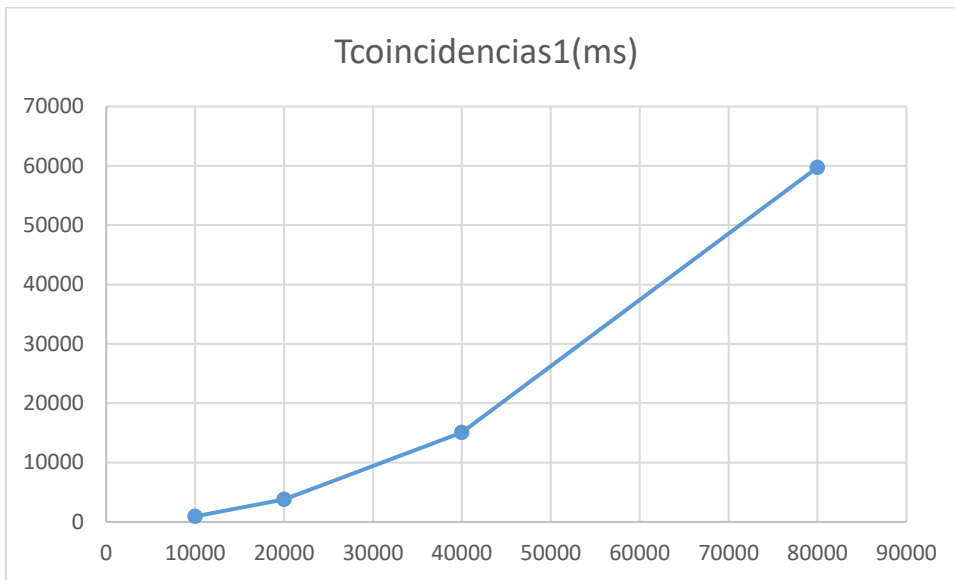
Medición de tiempos Vector5, Vector6 y Vector7:

n	Tsuma(ms)	Tmaximo(ms)
10000	0,11663	0,12704
20000	0,22585	0,25054
40000	0,45124	0,49714
80000	0,88732	1,01856
160000	1,7734	2,02031
320000	3,43	4,10931
640000	7,18	7,97
1280000	14,06	16,1
2560000	28,48	32,2
5120000	57,74	64,0
10240000	113,44	129,7
20480000	227,23	257,1
40960000	479,44	525,0
81920000	926,14	1039,4



n	Tcoincidencias1(ms)	Tcoincidencias2 (ms)
10000	938	0,1355
20000	3796	0,2674
40000	15047	0,5204
80000	59750	1,03
160000	FdT	2,18
320000	FdT	4,07

640000	FdT	8,43
1280000	FdT	17,08
2560000	FdT	32,2
5120000	FdT	67,96
10240000	FdT	137,53
20480000	FdT	264
40960000	FdT	531
81920000	FdT	1078



Los tiempos de ejecución de Vector5(suma) y Vector6(máximo) crecen de proporcional al tamaño del problema n . Este hecho confirma que ambos algoritmos tienen una complejidad $O(n)$.

El algoritmo coincidencias1 presenta complejidad cuadrática tal y como se puede apreciar en la gráfica. Por lo tanto, la diferencia respecto a coincidencias2(complejidad lineal) se va haciendo cada vez más notable.

Equipo utilizado en las mediciones

Procesador: Intel(R) Core(TM) i5-6500T CPU @ 2.50GHz 2.50 GHz

RAM instalada: 16,0 GB