

ALGORITMIA

PRÁCTICA 1.1

Héctor Lavandeira Fernández
UO277303 | UNIVERSIDAD DE OVIEDO – CURSO 2021/22

Características principales del ordenador:

Procesador:	i5-8250U
Memoria RAM:	8GB

NOTA: Todas las medidas de tiempos están en milisegundos.

• TOMA DE TIEMPOS DE EJECUCIÓN

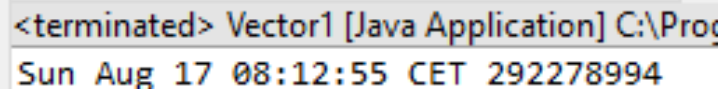
¿Cuántos años más podemos seguir utilizando esta forma de contar?

El método `currentTimeMillis()` empieza a contar desde el 1 de enero de 1970, por lo que solo podrá utilizarse mientras el valor de milisegundos pueda representarse en 64 bits.

Para saber cuál es el valor máximo, ejecutamos:

```
System.out.println(new Date(Long.MAX_VALUE));
```

Y obtenemos el siguiente resultado:



```
<terminated> Vector1 [Java Application] C:\Pro  
Sun Aug 17 08:12:55 CET 292278994
```

Como el año máximo es 292.278.994, podemos afirmar que el método podrá seguir utilizándose por lo menos durante 292.276.972 años.

¿Qué significa que el tiempo medido sea 0?

Significa que el proceso se ha realizado en menos de 1 milisegundo.

¿A partir de qué tamaño de problema (n) empezamos a obtener tiempo fiables?

El tamaño de problema aconsejado es aquel que proporcione mediciones de, por lo menos, 50 milisegundos.

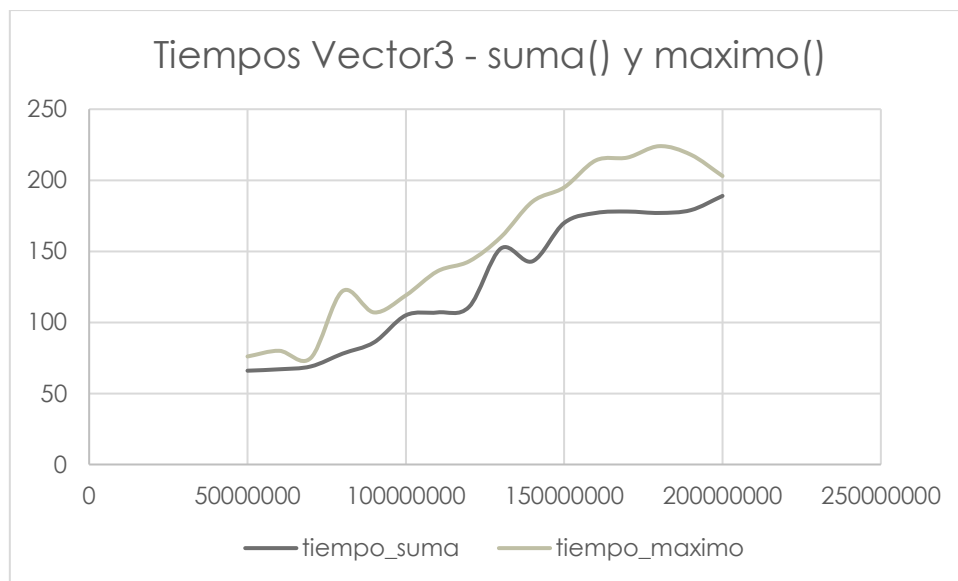
- **CRECIMIENTO DEL TAMAÑO DEL PROBLEMA**

¿Qué pasa con el tiempo si el tamaño del problema se multiplica por 5?

Al aumentar el tamaño del problema, aumentarán los tiempos de las mediciones.

¿Los tiempos obtenidos son los que se esperaban de la complejidad lineal $O(n)$?

No. Para ambos métodos, algunos tamaños de problema provocan picos en las mediciones de tiempo, por lo tanto, dichas mediciones no son fiables, ni se ajustan a la complejidad lineal.



- **TABLA 1**

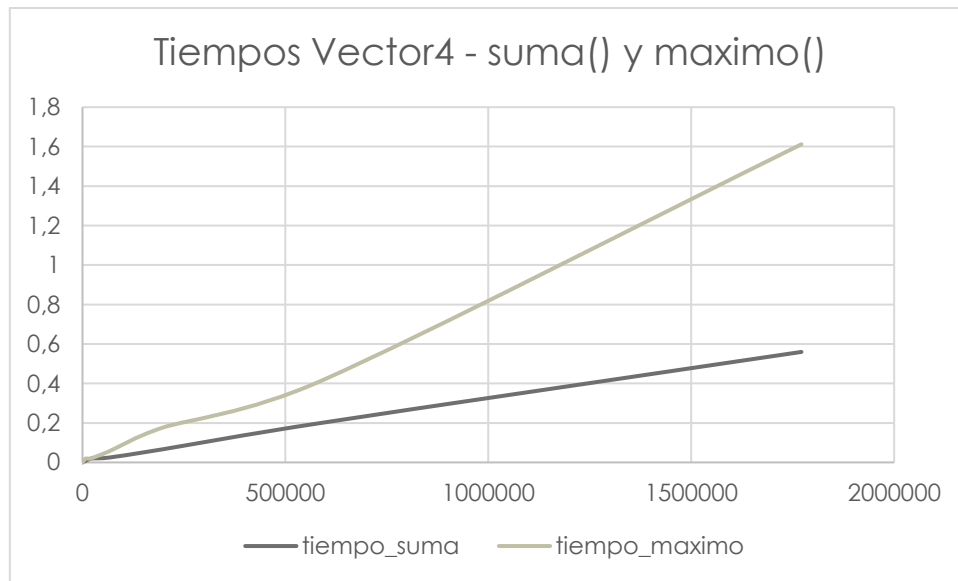
Tabla de mediciones para el método suma():

<i>n</i>	<i>tiempo</i>	<i>tiempo / repeticiones</i>	<i>repeticiones</i>
10	57	0,000019	3000000
30	102	0,000034	3000000
90	248	8,26667E-05	3000000
270	824	0,000274667	3000000
810	2510	0,000836667	3000000
2430	6978	0,002326	3000000
7290	22426	0,007475333	3000000
21870	58346	0,019448667	3000000
65610	2466	0,02466	100000
196830	6629	0,06629	100000
590490	20087	0,20087	100000
1771470	55983	0,55983	100000
135000000	127539	127,539	1000

Tabla de mediciones para el método máximo():

<i>n</i>	<i>tiempo</i>	<i>tiempo / repeticiones</i>	<i>repeticiones</i>
10	70	0,00007	1000000
30	103	0,000103	1000000
90	238	0,000238	1000000
270	700	0,0007	1000000
810	1981	0,001981	1000000
2430	6338	0,006338	1000000
7290	19115	0,019115	1000000
21870	2133	0,02133	100000
65610	5630	0,0563	100000
196830	17591	0,17591	100000
590490	41545	0,41545	100000
1771470	16118	1,6118	10000
135000000	121749	121,749	1000

NOTA: A la hora de representar los tiempos de ambas mediciones en un gráfico, he ignorado el último valor, que representa el último valor de tiempo antes de que pete (es demasiado grande, y los valores más pequeños no serían visibles).



¿Cumplen los valores obtenidos con lo esperado?

Los resultados obtenidos sí cumplen con lo esperado, ya que los tiempos crecen linealmente, de acuerdo a la complejidad $O(n)$ de ambos métodos.

- **TABLA 2**

Tabla de mediciones para el método `sumarDiagonal1()`:

<i>n</i>	<i>tiempo</i>	<i>tiempo / repeticiones</i>	<i>repeticiones</i>
3	52	0,000104	500000
6	92	0,000184	500000
12	200	0,0004	500000
24	719	0,001438	500000
48	2526	0,005052	500000
96	13083	0,026166	500000
192	40174	0,080348	500000
384	14936	0,14936	100000
768	5661	0,5661	10000
1536	21116	2,1116	10000
3072	73267	7,3267	10000

Gráfico con los tiempos representados:



Los resultados cumplen con lo esperado, ya que la complejidad del método `sumarDiagonal1()` es $O(n^2)$, y la gráfica tiene una tendencia cuadrática.

Tabla de mediciones para el método sumarDiagonal2():

<i>n</i>	<i>tiempo</i>	<i>tiempo / repeticiones</i>	<i>repeticiones</i>
3	33	0,000066	500000
6	38	0,000076	500000
12	43	0,000086	500000
24	50	0,0001	500000
48	68	0,000136	500000
96	112	0,000224	500000
192	313	0,000626	500000
384	883	0,001766	500000
768	1707	0,003414	500000
1536	14283	0,028566	500000
3072	47131	0,094262	500000

Gráfico con los tiempos representados:



Los resultados cumplen con lo esperado, ya que la complejidad de `sumarDiagonal2()` es $O(n)$, y la gráfica tiene tendencia lineal. Para valores pequeños del tamaño del problema, los tiempos varían y no siguen la tendencia lineal, pero al crecer el tamaño del problema, la complejidad del método se respeta.

- **BENCHMARKING: INFLUENCIA DE LA PLATAFORMA**

¿A qué se deben las diferencias de tiempos en la ejecución entre uno y otro programa?

La diferencia en los tiempos de ejecución se debe a que Java es más rápido por ser un lenguaje compilado, mientras que Python, al ser interpretado, es más lento.

Independientemente de los tiempos concretos ¿existe alguna analogía en el comportamiento de las dos implementaciones?

La analogía principal es que las mediciones de tiempo con ambas implementaciones cumplen la complejidad indicada.

<i>n</i>	<i>tiempo Java</i>	<i>tiempo Python</i>
1000000	11	48
2000000	14	96
4000000	7	183
8000000	13	411
16000000	24	917
32000000	49	1754
64000000	98	3013
128000000	196	6028
256000000	437	
512000000	689	
1024000000	1338	
2048000000	2308	
4096000000	5002	

