Algoritmia

Práctica 1.1

8/2/2026

Contenidos

1. Desbordamiento de fecha 2
2. Toma de tiempos nulos en `Vector2` 2
3. Toma de tiempos `Vector4` 2
4. Suma vs Máximo 3
5. Coincidencias 1 vs Coincidencias 2 4
6. Comparación con tiempos teóricos 5
7. Condiciones de medida y características del ordenador . . . 6

1. Desbordamiento de fecha

Sabemos que `currentTimeMillis()` devuelve un entero `long` de 64 bits. Así, al tener signo, el máximo valor que puede representar es $2^{63} - 1$. Hagamos el factor de conversión de milisegundos a años:

$$(2^{63} - 1)\text{ms} \cdot \frac{1\text{s}}{10^3\text{ms}} \cdot \frac{1\text{h}}{3600\text{s}} \cdot \frac{1\text{d}}{24\text{h}} \cdot \frac{1\text{año}}{365.25\text{d}} = 292.271.023\text{años}$$

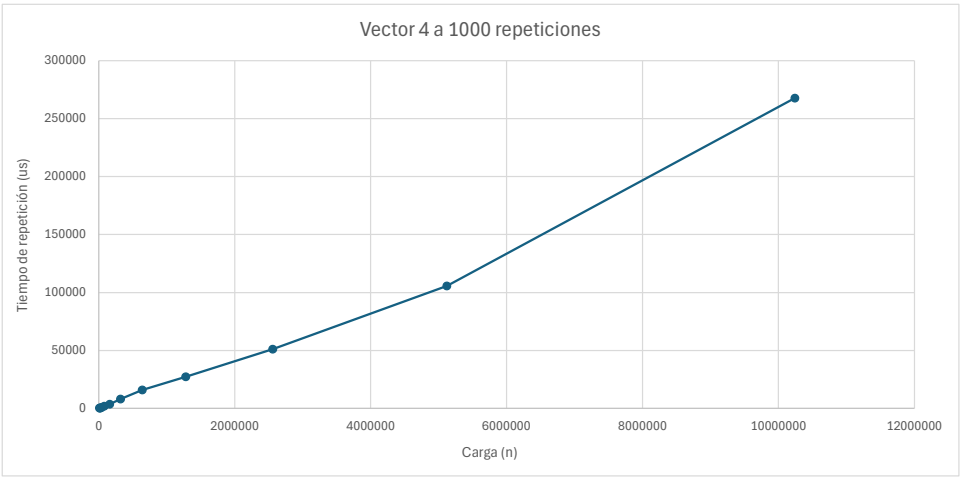
2. Toma de tiempos nulos en `Vector2`

Para cargas suficientemente bajas, el tiempo transcurrido para la operación es inferior al milisegundo. Por tanto, al restar los resultados de `getCurrentTimeMillis()` antes y después de llamar al método se obtendrá 0 como resultado.

La carga más baja para la que he podido tomar una medición fiable ha sido la siguiente:

```
t1=1770804162055  *** t2=1770804162106
n= 2000000 Tiempo metodo suma = 51
Resultado de la suma de elementos = 39692
```

3. Toma de tiempos `Vector4`



Toma de tiempos para Vector4

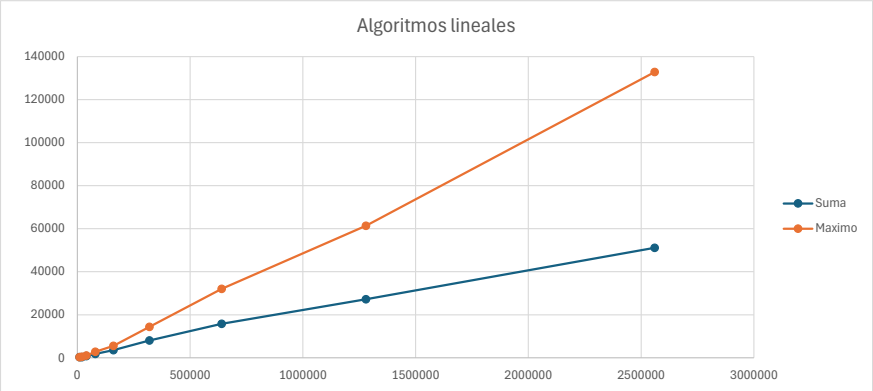
El tiempo de respuesta para cada repetición parece seguir una correspondencia lineal con la carga, pues se duplica también (de manera aproximada).

4. Suma vs Máximo

Tras ejecutar `Vector4` y `Vector5` con 1000 repeticiones cada uno, estos han sido los tiempos resultantes (en μs):

n	$Tsuma$	$Tmaximo$
10.000	269	337
20.000	334	430
40.000	799	1.049
80.000	1.837	2.779
160.000	3.568	5.513
320.000	8.031	14.411
640.000	15.815	32.013
1.280.000	27.232	61.351
2.560.000	51.124	132.819
5.120.000	105.653	FdT
10.240.000	FdT	FdT
20.480.000	FdT	FdT
40.960.000	FdT	FdT
81.920.000	FdT	FdT

Aunque el algoritmo "máximo" tiene tiempos más altos, ambos presentan un comportamiento lineal.



Comparativa gráfica entre los tiempos de `suma` y `maximo`

5. Coincidencias 1 vs Coincidencias 2

n	*Coincidencias 1(ms)*	*Coincidencias 2(ms)*
10.000	1.161	0,42
20.000	4.955	0,54
40.000	18.178	1,16
80.000	73.149	3,061
160.000	291.764	4,841
320.000	FdT	11,926
640.000	FdT	20,15
1.280.000	FdT	43,668
2.560.000	FdT	89,01
5.120.000	FdT	180,352

Debido al elevado coste temporal de ejecutar `Coincidencias1`, las tomas de tiempos de este algoritmo se hicieron con una única repetición. Las de `Coincidencias2`; en cambio, se tomaron como promedio de 1000 repeticiones (de ahí los decimales al hacer la media).

6. Comparación con tiempos teóricos

Los algoritmos `suma` y `maximo` iteran por cada elemento del array, realizando operaciones de tiempo constante en cada paso. Por tanto, tienen complejidad $O(n)$. Veamos si el cociente entre los tiempos empíricos y n tiende a una constante no nula.

n	T_{Suma}/n	T_{Max}/n
10.000	0,0269	0,0337
20.000	0,0167	0,0215
40.000	0,0200	0,0262
80.000	0,0230	0,0347
160.000	0,0223	0,0345
320.000	0,0251	0,0450
640.000	0,0247	0,0500
1.280.000	0,0213	0,0479
2.560.000	0,0200	0,0519
5.120.000	0,0206	-

Aun habiendo modificado n en gran medida, los cocientes parecen mantenerse estables, sobre todo para `suma`. Los cocientes d máximo, aunque lo logran, tardan algo más en estabilizarse.

Veamos ahora el la comparativa para `coincidencias1` y `coincidencias2`. El primero de ellos, itera sobre el primer vector y en cada paso recorre el segundo vector al completo. Si n es el tamaño de cada vector (ambos de igual tamaño), la complejidad del algoritmo es por tanto $O(n^2)$. En cuanto a `coincidencias2`, itera sobre ambos vectores simultáneamente utilizando una única variable índice, luego su complejidad es $O(n)$.

n	n^2/T_1	n/T_2
10.000	86132,64427	23809,52381
20.000	80726,53885	37037,03704
40.000	88018,48388	34482,75862
80.000	87492,65198	26135,24992
160.000	87742,14776	33051,02252
320.000	-	26832,13148
640.000	-	31761,7866
1.280.000	-	29312,08207
2.560.000	-	28760,81339
5.120.000	-	28388,92832

Por legibilidad, hemos optado por dividir $\frac{f_i(n)}{T_i}$ (el resultado sigue siendo válido).

7. Condiciones de medida y características del ordenador

Todas las mediciones han sido realizadas en un mismo ordenador con Eclipse IDE usando el argumento `-Xint` para la máquina virtual de java.

- CPU: Intel Core i7-6700HQ CPU @ 2.60GHz
- Memoria: 16GB RAM