PRÁCTICA 0

ACTIVIDAD 1: POTENCIA DE LAS CPUs

PC CASA: Intel ® Core(TM) i5-8250U CPU @ 1.60Ghz 1.80 GHz, MEMORIA: 8.00 GB
PC LAB: Intel(R) Core (™) 2 Duo CPU E7600 @ 3.06 GHz 3.06GHz,MEMORIA: 4.00 GB

#	CPU	milisegundos	1-Core (max)	Operaciones (aprox)
1	i7-4790	216	118	25488
2	i5-7600	205	126	25830
3	i3-3220	267	90.6	24190.2
4	Core2 Duo E7600	375	55.9	20962.5
5	i5-8250U	235	105	24675

¿Crees que podrías mezclar valores de diferentes CPUs en un mismo estudio analítico de los tiempos de ejecución de un algoritmo?

- No se pueden mezclar valores de diferentes CPUs en un mismo estudio analítico de los tiempos de ejecución porque cada CPU ejecuta el algoritmo a un ritmo distinto.

ACTIVIDAD 2: INFLUENCIA DEL SISTEMA OPERATIVO

1. ¿Qué plan de energía crees que es el más adecuado para realizar mediciones?

- Para realizar mediciones el plan de energía más adecuado es el de máxima energía porque las mediciones son más rápidas.

2. Si tuvieses que realizar la medición de un experimento muy largo, ¿podrías utilizar el ordenador para por ejemplo ver un vídeo de YouTube?

- Para realizar mediciones es imprescindible que no se ejecute otra tarea pues esta influiría en el rendimiento y las mediciones serían erróneas.

3. ¿Crees conveniente realizar varias mediciones simultáneamente en el mismo ordenador?

- No, porque puede dar lugar a resultados erróneos. Cada medición debe realizarse de una en una para evitar que intervengan otros factores externos y den valores inexactos.

PRÁCTICA 1

Todas las mediciones se han realizado desde PC CASA.

2. TOMA DE TIEMPOS DE EJECUCIÓN

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

- Cómo utilizamos un long (8 bytes -> 64 bits) la cuenta para calcular el número de años en los que se podrá seguir utilizando esta forma de contar es:

$$(2^{63}) / (365\ dias\ *\ 24\ horas\ *3600\ s)\ =\ 2.92 * e^{11}$$

- También se puede calcular utilizando la función System.out.println(new Date(Long.MAX_VALUE)); que devuelve Sun Aug 17 03:12:55 GMT-04:00 292278994
- CONCLUSIÓN: Se puede utilizar durante algo más de 292 millones de años.

INT -> 4Bytes -> 32 bits 2^31 / (365*24*3600) = 0,068 ~ 6 meses

1. ¿Qué significa que el tiempo medido sea 0?

- Que el tiempo medido sea 0 indica que la ejecución del problema es muy rápida como para obtener tiempos representativos.

2. ¿A partir de qué tamaño de problema (n) empezamos a obtener tiempos fiables?

- Empezamos a obtener tiempos fiables (superiores a 50) a partir de $n = 10^8$.
- Con $n = 10^9$ tenemos un OutOfMemoryException, aunque con 10^8 en algunas ocasiones las mediciones son menores a 50.
- Un buen número para obtener tiempos fiables es $n = 2 * 10^8$.

3. CRECIMIENTO DEL TAMAÑO DEL PROBLEMA

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

- Cuando se empezaba a mostrar una complejidad lineal da un error por falta de espacio.

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

- No tenemos forma de saberlo porque ocurre un OutOfMemoryError cuando por fin empiezan a mostrarse tiempos de ejecución.

n	t suma (ms)	t max (ms)
10	0,000016	0,000015
30	0,000016	0,000016
90	0,000015	0,000031
270	0,000078	0,000063
810	0,000235	0,000203
2430	0,000718	0,000546
7290	0,002157	0,001672
21870	0,006578	0,005187
65610	0,019513	0,015528
196830	0,060113	0,047879
590490	0,175428	0,145182
1771470	0,57924	0,532891
5390490	1,75413	1,86972
15943230	5,25861	5,6237
47829690	15,5478	16,9257

4. TOMA DE TIEMPOS DE EJECUCIÓN PEQUEÑOS (<50 ms)

¿Cumplen los valores obtenidos con lo esperado?

- Se puede observar cómo cumple con una complejidad lineal de 3, ya que los valores temporales se van triplicando en cada iteración.

n	t diagonal1 (ms)	t diagonal2 (ms)
3	0,000032	0,000015
6	0,000046	0,000016
12	0,000219	0,000016
24	0,000609	0,000016
48	0,002761	0,000062
96	0,011218	0,000109
192	0,042524	0,000203
384	0,154651	0,000625
768	0,53175	0,001609
1536	2,08498	0,012248
3072	9,6993	0,032864
6144	24,0967	0,094467

6. OPERACIONES SOBRE MATRICES

¿Cumplen los valores obtenidos con lo esperado?

- Los valores obtenidos cumplen con lo esperado porque Diagonal1 tiene una complejidad cuadrática y Diagonal2 tiene una complejidad lineal, por lo que Diagonal2 es mucho más eficiente.

8. BENCHMARKING: INFLUENCIA DE LA PLATAFORMA

```
<terminated> TiemposJava (1) [Java Application] C:\Program Files\Ac
TIEMPOS LINEALES EN JAVA (MILISEG.)
CONTADOR=1000000  n=1000000  TIEMPO=0
CONTADOR=2000000  n=2000000  TIEMPO=0
CONTADOR=4000000  n=4000000  TIEMPO=0
CONTADOR=8000000  n=8000000  TIEMPO=0
CONTADOR=16000000  n=16000000  TIEMPO=15
CONTADOR=32000000  n=32000000  TIEMPO=16
CONTADOR=64000000  n=64000000  TIEMPO=16
CONTADOR=128000000  n=128000000  TIEMPO=62
CONTADOR=256000000  n=256000000  TIEMPO=111
CONTADOR=512000000  n=512000000  TIEMPO=203
CONTADOR=1024000000  n=1024000000  TIEMPO=390
CONTADOR=2048000000  n=2048000000  TIEMPO=754
CONTADOR=4096000000  n=4096000000  TIEMPO=1514
CONTADOR=8192000000  n=8192000000  TIEMPO=3047
CONTADOR=16384000000  n=16384000000  TIEMPO=6219
TIEMPOS CUADRATICOS EN JAVA(MILISEG.)
CONTADOR=10000  n=100  TIEMPO=14
CONTADOR=40000  n=200  TIEMPO=0
CONTADOR=160000  n=400  TIEMPO=1
CONTADOR=640000  n=800  TIEMPO=2
CONTADOR=2560000  n=1600  TIEMPO=0
CONTADOR=10240000  n=3200  TIEMPO=0
CONTADOR=40960000  n=6400  TIEMPO=15
CONTADOR=163840000  n=12800  TIEMPO=78
CONTADOR=655360000  n=25600  TIEMPO=235
CONTADOR=2621440000  n=51200  TIEMPO=993
CONTADOR=10485760000  n=102400  TIEMPO=3913
CONTADOR=41943040000  n=204800  TIEMPO=15556
```

```
[Running] python -u "d:\alg77777777\p1\TiemposPython.py"
TIEMPOS LINEALES (MILISEG.)
CONTADOR 1000000 n= 1000000 *** tiempo 48
CONTADOR 2000000 n= 2000000 *** tiempo 95
CONTADOR 4000000 n= 4000000 *** tiempo 193
CONTADOR 8000000 n= 8000000 *** tiempo 388
CONTADOR 16000000 n= 16000000 *** tiempo 758
CONTADOR 32000000 n= 32000000 *** tiempo 1524
CONTADOR 64000000 n= 64000000 *** tiempo 3056
CONTADOR 128000000 n= 128000000 *** tiempo 6037
TIEMPOS CUADR?TICOS (MILISEG.
CONTADOR 10000 n= 100 *** tiempo 0
CONTADOR 40000 n= 200 *** tiempo 0
CONTADOR 160000 n= 400 *** tiempo 7
CONTADOR 640000 n= 800 *** tiempo 28
CONTADOR 2560000 n= 1600 *** tiempo 118
CONTADOR 10240000 n= 3200 *** tiempo 491
CONTADOR 40960000 n= 6400 *** tiempo 1912
CONTADOR 163840000 n= 12800 *** tiempo 7772

[Done] exited with code=0 in 22.849 seconds
```

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

- Las diferencias de tiempos en la ejecución entre Java y Python se deben a que Java es un lenguaje compilable y Python es un intérprete. Es decir, Java lo traduce una vez a código intermedio y luego trabaja con este código mientras que Python lo pasa a código máquina y lo ejecuta en cada llamada, lo que aumenta su tiempo de ejecución.

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

- Existe una analogía en el comportamiento de las dos implementaciones ya que la complejidad teórica es inherente del algoritmo. Ambos programas respetan esa complejidad, que es independiente del entorno.

PRÁCTICA 2

Todas las mediciones se han realizado desde PC CASA.

Tabla 1. Dos algoritmos con misma complejidad:

n	t bucle2 (ms)	t bucle3 (ms)	t bucle2/t bucle3 (ms)
8	0,000047	0,000016	2,9375
16	0,000078	0,000047	1,659574468
32	0,000219	0,000093	2,35483871
64	0,000593	0,000266	2,229323308
128	0,001795	0,000937	1,915688367
256	0,006904	0,00432	1,598148148
512	0,021851	0,01543	1,416137395
1024	0,087162	0,0514	1,695758755
2048	0,329859	0,181287	1,819540287
4096	1,2825	0,67366	1,903779355
8192	5,0526	2,66884	1,893182057
16384	18,76	10,1257	1,852711417
32768	73,569	40,162	1,831806185
65536	277,009	159,563	1,736047831

Aunque ambos algoritmos tienen una complejidad O(n^2) se puede apreciar que bucle3 es más eficiente que bucle2 (el resultado de la división es superior a 1)

Tabla 2. Dos algoritmos con distinta complejidad:

N	t bucle1 (ms)	t bucle2 (ms)	t bucle1/t bucle2 (ms)
8	0,000026	0,000047	0,553191489
16	0,000051	0,000078	0,653846154
32	0,000123	0,000219	0,561643836
64	0,000277	0,000593	0,467116358
128	0,000621	0,001795	0,345961003
256	0,001388	0,006904	0,201042874
512	0,003091	0,021851	0,141458057
1024	0,006776	0,087162	0,0777403
2048	0,014728	0,329859	0,04464938
4096	0,032138	1,2825	0,025058869
8192	0,068999	5,0526	0,013656137
16384	0,14834	18,76	0,007907249
32768	0,3143	73,569	0,00427218
65536	0,66847	277,009	0,002413171

Se aprecia que bucle1 es infinitamente más eficiente que bucle2 porque la división tiende a 0. Bucle1 tiene una complejidad O($n\ logn$) y bucle2 tiene una complejidad O(n^2)

Tabla 3. Complejidad del resto de los algoritmos

Incógnita tiene una complejidad O(n^3).

N	t bucle4 (ms)	t bucle5 (ms)	t incognita (ms)
8	0,002249	0,000435	0,000156
16	0,002686	0,002704	0,000764
32	0,2513	0,015946	0,005231
64	1,325	0,110591	0,040891
128	16,5389	1,04505	0,21889
256	221,778	7,54922	2,0795
512	4192	57,968	8,849
1024	64466	542,757	66,226
2048	940962	4178	561,474