Miguel Méndez Murias
UO287687 - Algoritmia

Práctica 0

Actividad 1: Potencia de las CPUs

Tarea 1:

Datos de mi sistema:

- Modelo del procesador: 12th Gen Intel(R) Core(TM) i7-12700H 2.70 GHz
- Memoria RAM : 16 GB

Datos sobre el modelo del procesador:

- Índice medio de operaciones enteras y reales por unidad de tiempo:
  - Un núcleo: 182
  - Dos núcleos: 350
  - Cuatro núcleos: 661
  - Ocho núcleos: 1096
  - 64 núcleos (solo tiene 14): 1778

Tiempo que ha tardado en finalizar el programa Benchmarking1: 176 ms

A partir del tiempo que ha tardado en ejecutarse el programa Benchmarking1 y el valor SC mix avg para nuestro procesador podemos calcular el índice aproximado de operaciones enteras/reales que necesitó el programa.
Se calcula de la siguiente forma:

$$Indice = SC\ Mix\ Avg * tiempo_{Benchmarking1} = 182 * 176\,ms = 32\,032$$

Tarea 2:

En esta tarea hay que recabar información sobre el SC Mix Avg y obtener el índice del número de operaciones enteras/reales a partir de los tiempos de ejecución del programa Benchamarking1 para distintas CPUs.
A continuación, se muestra una tabla con todos los datos recabados:

#	CPU	Milisegundos	SC Mix (avg)	Operaciones (aprox)
1	i7-4500U	285	63,9	18211,5
2	I3-3220	267	71,8	19170,6
3	I5-4590	219	90,6	19841,4
4	I7-4790	207	98,4	20368,8
5	Intel Pentium Gold G5400	215	99,2	21328
6	i7-12700H	176	182	32032
7	i5-10400F	195	118	23010

Conclusión:
Viendo los resultados en milisegundos, ¿crees que podrías mezclar valores de diferentes CPUs

en un mismo estudio analítico de los tiempos de ejecución de un algoritmo?
No, ya que la diferencia de operaciones entre diferentes CPUs es muy elevado, llegando casi a duplicarse entre el i7-4500U y el i7-12700H, por lo que los resultados que se obtendrían de hacer estudios analíticos con distintas CPUs para un mismo algoritmo sería muy poco fiables y precisos.

Actividad 2: Influencia del sistema operativo

Conclusiones
1. ¿Qué plan de energía crees que es el más adecuado para realizar mediciones?
El plan más adecuado es el de alto rendimiento, ya que es en el que menos se limita el rendimiento de la CPU, por lo que las mediciones son más fiables al tener una mayor capacidad de encargarse de otras tareas que realiza el sistema. Por ello, se puede decir que el modo de alto rendimiento es el que nos ofrece unas mediciones más precisas y fiables.
2. Si tuvieses que realizar la medición de un experimento muy largo, ¿podrías utilizar el ordenador para por ejemplo ver un vídeo de YouTube?
No sería buena idea porque la CPU tendría que encargarse de ejecutar el programa sobre el que se está realizando la medición al mismo tiempo que se encarga de mostrar el video de YouTube, lo que hace que los resultados no sean fiables. Esto se debe a que la CPU no estaría usando los máximos recursos posibles para realizar la medición, que sería lo ideal.
3. ¿Crees conveniente realizar varias mediciones simultáneamente en el mismo ordenador?
No es conveniente realizar varias mediciones simultáneamente en el mismo ordenador, ya que la CPU, al tener que repartir recursos entre varias programas, no estaría ejecutando cada programa de la forma idónea para medir, por lo que nos darían resultados irreales que no se corresponden con el rendimiento real. Esto se puede comprobar al ejecutar el programa cpuburn al mismo tiempo que hacemos la medición del programa Benchmarking1, ya que el tiempo aumenta en gran medida, pasando de 176 ms hasta llegar a más de 250 ms (además, se produce una gran variabilidad en los tiempos), por lo que podemos decir que las mediciones al ejecutar varios programas simultáneamente no son fiables.

Miguel Méndez Murias
UO287687 - Algoritmia

Práctica 1.1

Toma de tiempos de ejecución

Teniendo en cuenta que para medir los tiempos de ejecución se utiliza el método currentTimeMillis() que devuelve un long (64 bits) y el contador se inició en 1970 ¿Cuántos años más podremos seguir utilizando esta forma de contar?
Sabiendo que almacena un número de 64 bits, el número máximo que puede almacenar es 2^63 – 1, lo que se corresponde con 9,22 * 10^18 de milisegundos. Si pasamos este dato a años nos daría como resultado más de 292 millones de años, por lo que no tenemos que preocuparnos por el tiempo máximo que puede medir, ya que ese tiempo es muy superior incluso a la existencia de los seres humanos.

1. ¿Qué significa que el tiempo medido sea 0?
Significa que el valor del tiempo es medido es inferior a 1 milisegundo, que es la precisión máxima que nos da el método utilizado (currentTimeMillis()).

2. ¿A partir de qué tamaño de problema (n) empezamos a obtener tiempos fiables?
Teniendo en cuenta que consideramos un tiempo fiable a partir de 50 ms, para el caso del ordenador con el procesador Intel Core i7 12700H utilizado en la práctica anterior, se puede considerar fiable cualquier tamaño de n por encima de 160000000, que es el tamaño a partir del cuál el tiempo al ejecutar el programa Vector2 es superior a 50 milisegundos.

Crecimiento del tamaño del problema

Para esta parte de la práctica, realizamos mediciones con dos algoritmos diferentes, uno que se encargaba de hacer una suma y otro de obtener el máximo. El objetivo de esto, era calcular la complejidad de los algoritmos y ver como se respetaba la proporción en el aumento del tiempo empleado al aumentar el tamaño del problema. Además, tuvimos que ejecutar cada algoritmo con varias repeticiones, ya que las mediciones se hacían en milisegundos, por lo que no se podían apreciar valores del tiempo suficientemente altos como para poder considerarlos válidos. Es por esta razón, que nos basamos en la tabla siguiente para poder conseguir valores mucho más fiables:

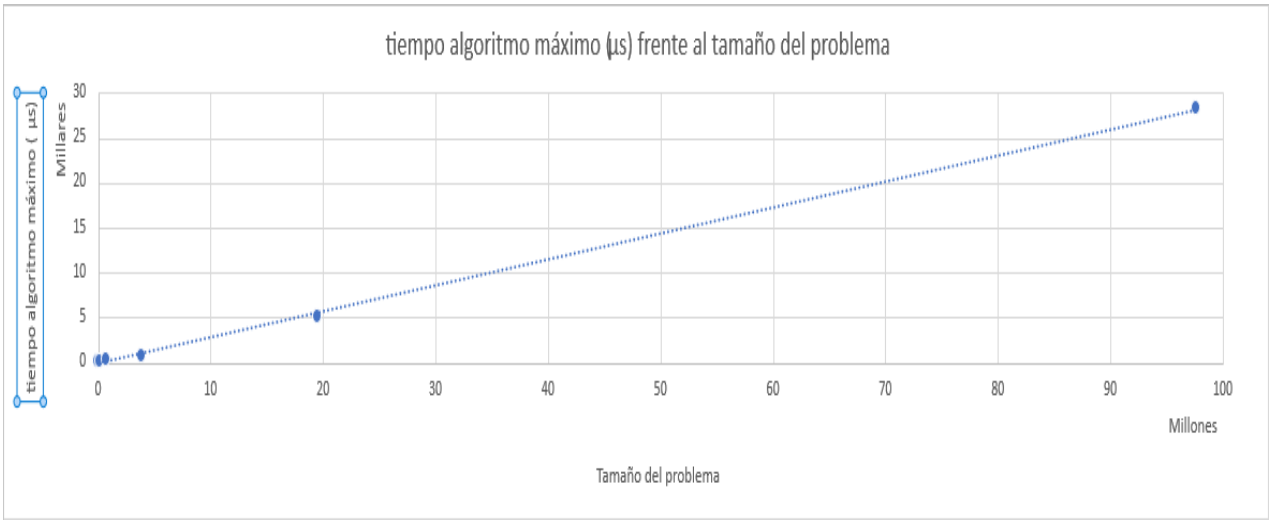
Repeticiones	*Unidades de tiempo en:*
1	**Milisegundos (10^{-3} s)**
10	decimas de ms (10^{-4} s)
100	centésimas de ms (10^{-5} s)
1 000	**microsegundos (µs) (10^{-6} s)**
10 000	decimas de µs (10^{-7} s)
100 000	centésimas de µs (10^{-8} s)
1 000 000	**Nanosegundos (10^{-9} s)**
.....	.....
10^9	**Picosegundos (10^{-12} s)**

Miguel Méndez Murias
UO287687 - Algoritmia

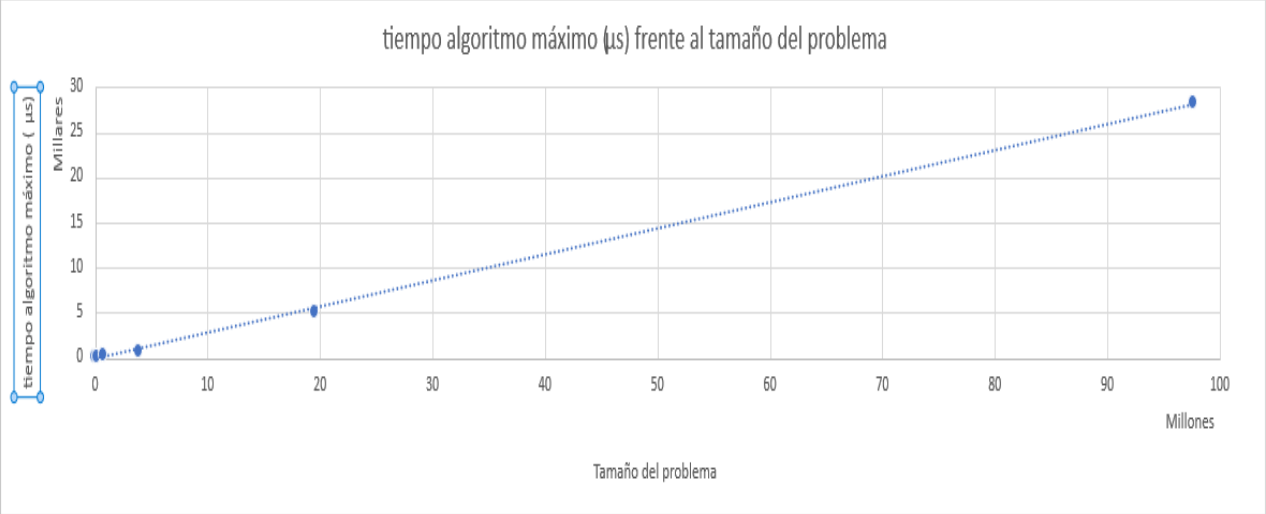
Además, al aumentar mucho el número de repeticiones, para tamaños de problema muy elevados el tiempo era demasiado grande, por lo que se decidió cambiar el número de repeticiones a partir de un cierto tamaño del problema y posteriormente ajustar el tiempo para que estuvieran en la misma escala y de esta forma se pudiesen comparar.
Los resultados de medir ambos algoritmos fueron los siguientes:

Tamaño del problema	Número de repeticiones	Tiempo algoritmo suma (µs)	Tiempo algoritmo máximo (µs)
10	100000000	0,0017	0,00312
50	100000000	0,00663	0,01458
250	100000000	0,04416	0,05774
1250	100000000	0,27204	0,20644
6250	100000000	1,40025	1,02512
31250	1000000	6,9	4,58
156250	1000000	34,952	23,63
781250	1000000	181,602	125,372
3906250	1000	981	604
19531250	1000	5507	4956

A partir de estos datos, comparamos los tiempos de cada algoritmo con el tamaño del problema y realizamos un gráfico con esta comparación:



Miguel Méndez Murias
UO287687 - Algoritmia



Una vez que tenemos los gráficos creados podemos garantizar que los algoritmos cumplen una complejidad O(n), ya que al aumentar el tamaño del problema, el tiempo empleado aumenta el misma proporción.

*Todas las mediciones han sido realizadas con un ordenador de las siguientes características:

- Procesador: 12th Gen Intel(R) Core(TM) i7-12700H 2.70 GHz
- Memoria RAM: 16GB

¿Cumplen los valores obtenidos con lo esperado?

Sí, ya que se esperaba que ambos algoritmos siguiesen una relación directamente proporcional entre el tiempo de ejecución y el tamaño del problema, lo cuál se cumple.

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

El tiempo aumenta de forma proporcional al tamaño del problema, es decir, los tiempos también se multiplican por 5. Esto es así, ya que el problema tiene una complejidad O(n).

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

Sí, ya que los tiempos aumentan en la misma proporción al tamaño del problema.

Operaciones con matrices

Posteriormente, nos encargamos de medir el tiempo de ejecución de dos algoritmos para sumar los elementos de la diagonal de matrices.
El primer algoritmo, recorría todos los elementos de la matriz, pero solo sumaba los elementos de la diagonal, mientras que el segundo solo recorría y sumaba los elementos de la diagonal.

Además, como había bastante diferencia en los tiempos para distintos tamaños del problema, realizamos las mediciones con distintas repeticiones para obtener los tiempos en distintas unidades. De esta forma, conseguimos ahorrar tiempo al medir y lo único que tuvimos que hacer para poder comparar los valores era poner todo en las mismas unidades (en este caso, pasamos todos los valores a nanosegundos).

Los resultados obtenidos fueron los siguientes:

Miguel Méndez Murias
UO287687 - Algoritmia

Tamaño problema	Tiempo diagonal 1(ns)	Tiempo diagonal 2(ns)
3	5,7	2,02
6	22,82	3,3
12	112,1	5,38
24	248	9,93
48	1023	18,49
96	3585	36,27
192	11620	74,27
384	37644	222,72
768	129627	630,78
1536	589328	1766
3072	1470000	21757
6144	6378000	63429
12288	33635000	143300

Miguel Méndez Murias
UO287687 - Algoritmia

*Todas las mediciones han sido realizadas con un ordenador de las siguientes características:

- Procesador: 12th Gen Intel(R) Core(TM) i7-12700H 2.70 GHz
- Memoria RAM: 16GB

¿Cumplen los valores obtenidos con lo esperado?
Era de esperar que el tiempo de la primera forma de calcular la suma de la diagonal fuera aumentando de forma cuadrática (complejidad $O(n^2)$), ya que se tenían que recorrer todas las filas y columnas. Por el contrario, el tiempo de la segunda forma de calcular la diagonal era de esperar que aumentase de forma lineal con el tamaño del problema (complejidad $O(n)$). Tras realizar las mediciones y comparar los datos, podemos garantizar que se cumplen las complejidades y con ello los valores esperados.
Sin embargo, como se puede observar en la tabla y en la gráfica, al pasar de un tamaño del problema de 1536 a 3072 se produce una variación muy grande que no se corresponde con la complejidad real, aunque para el resto de valores si que se cumple la complejidad (como se puede ver en la gráfica, es una línea recta). Aunque si que es necesario remarcar ese caso, no es conocido por que se produce, pero se supone que será por operaciones propias de la cpu (por ejemplo: se producen muchos fallos de acceso a memoria) y no por el propio algoritmo.

Influencia de la plataforma

En esta parte de la práctica, comprobamos el comportamiento de dos algoritmos similares en distintas plataformas. Más en concreto, medimos el tiempo del programa Benchmarking1 en java y en python para poder ver similitudes y diferencias en su comportamientos.
Ambos programas tenían como condición de parada que el tiempo de ejecución para el tamaño del problema fuese mayor a cinco segundos. Teniendo esto en cuenta, el resultado obtenido fue el siguiente: el algoritmo ejecutado en java, aún siendo el mismo, tenía un mayor rendimiento ya que podía ejecutar tamaños del problema mucho mayores en el mismo tiempo que en python.

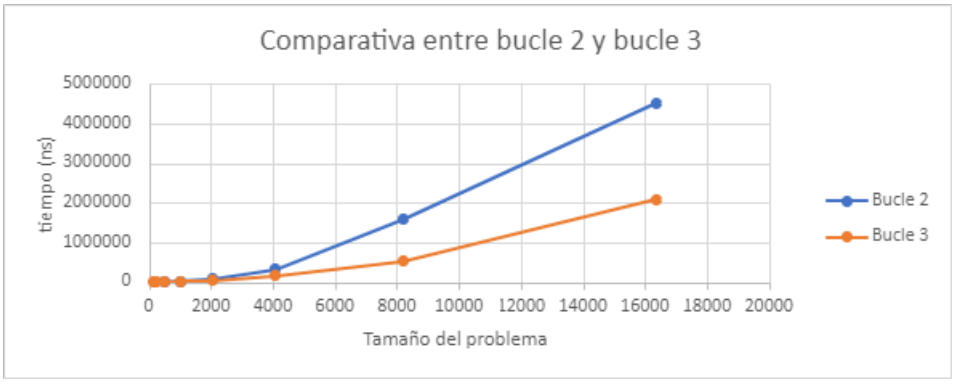
1. ¿A qué se deben las diferencias de tiempos en la ejecución entre uno y otro programa?
Las diferencias de tiempo de ejecución entre ambos programa se deben a que están desarrollados en distintos lenguajes de programación, cada uno con sus características: java es un lenguaje compilado, lo que permite ejecutar los programas de forma más rápida que en python, que es compilado.
2. Independientemente de los tiempos concretos ¿existe alguna analogía en el comportamiento de las dos implementaciones?
Sí, en ambos casos se cumplen las complejidades lineal y cuadrática, independientemente del lenguaje de programación utilizado.

Miguel Méndez Murias
UO287687 - Algoritmia

Práctica 1.2

Como primer ejercicio de la práctica, comparamos dos algoritmos con la misma complejidad, más concretamente con $O(n^2)$. Los resultados fueron los siguientes:

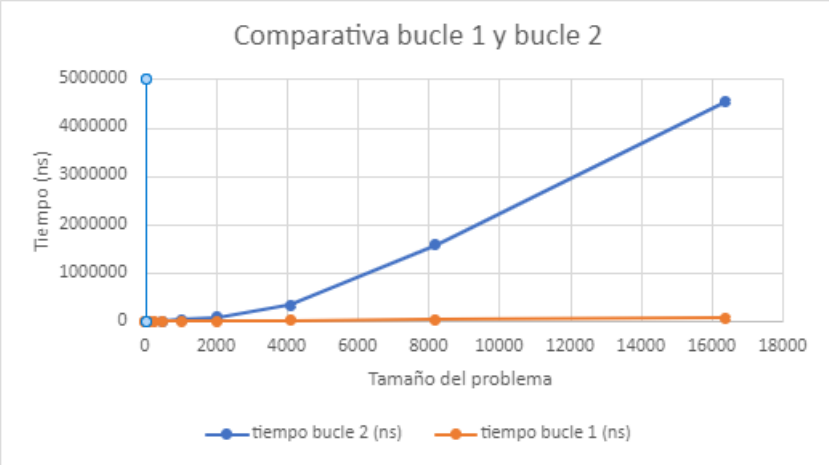
Tamaño del problema	Tiempo bucle 2 (ns)	Tiempo bucle 3 (ns)	Relación bucle 2/ bucle 3
8	-	-	-
16	-	-	-
32	-	-	-
64	-	-	-
128	-	560	-
256	-	1280	-
512	6840	3290	2,079027356
1024	26410	13070	2,020657995
2048	87770	44350	1,97903044
4096	329490	154370	2,134417309
8192	1583690	528820	2,994761923
16384	4526090	2094100	2,161353326



A continuación, comparamos dos algoritmos con distinta complejidad, uno de ellos con complejidad $O(n^2)$ y otro con complejidad $O(n*\log(n))$:

Tamaño del problema	Tiempo bucle 2 (ns)	Tiempo bucle 1 (ns)	Bucle1/bucle2
8	-	13,3	-
16	-	36,7	-
32	-	74,7	-
64	-	150	-
128	-	352	-
256	-	778	-
512	6840	1684	0,24619883
1024	26410	3623	0,137182885
2048	87770	7838	0,089301584
4096	329490	16242	0,049294364

8192	1583690	36509	0,023053123
16384	4526090	75830	0,016753975



Complejidad del resto de algoritmos

Como parte final de la práctica, realizamos dos algoritmos, uno que cumpliese una complejidad $O(n^4)$ y otro que cumpliese una complejidad $O(n^3*log(n))$. Por último, tuvimos que ejecutar un algoritmo del que desconocíamos la complejidad y adivinarla a partir de los tiempos que obtuviésemos de las mediciones.

Tamaño del problema	Tiempo bucle 4 (µs)	Tiempo bucle 5 (µs)	Incognita (µs)
8	0,213	0,194	0,091
16	1,697	1,155	0,51
32	16,077	9,858	3,398
64	149	97	24,836
128	1898	805	116
256	28755	6229	860
512	134326	49089	6359
1024	1957255	431256	46952
2048	30424784	3702783	362790
4096	509236148	32223429	2718651
Complejidad	$O(n^4)$	$O(n^3*log(n))$	$O(n^3)$

Miguel Méndez Murias
UO287687 - Algoritmia