PREGUNTAS SECCIÓN 1.1

Sesión práctica 2

Jonathan Arias Busto
UO283586
71780982-Y
Escuela de Ingeniería Informática - EII

ACTIVIDAD 1: Toma de tiempos

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

Siguiendo los siguientes calculos:

1. Cada año tiene 31536000000 mili segundos
2. Un dato de tipo long tiene un rango máximo de 9 223 372 036 854 775 807.

3. Si hacemos la división de long ente los mili segundos de un año nos sale que tenemos otros 292 Millones de años.

En conclusión, podremos seguir utilizando este sistema de contar tiempo durante 292 Millones de años más.

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

Significa que el algoritmo se ejecuta muy rápido, seguramente en micro segundos o mas bajo.

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

A partir de una n de: 250000000.
En este caso el tiempo medido es de 84 ms.

ACTIVIDAD 2: Crecimiento del tamaño del problema

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

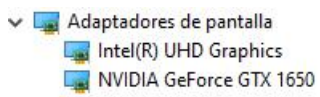
En el caso de la toma de tiempos para el algoritmo de suma, con complejidad O(n), teóricamente los tiempos deberían de multiplicarse por 5. Esto no ocurre en la realidad al 100%, pero si que sigue la tendencia linea que debe de seguir teniendo en cuenta un margen de error debido a la maquina en la que se ejecuta dicho algoritmo.

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

Si son los tiempos esperados de dicha complejidad, ya que como apuntaba en el anterior apartado sigue la tendencia lineal que debería, es decir sigue bastante bien la proporción de tiempo. Con esto me refiero a que mas o menos se van multiplicando los tiempos por 5 a la vez que se va multiplicando por 5 el tamaño del problema.

INFORMACIÓN SOBRE OREDENADOR USADO PARA MEDIR TIEMPOS

Procesador	Intel(R) Core(TM) i5-10300H CPU @ 2.50GHz 2.50 GHz
RAM instalada	16,0 GB (15,8 GB usable)



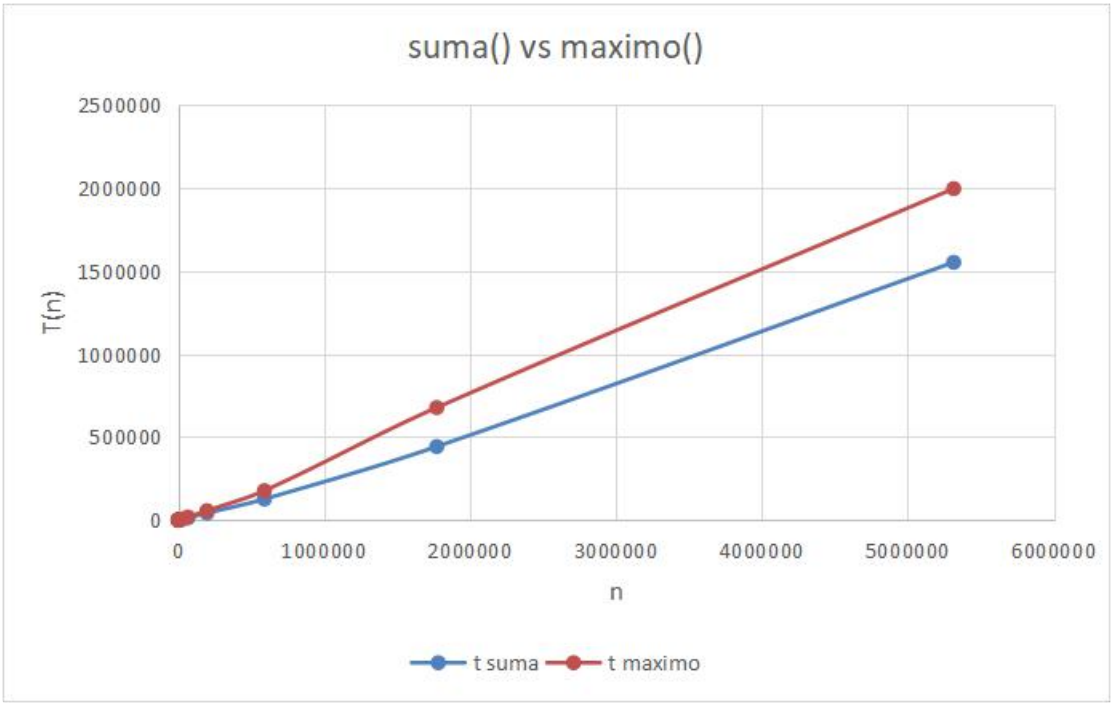
Se trata de un portátil con el modo de energía en “High performance”.

ACTIVIDAD 3: Toma de tiempos

La primera tabla consta de los tiempos de la clase Vector4.java y de la medición del algoritmo suma() y maximo().

n	t suma	t maximo
10	34	42
30	35	43
90	40	64
270	77	161
810	213	245
2430	535	656
7290	1575	1945
21870	4712	5821
65610	13939	18653
196830	42342	56744
590490	127082	177460
1771470	443184	677925
5314410	1553010	1996920
	unidad = ns	unidad = ns

La gráfica correspondiente es:



En esta gráfica se puede comprobar como en el caso del algoritmo de maximo() tarda mas tiempo en ejecutarse, ya que contiene una condición dentro del bucle principal del algoritmo que añade algo de complejidad al algoritmo en la práctica.
En el caso del algoritmo suma() sigue los pasos del anterior algoritmo pero no tiene esta condición dentro del bucle, por lo que tarda menos en ejecutarse.

Ambos algoritmos tienen una complejidad de O(n), pero en el caso del algoritmo máximo, a la hora de realizar la medición de tiempos, se puede apreciar que la condición afecta a estos resultados.

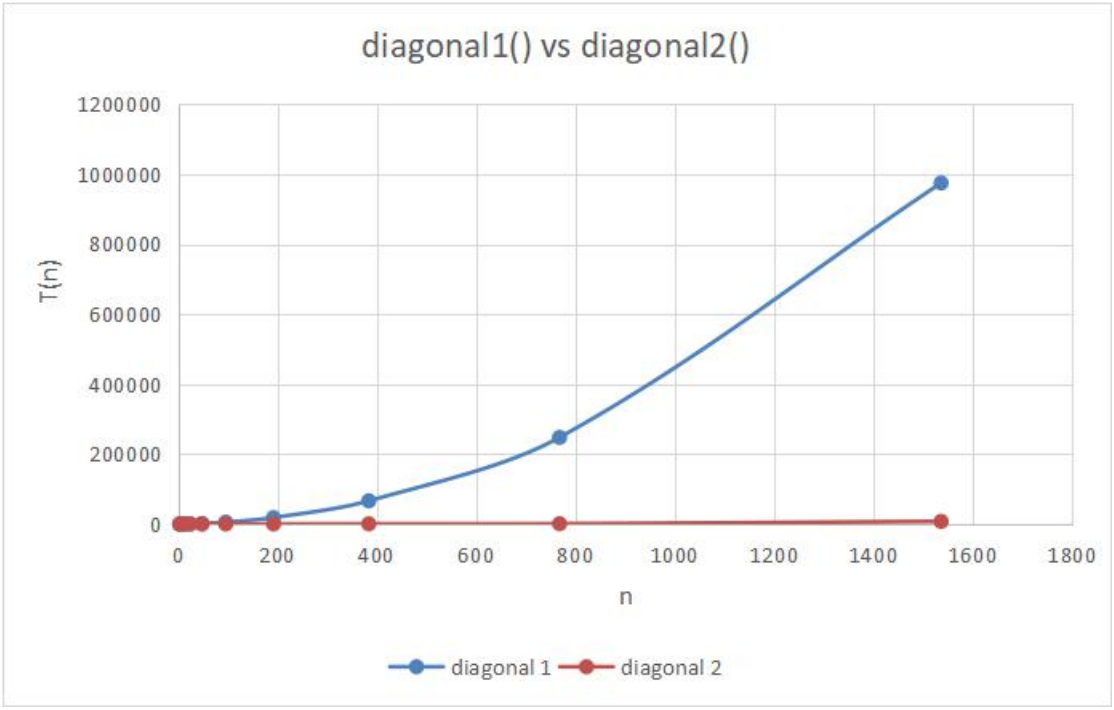
Se puede concluir que se obtienen los resultados esperados en ambos casos.

ACTIVIDAD 4: Operaciones sobre matrices

La segunda tabla consta de los tiempos de la clase MatrizOperacionesTiempo.java de la que vamos a medir la diferencia de tiempos entre una complejidad O(n) y una O(n^2). En la tabla se puede apreciar perfectamente la diferencia.

n	t diagonal(1)	t diagonal(2)
3	15	11
6	34	19
12	102	12
24	353	22
48	1444	57
96	5208	77
192	18260	134
384	66291	395
768	247633	821
1536	974976	7555
	unidad = ns	unidad = ns

La gráfica correspondiente a esta tabla es la siguiente:



Como se puede apreciar en el caso del algoritmo diagonal1() podemos comprobar como tiene un tendencia exponencial O(n^2).
En el caso del algoritmo diagonal2() la tendencia es mucho más lineal, de ahí que no se llegue a ver como sube en la gráfica, puesto que hay una diferencia enorme entre el algoritmo diagona1() y diagonal2().

Estos resultados son los esperados puesto que sabemos que la complejidad de los algoritmos diagonal1() y diagonal2() son O(n^2) y O(n) respectivamente.

ACTIVIDAD 5: Diferencia entre Python y Java

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

Las diferencias de tiempos son debido a la forma de interpretar el código del lenguaje. Python tiene una sintaxis mucho más ligera para el programador pero el interprete tiene que realizar muchas más comprobaciones que en el caso de Java. Por eso Python es bastante más lento que Java.

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

La forma en que se miden los tiempos, en ambos casos están dentro de un bucle while que se ejecuta hasta que una condición se cumpla. Y también los algoritmos como tal siguen la misma filosofía, es decir, un bucle for en el caso lineal y dos bucles for anidados en el caso cuadrático.