TAREA SESION 2 IKER NUÑEZ MEANA UO306425

0) Complejidades temporales de los algoritmos

Bucle 1-> 0(n * log n)
Bucle 2-> 0(n^2 * log n)
Bucle 3 -> 0(n^2 * log n)
Bucle 4 -> 0(n^3)

Luego, tendremos:
Bucle 5-> 0(n^2 * log^2(n))
Bucle 6 -> 0(n^3 * log n)
Bucle 7 -> 0(n^4)

1) Algunos modelos iterativos

A continuación, se pide medir los tiempos de los 4 primeros algoritmos. Para que el Bucle4 no se saliera fuera de tiempo, utilizamos un parámetro de repeticiones = 1, y aun, así como veremos las ultimas 3 n se salieron fuera de tiempo:

n	Bucle1	Bucle2	Bucle3	Bucle4
100	0	1	1	1
200	0	0	4	6
400	0	3	15	44
800	0	18	66	331
1600	0	61	262	2805
3200	1	242	1105	22250
6400	1	1092	4729	175379
12800	3	4608	19979	fdt
25600	6	18927	83250	fdt
51200	10	76024	343020	fdt

2) Creación de modelos iterativos de una complejidad dada

Se piden implementar otras 3 clases: Bucle5, Bucle6 y Bucle7 en Java, modificando las anteriores de manera que tengan la complejidad pedida arriba. Tras implementarlas, volvemos a medir los tiempos para repeticiones = 1:

n	Bucle5	Bucle6	Bucle7
100	1	10	45
200	4	91	582
400	18	805	15305
800	91	7020	265501
1600	444	137974	10148737
3200	2131	1207614	fdt
6400	10019	fdt	fdt
12800	46971	fdt	fdt
25600	211785	fdt	fdt
51200	fdt	fdt	fdt

Los resultados son los esperados, el tiempo va aumentando en los 3 bucles debido a que aumenta la complejidad.

3)Relación temporal de algoritmos

Caso 1: Algoritmos de distinta complejidad:

En este caso, como vamos a poner el de menor complejidad arriba, a medida que n crezca, el cociente debería aproximarse a 0, ya que denominador > numerador:

n	Bucle1	Bucle2	t1/t2
100	0	1	0
200	0	0	0
400	0	3	0
800	0	18	0
1600	0	61	0
3200	1	242	0,00413223
6400	1	1092	0,00091575
12800	3	4608	0,00065104
25600	6	18927	0,00031701
51200	10	76024	0,00013154

Efectivamente, los resultados son los esperados, tanto por la tendencia del cociente, como que los tiempos de bucle 1 son inferiores (bastante) a los de bucle 2.

Caso 2: Algoritmos de misma complejidad:

Medir tiempos es interesante por esto. A pesar de que dos algoritmos tengan la misma complejidad, pequeños detalles en la implementación pueden hacer que uno tarde mucho mas que otro. Por esta razón, es importante no solo mirar la complejidad, sino mirar también la estructura del código.
En este caso, vamos a poner el que mas tarda en el numerador, luego como numerador > denominador, el cociente debería tender a infinito (o en nuestro caso para pocas n crecer sin más):

n	Bucle3	Bucle2	t3/t2
100	1	1	1
200	4	0	4
400	15	3	5
800	66	18	3,66666667
1600	262	61	4,29508197
3200	1105	242	4,5661157
6400	4729	1092	4,33058608
12800	19979	4608	4,33572049
25600	83250	18927	4,39847836
51200	343020	76024	4,51199621

Efectivamente, los resultados son los esperados, tanto por el cociente, como por los tiempos de ambos algoritmos (t3 >> t2)

Caso 3: Mismo algoritmo en diferentes entornos de desarrollo:
Para terminar, vamos a comparar 2 entornos de ejecución: CPython y Java para ver cuál es más rápido ejecutando exactamente el mismo algoritmo.
Java debería ser más rápido, simplemente por ser compilado y no interpretado. Pero, además, en cada iteración se incrementa una variable contador que es un entero. Sin embargo, como Python no tiene tipos, deberá comprobar en cada iteración, de que tipo es la variable. Java no. Solo ahí habría una diferencia de tiempos.

n	Bucle4 Py	Bucle4 Java	tPython/tJava
100	6	1	6
200	50	6	8,33333333
400	321	44	7,29545455
800	2750	331	8,3081571
1600	31326	2805	11,1679144
3200	449582	22250	20,2059326
6400	fdt	175379	5,7019E-06

Los resultados son los esperados, pues java es mas rápido. El cociente también es razonable, pues va creciendo.