Informe práctica 1:

Uxío Merino Currás (uxio.merino@udc.es) y Mario Chan García (mario.chan@udc.es), grupo 1.3 GCEID

En esta practica estudiamos tres enfoques distintos de la secuencia de Fibonacci para medir sus tiempos de ejecución y realizar un análisis de dichos resultados. En primer lugar, tenemos una función recursiva (‘fib_recursive(n)’), a continuación una iterativa (‘fib_iterative(n)’), que se realiza a través de un bucle; y por último, una función más compleja que se ejecuta utilizando la fórmula de Binet (‘fib_binet(n)’).
Estas funciones se han probado utilizando el Sistema Operativo Windows 10 Home 20H2 x64 y el entorno Spyder (Phyton 3.8). Las especificaciones de la máquina son las siguientes:

Procesador: Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
Ram: 16,00 GB

Antes de mostrar las medidas obtenidas por nuestra máquina, cabe destacar que estas han sido recogidas tras realizar un bucle que ha repetido la función 1000 veces y posteriormente divididas entre el número de ejecuciones, ya que los tiempos de ejecución para varias ‘n’ eran menores a 500 microsegundos. De esta forma, las mediciones serán más fáciles de acotar y no tan imprecisas.

Tabla de resultados de la función recursiva (tiempo medido en nanosegundos):

n	t(n)	t(n)/math.pow(x,2)	t(n)/math.pow(1.6180,n)	t(n)/math.pow(2,n)
2	2	0	0.7640	0.5
4	2	0	0.2918	0.125
8	14	0	0.2981	0.055
16	1001	3	0.4537	0.0153
32	1697589	1657	0.3488	0.000395

Tabla de resultados de la función iterativa (tiempo medido en nanosegundos):

n	t(n)	t(n) / log(n)	t(n) / n	t(n) / n^2
2	1.9	2.7411	0.95	0.475
4	1.4	1.0099	0.35	0.0875
8	1.7	0.8175	0.2125	0.0266
16	3.4	1.2263	0.2125	0.0133
32	5.7	1.6447	0.178125	0.005566
64	11.0	2.6449	0.171875	0.00269
128	21.9	4.5137	0.17109	0.00134

Tabla de resultados de la función de la fórmula de Binet (tiempo medido en nanosegundos):

n	t(n)	t(n) / 1	t(n) / n
2	5.1	5.1	2.55
4	2.4	2.4	0.6
8	2.2	2.2	0.275
16	2.1	2.1	0.13125
32	2.1	2.1	0.0656
64	2.1	2.1	0.03281
128	2.1	2.1	0.01640

Conclusiones:

En primer lugar, en la versión recursiva, las cotas que han sido proporcionadas se comprueba que son correctas: la cota subestimada tiende a infinito a medida que la n también crece, la cota óptima (aunque no se puede observar exactamente que tienda a una

constante concreta) oscila entre unos valores muy pequeños, por lo que asumimos que si tuviésemos más medidas acabaría por aproximarse a una constante; y la cota sobreestimada se observa cómo decrece rápidamente a medida que la n crece, por lo que decimos que tiende a 0. Como las tres cotas proporcionadas cumplen lo que deberían, aseguramos que las tres son correctas.

En segundo lugar, en el algoritmo iterativo, se nos pedía a nosotros estimar las tres cotas: la subestimada, la óptima, y la sobreestimada. Nuestras respectivas estimaciones han sido $\Theta(\log(n)$, $\Theta(n)$ y $\Theta(n^2)$. Podemos comprobar que estas son correctas de igual manera que en el anterior algoritmo: la cota subestimada, a partir de la tercera medida, crece rápidamente a medida que la ‘n’ tiende a infinito; la cota ajustada oscila entre valores muy pequeños (0.17109 y 0.2125, también a partir de la tercera medida), por lo que también tenderá a una constante entre esos valores si la ‘n’ tendiera a infinito; y la cota sobreestimada tiende a 0 a medida que crece la ‘n’, lo cuál es fácilmente comprobable viendo lo rápido que descienden los valores de la cota a medida que aumenta la ‘n’.

Por último, en la versión que se ejecuta a través de la fórmula de Binet, simplemente se nos pedía una cota ajustada y una sobreestimada, y nuestras respectivas estimaciones han sido $\Theta(1)$ y $\Theta(n)$. Para comprobarlas, recurrimos al mismo método que para los dos algoritmos anteriores: la cota ajustada se estabiliza en el valor 2.1 a medida que la ‘n’ tiende a infinito, por lo que la constante a la que tiende es ese valor específico; mientras que la cota sobreestimada vuelve a tender a 0 cuando la ‘n’ aumenta su valor, decreciendo rápidamente.