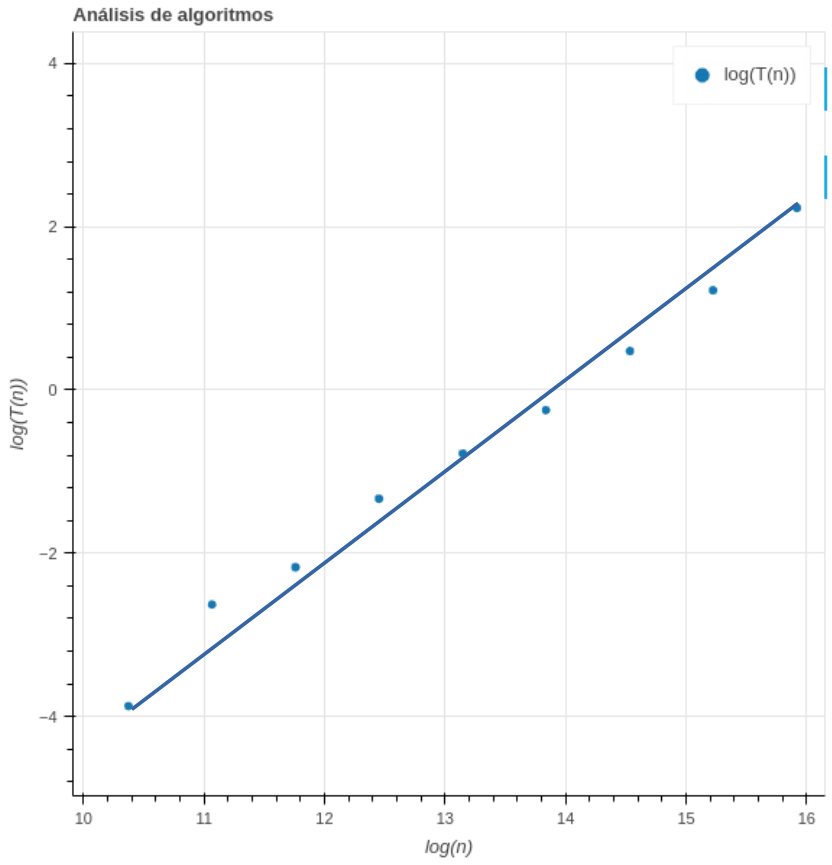
MEMORIA DEL ANÁLISIS DEL MÉTODO MULTIPLY

Método Experimental

Para obtener unas mediciones precisas y algo más sofisticadas acerca del método multiply, utilizaremos el código de **DoublingTest** creando un Objeto Int = 42 y un array de n objetos Int que comenzará con n = 32000 objetos Int (tamaño del array inicial), cuyos valores son números aleatorios entre 100000000 y 1000000000. Los arrays doblarán su tamaño en cada iteración del bucle del programa principal. Ejecutando el código en nuestro ordenador hemos obtenido las siguientes medidas:

n	T(n)
32000	0.02075763500
64000	0.07236582500
128000	0.11399450000
256000	0.26386930100
512000	0.45855769500
1024000	0.78123839600
2048000	1.61042002200
4096000	3.38990205000
8192000	9.30737568800

Con estos datos obtenemos la siguiente grafica representando lg(T(n)) en el de ordenadas:



Colocando una recta sobre el primer y el ultimo punto podemos observar que todos los puntos están próximos a la recta de una pendiente 2 aproximadamente. De la gráfica anterior tenemos la siguiente función:

$lg(T(n)) = 2lg(n) - lg(a)$

Si despejamos T(n), obtenemos:

$T(n) = n^2/a$

Para obtener el valor de la constante "a", podemos sustituir para valores experimentales n = 32000 y T(n) = 0,021. Obtenemos $a = 4{,}93x10^{10}$. Dicha constante solo es valida para el procesador en el que se tomaron las medidas, pues la constante será distinta dependiendo de la máquina en la que se tomen las pruebas. Con la constante anterior, podemos observar que para n = 128000, obtenemos $T(128000) = 128000^2 / 4{,}93x10^{10} = 0.3 \sim 0.26$ valor bastante parecido al que obtuvimos en los resultados experimentales.

Podremos decir que el algoritmo tiene un orden de crecimiento cuadrático, n^2.

Otra técnica para predecir el orden de crecimiento del método multiply sería usando el programa DoublingRatio, del cual obtendremos a parte del tiempo que tarda multiply en ejecutarse, un ratio que nos ayudará a definir el orden de crecimiento. Ejecutando DoublingRatio obtenemos las siguientes medidas para este ordenador:

n	T(n)	ratio
250	0.00187845200	0.05671244143
500	0.00180934100	0.96320853554
1000	0.00311801000	1.72328488660
2000	0.00663847400	2.12907399271
4000	0.01269269700	1.91199016521
8000	0.02727987600	2.14925764004
16000	0.04560366200	1.67169608835
32000	0.04822373600	1.05745314927
64000	0.08842410300	1.83362199478
128000	0.11286999000	1.27646180363
256000	0.24389361700	2.16083670248
512000	0.43565541800	1.78625182306
1024000	0.81582337700	1.87263452557
2048000	1.61399760600	1.97836645958
4096000	3.42755637900	2.12364402912
8192000	8.74831699000	2.55234809370

Podemos observar que el ratio se aproximaría en torno a 3 seguramente si continuáramos obteniendo más mediciones para una n mayor. El modelo aproximado del orden de crecimiento del tiempo de ejecución es una ley de potencia, con exponente igual al logaritmo de base 2 del ratio. En este ejemplo el ratio será 3 por lo que el exponente es lg 3 = 1,6 , que no daría un orden de crecimiento cuadrático pero tampoco lineal, por lo que se quedaría en medio de ambas, aproximándose más al crecimiento cuadrático, tal como deducimos anteriormente con DoublingTest.

Análisis de la memoria

A continuación analizaremos el uso de la memoria del método multiply cuando es llamado con uno de los factores siendo el número 42.

El coste de memoria de un objeto Int será: 16 bytes por ser un objeto + 8 bytes de la variable de instancia String(aparte estaría el coste de memoria del propio objeto String, 56 + 2n bytes siendo n la longitud del String) + 4 bytes por la variable de instancia de tipo primitivo int + (24 + 4n + padding(4bytes)) un array de n int.

Por tanto el coste de memoria del número 42 será de 64 bytes, y el del factor por el que será multiplicado de 56 + 4n.

Sabiendo esto, continuamos calculando el coste de memoria de la llamada a multiply. Se crean múltiples variables de tipo primitivo int en bucles for por lo que no se tendrán en cuenta a la hora del consumo total de memoria pues son relativamente insignificantes.

Se realiza una llamada a un métodos externos de multiply que tiene el siguiente coste de memoria:
(7 * (24 + 4n + padding(4bytes)) + 24 + 8 + padding(4bytes)(Int 42))(arrays)+ 9 * 4 (int) = 268 + 28n bytes.

Por lo tanto el consumo total de memoria de una llamada al método multiply con el factor 42 fijo es 268 + 28n + 64 * 2 + 56 + 4n = 452 + 32n bytes, siendo n cuan grande sea el factor por el que se vaya a multiplicar a 42.

Conclusiones

La llamada a multiply no termina de ser lo suficientemente óptima además de que consume bastante memoria debido a la creación de múltiples variables en el momento de su ejecución a pesar de que la mayor parte de ellas son variables locales y desaparecerán al terminar la ejecución de multiply. Esto puede provocar que en el ordenador o dispositivo en el que se ejecute el programa necesite mucha memoria para poder calcular números muy muy grandes sin problemas, hablando por experiencia debido a las pruebas realizadas desde mi ordenador.