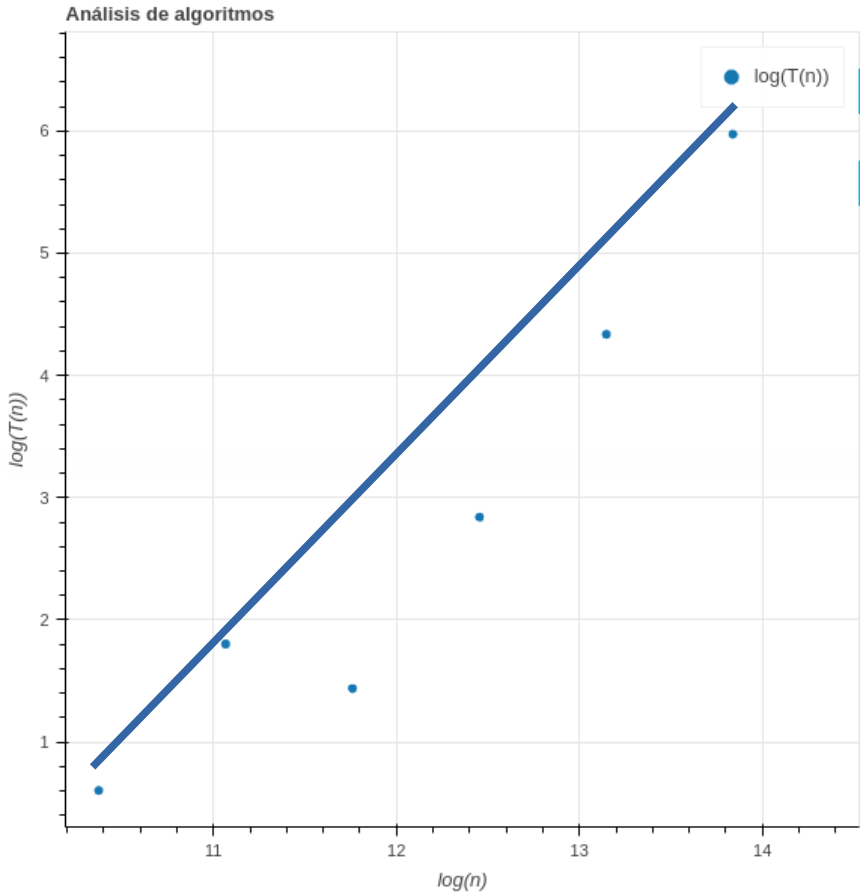
ANÁLISIS DE INTERSECCIÓN

Intersección

Para obtener unas mediciones precisas y algo más sofisticadas acerca del código Intersección, utilizaremos el código de **DoublingTest** creando dos arrays de n int que comenzará con n = 32000 (tamaño del array inicial), cuyos valores son números aleatorios entre -1000000000 y 1000000000. Los arrays doblarán su tamaño en cada iteración del bucle del programa principal. Ejecutando el código en nuestro ordenador obtenemos las siguientes medidas:

n	T(n)
32000	1.82877009300
64000	6.06628864600
128000	4.20516746500
256000	17.12657255700
512000	76.42688888400
1024000	393.11599315300

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 1.73 ~ 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) = 1,83. Obtenemos $a = 5{,}6x10^8$. Dicha constante solamente es válida 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 $T(64000) = 64000^2 / 5{,}6x10^8 = 7{,}3 \sim 6.1$ valor bastante parecido al que obtuvimos en los resultados experimentales.

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

Otra forma para predecir el orden de crecimiento del código Intersección sería usando el programa DoublingRatio, del cual obtendremos 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	0.00117335500	4.10004507637
500	0.00062260500	0.53061946299
1000	0.00243241400	3.90683338553
2000	0.01034948100	4.25481887541
4000	0.06359224500	6.14448637569
8000	0.01970896300	0.30992714599
16000	0.06878551100	3.49006241475
32000	0.25880091400	3.76243354505
64000	1.05004705600	4.05735451151
128000	4.19899608200	3.99886467755
256000	16.86037942700	4.01533583212
512000	69.29252726100	4.10978457282
1024000	317.18205982300	4.57743529296

Podemos observar que el ratio se aproxima en torno a 4 a pesar de alguna medida irregular. 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 4 por lo que el exponente es lg 4 = 2 , que nos daría un orden de crecimiento cuadrático. Este orden de crecimiento es igual al que hemos obtenido con DoublingTest, por lo que asumiremos que el orden de crecimiento del código de ParejasIguales tiene un orden de crecimiento cuadrático.

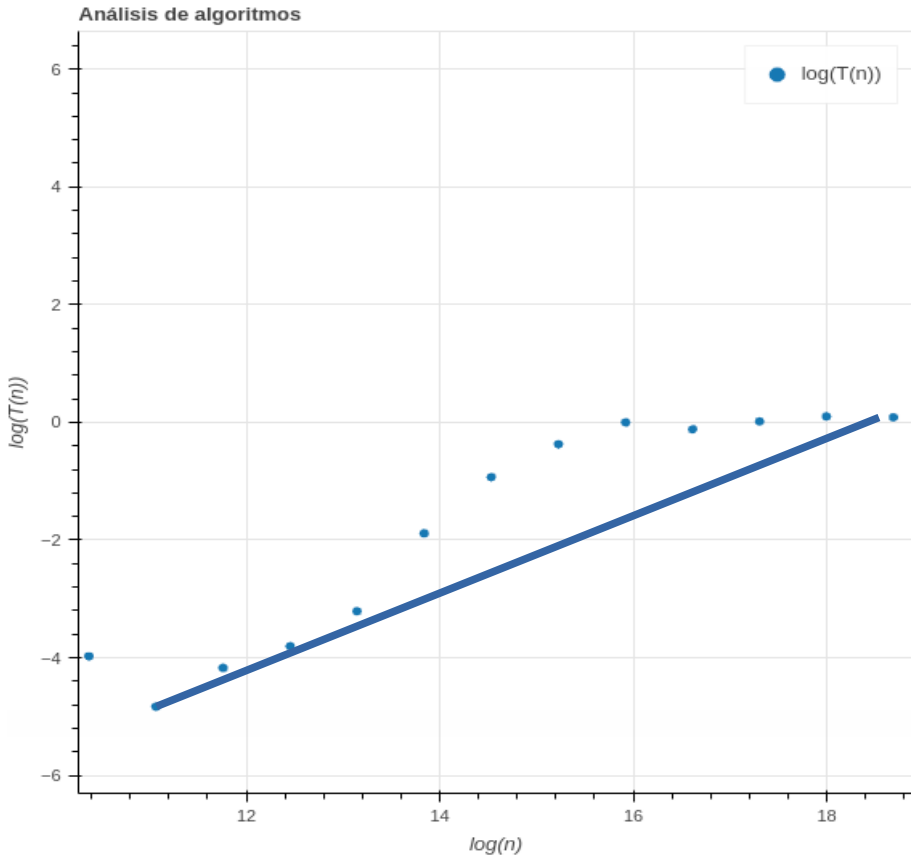
Interseccion Fast

Para obtener un tiempo de ejecución menor al que hemos obtenido con el código Intersección, utilizaremos la función Arrays.sort() para ordenar los arrays y de esta manera poder reducir el tiempo de ejecución considerablemente. Para obtener las medidas utilizaremos al igual que con Intersección los códigos DoublingTest y DoublingRatio en las mismas condiciones.

DoublingTest:

n	T(n)
32000	0.01870112200
64000	0.00796193600
128000	0.01541216300
256000	0.02227231200
512000	0.04034379300
1024000	0.15210327500
2048000	0.39402254000
4096000	0.68764517200
8192000	0.99868235600
16384000	0.88836466600
32768000	1.01792147900
65536000	1.10663298800
131072000	1.08911060400

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



Colocando una recta sobre los puntos, sin tener en cuenta alguna irregularidad en algunas de las medidas, observamos que la gran parte de los puntos están próximos a la recta de pendiente 0,95~1. De la gráfica anterior tenemos la siguiente función:

$lg(T(n)) = 1lg(n) - lg(a)$

Si despejamos T(n), obtenemos:

$T(n) = n / a$

Al igual que con Intersección podemos obtener a sustituyendo n = 512000 y T(n) = 0.04. Obtenemos $a = 12{,}7x10^7$. Con la constante anterior, podemos observar que para n = 1024000, obtenemos $T(1024000) = 1024000 / 12{,}7x10^7 = 0{,}081 \sim 0{,}15$ valor bastante parecido al que obtuvimos en los resultados experimentales.

De este análisis, del código IntersecciónFast, deducimos que tiene un orden de crecimiento muy cercano al lineal, n.

A continuación utilizaremos DoublingRatio para calcular el orden de crecimiento de otra forma distinta que con DoublingTest. Las medidas que hemos obtenido son:

n	**T(n)**	**ratio**
250	0.00046157400	0.60170980728
500	0.00029167300	0.63190950963
1000	0.00059674000	2.04592128857
2000	0.00115414700	1.93408687200
4000	0.00233050600	2.01924538209
8000	0.00467130300	2.00441577923
16000	0.00301356100	0.64512214258
32000	0.00429211400	1.42426650730
64000	0.00654443200	1.52475726414
128000	0.03206414600	4.89945437587
256000	0.01253941400	0.39107275772
512000	0.06607224700	5.26916544904
1024000	0.12010354700	1.81776089740
2048000	0.36326941900	3.02463522580
4096000	0.56285694700	1.54942012061
8192000	1.00277631500	1.78158290547
16384000	0.81185276200	0.80960504337
32768000	0.67393995400	0.83012583752
65536000	1.11324898100	1.65185188145
131072000	1.38208680900	1.24148939958

Podemos observar que el ratio se aproxima en torno a 2 a pesar de alguna medida irregular. Como dijimos anteriormente en Intersección si realizamos el siguiente logaritmo, lg 2 = 1 , obtenemos un orden de crecimiento lineal. Este orden de crecimiento es igual al que hemos obtenido con DoublingTest.

Por lo tanto, el tiempo de ejecución total de IntersecciónFast será la suma del tiempo de ejecución de las tres llamadas de la función de Arrays.sort(), que tiene un orden de crecimiento O(nlog(n)), más el resto del código, que tiene un orden de crecimiento O(n) tal como hemos deducido anteriormente con DoublingTest y DoublingRatio. Entonces tenemos lo siguiente:

$T(n) \sim 3n + 3n\log(n)$

$T(n) \sim n\log(n)$

De esta manera podemos decir que IntersecciónFast tiene un orden de crecimiento linearítmico.

Conclusiones

Podemos observar que habiendo ordenado los arrays previamente, resulta mucho más rápido el tener que encontrar los números que tienen en común ambos arrays ya que podemos desarrollar un algoritmo mucho más eficiente respecto al código Intersección en el que no hemos ordenado los arrays.