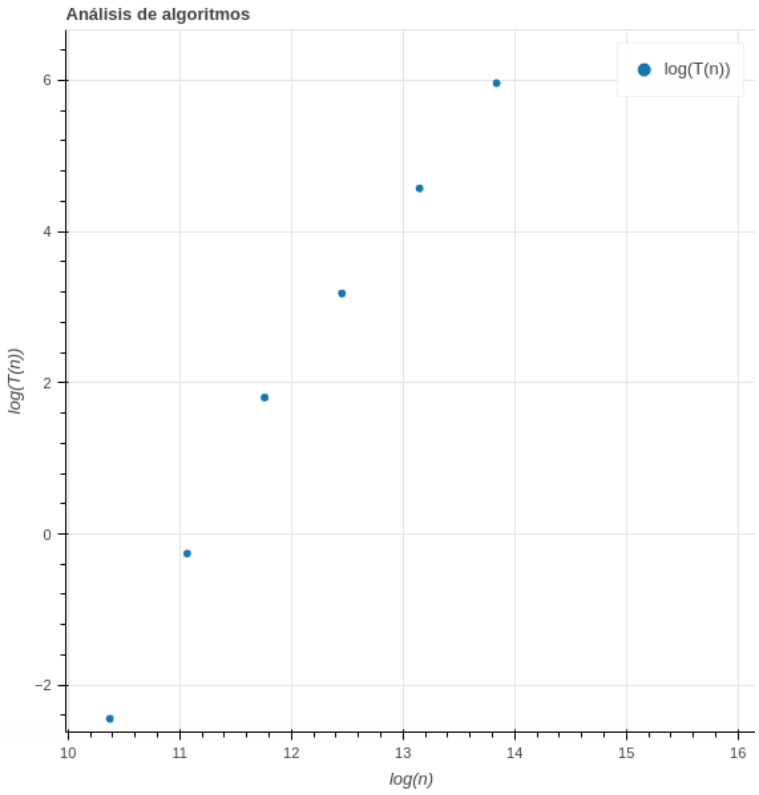
ANÁLISIS DE PAREJAS IGUALES

Parejas Iguales

Para obtener unas mediciones precisas y algo más sofisticadas acerca del código ParejasIguales, utilizaremos el código de **DoublingTest** creando un array 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	0.08683342700
64000	0.77020339500
128000	6.07169959400
256000	24.00670722400
512000	96.45203216900
1024000	388.10324895800

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 3 aproximadamente. De la gráfica anterior obtenemos la siguiente función:

$lg(T(n)) = 3lg(n) - lg(a)$

Si despejamos T(n), obtenemos:

$T(n) = n^3/a$

Para obtener el valor de la constante "a", podemos sustituir para valores experimentales n = 32000 y T(n) = 0.087. Obtenemos $a = 3.77x10^{14}$. Dicha constante solo 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 $T(128000) = 128000^3 / 3.77x10^{14} = 5,6 \sim 6.07$ valor bastante parecido al que obtuvimos en los resultados experimentales.

Podremos decir que el algoritmo tiene un orden de crecimiento cúbico, n^3.

Otra forma para predecir el orden de crecimiento del código ParejasIguales 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	0.00134225800	3.80202131215
500	0.00139878500	1.04211336420
1000	0.00110600100	0.79068691757
2000	0.00486659900	4.40017594921
4000	0.00390584300	0.80258163864
8000	0.00470427400	1.20441963489
16000	0.01936082900	4.11558276580
32000	0.07434915000	3.84018421938
64000	1.24084368800	16.68941323472
128000	6.00822099200	4.84204501349
256000	5.53037512900	0.92046799483
512000	23.39469646000	4.23021873097
1024000	98.41581117100	4.20675734516
2048000	527.94591911600	5.36444208338
4096000	2288.69638271900	4.33509626621

Podemos observar que el ratio se aproxima en torno a 4.5 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 4.5 por lo que el exponente es lg 4.5 = 2,17 , que nos daría un orden de crecimiento bastante cercano al cuadrático. Este orden de crecimiento es distinto al que hemos obtenido con DoublingTest, por lo que asumiremos que el orden de crecimiento del código de ParejasIguales tiene un orden de crecimiento que no es cuadrático pero tampoco cúbico.

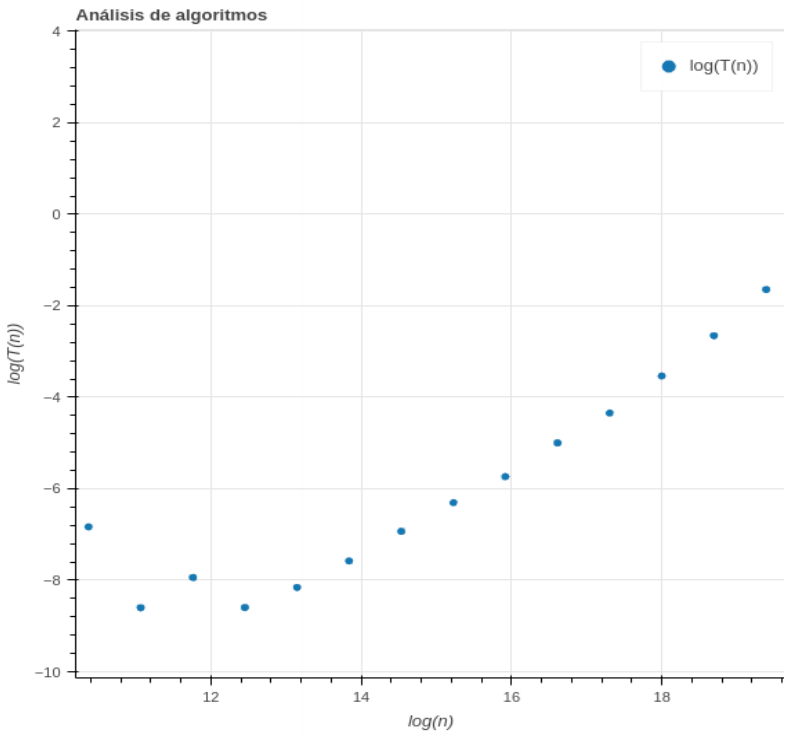
Parejas Iguales Fast

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

DoublingTest:

n	T(n)
32000	0.00107551900
64000	0.00018381000
128000	0.00035662900
256000	0.00018518200
512000	0.00028599400
1024000	0.00051044200
2048000	0.00098235100
4096000	0.00182816100
8192000	0.00323227500
16384000	0.00676516400
32768000	0.01298501200
65536000	0.02927704200
131072000	0.07019561300
262144000	0.19347726000

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 0.63 . De la gráfica anterior tenemos la siguiente función:

lg(T(n)) = 0.63lg(n) - lg(a)

Si despejamos T(n), obtenemos:

T(n) =$n^{0.63}$/a

Al igual que con ParejasIguales podemos obtener a sustituyendo n = 512000 y T(n) = 0.00028. Obtenemos a = $13.82x10^6$. Con la constante anterior, podemos observar que para n = 1024000, obtenemos T(1024000) =$1024000^{0.63}$/ $13.82x10^6$ = 0,00044 ~ 0.00051 valor bastante parecido al que obtuvimos en los resultados experimentales.

De este análisis, del bucle for del código ParejasIgualesFast, deducimos que tiene un orden de crecimiento aproximadamente 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	0.00001926600	1.37742189176
500	0.00001829500	0.94960033219
1000	0.00003810200	2.08264553157
2000	0.00007670400	2.01312267073
4000	0.00018722300	2.44085054234
8000	0.00037085800	1.98083568792
16000	0.00075998100	2.04925065659
32000	0.00021157800	0.27839906524
64000	0.00028757100	1.35917250376
128000	0.00035256400	1.22600679484
256000	0.00020025700	0.56800183796
512000	0.00036636900	1.82949410008
1024000	0.00058979100	1.60982779657
2048000	0.00106169000	1.80011224315
4096000	0.00170097300	1.60213715868
8192000	0.00319269200	1.87697982272
16384000	0.00639211800	2.00210919187
32768000	0.01325302000	2.07333782011
65536000	0.02921762300	2.20460113997
131072000	0.07005523000	2.39770463189

Podemos observar que el ratio se aproxima en torno a 2 a pesar de alguna medida irregular. Como dijimos anteriormente en ParejasIguales 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 que asumiremos que el orden de crecimiento del for del código de ParejasIguales tiene un orden de crecimiento lineal.

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

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

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

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

Conclusiones

Podemos observar que habiendo ordenado el array previamente, obtenemos un algoritmo mucho más rápido respecto al de ParejasIguales al tener que encontrar cuántas parejas de números iguales hay en el array.