Análisis de Intersección

En este análisis voy a comparar dos algoritmos que escriben por pantalla la intersección de dos arrays que se les pasan como argumentos. El tiempo de ejecución de estos algoritmos depende principalmente del tamaño de los arrays, cuyas longitudes serán n y m (en los experimentos, n = m).

El primer algoritmo, Interseccion, tiene 3 bucles, 2 de ellos anidados bajo el primero. El primero recorre el primer array, el segundo comprueba que el término que se ha encontrado no sea parte de la intersección ya encontrada, en cuyo caso el tercer bucle buscará el término en el segundo array, recorriéndolo hasta encontrarlo o terminar el array. Por tanto, el primer bucle tendrá n iteraciones, el segundo tendrá 1 iteración la primera vez y acabará con tantas iteraciones como números distintos haya en la intersección, y el tercero tendrá tantas iteraciones como cueste encontrar el término si existe en el otro array, o m iteraciones si el término no existe.

Esto nos deja con un caso favorable muy optimista, en el caso en que tengamos un primer array con todos los términos iguales y un segundo array donde el primer término sea igual que los del primer array. En este caso, el primer bucle tendrá n iteraciones, el segundo 1, y el tercero sólo se pasará en la primera iteración del primer bucle, y sólo será necesario comprobar el primer término. El mejor caso posible tiene n iteraciones. Sin embargo, el pero caso posible, en el que el primer array tiene todas sus cifras distintas y todas presentes en el segundo array, tendrá n iteraciones del primer bucle, que resultan en n*(n+1)/2 iteraciones del segundo bucle y n*(m/2) iteraciones del tercer bucle, siendo pues el crecimiento para n = m, $(n^2 + n + n^2)/2 \sim n^2$.

Considerando estos casos, el crecimiento está acotado entre n y n^2, dependiendo principalmente del número de cifras distintas presentes en el primer array. Para arrays con un margen de valores amplio, los resultados de DoublingRatio:

n:	t:	ratio:
16000	0.1	3.7
32000	0.5	4.3
64000	1.5	2.7
128000	5.7	3.9
256000	22.3	3.9
512000	89.6	4.0
1024000	434.0	4.8

Pese a presentarse algo inestables, tienen una clara tendencia al crecimiento cuadrático, pues nos encontramos más a meudo en escenarios poco favorables para el algoritmo. Sin embargo, reduciendo drásticamente el número de valores posibles en el array, nos encontramos con que los resultados de DoublingRatio:

n:	t:	ratio:
2048000	0.0	3.1
4096000	0.1	3.0
8192000	0.1	1.5
16384000	0.1	1.2
32768000	0.3	2.1
65536000	0.6	1.9

Se mantienen inestables, pero en este caso se aproximan al crecimiento lineal previsto para los casos más favorables.

El segundo algoritmo, InterseccionFast, tiene como objetivo optimizar el primero, primero llamando a Arrays.sort y después implementando un método que aproveche tener los arrays ordenados. Este algoritmo tiene dos bucles, uno que recorre el primer array, y otro que cada vez que encuentra un número distinto en el primer array realiza una búsqueda binaria de ese número en el segundo array. Aquí, la diferencia entre el mejor caso y el peor es negligible, pues tan solo se reduce el número de búesquedas binarias a realizar. En un caso ideal, donde el primer array tiene muy pocos números distintos, el número de iteraciones se aproxima a n, mientras que en el peor caso posible, donde tienes que hacer muchas búsquedas binarias, el bucle más interno se ejecuta nlog(m) veces. Sin embargo, puesto que ordenar los arrays requiere nlog(n) y mlog(m) iteraciones respectivamente, la diferencia entre los casos favorables de n iteraciones y los poco favorables con nlog(m) iteracions no cambia el orden de magnitud. Considerando que n = m, el crecimiento varía entre 2nlog(n) + n y 3nlog(n), linearítmico en ambos casos. Los resultados de DoublingRatio:

n:	t:	ratio:
4096000	1.8	2.3
8192000	4.2	2.3
16384000	9.3	2.2
32768000	20.6	2.2
65536000	47.3	2.3

Se mantienen un poco por encima del crecimiento lineal, manteniéndose dentro del margen de lo que se podría esperar del orden de crecimiento de nuestro algoritmo.

Como conclusión, InterseccionFast tiene un margen mucho más estable que Interseccion, y en la mayoría de escenarios, presenta una optimización frente al crecimiento cuadrático de Interseccion. Sin embargo, es posible que para arrays cuyos valores se encuentren en un margen muy reducido, la optimización sea negligible o inexistente.