

## Contenido

Definición de colisión en inserción y búsqueda. Implementación de contadores de operaciones necesarias para insertar o buscar un elemento.....	2
Encadenamiento .....	2
Recolocación .....	2
Influencia del tamaño (Tam) elegido para la tabla .....	2
Encadenamiento .....	2
Recolocación Lineal.....	4
Conclusiones .....	6
Influencia de la elección de la función hash utilizada para obtener la posición del jugador en la tabla .....	7
Encadenamiento .....	7
Recolocación Lineal.....	11
Conclusiones .....	16
Influencia de la elección de la clave utilizada para obtener la posición del jugador en la tabla .....	16
Encadenamiento .....	16
Recolocación Lineal.....	18
Conclusiones .....	19
Influencia de la elección de la estrategia de recolocación .....	20
Lineal A = 1 .....	20
Lineal A = 3 .....	20
Lineal A = 7 .....	21
Cuadrática .....	22
Conclusiones .....	22
Estimación de la eficiencia en el acceso a los datos (búsqueda).....	23
Encadenamiento .....	23
Recolocación Lineal.....	24
Recolocación Cuadrática.....	25

Definición de colisión en inserción y búsqueda. Implementación de contadores de operaciones necesarias para insertar o buscar un elemento.

#### Encadenamiento

```
int ColisionesProducidas(TablaHash t, tipoelem elemento);
```

Devuelve el número de valores almacenados con ese mismo hash.

```
int PasosAdicionalesInsercion(TablaHash t, tipoelem elemento);
```

Devuelve -1 (no se producen colisiones al ser una lista).

```
int PasosAdicionalesBusqueda(TablaHash t, tipoelem elemento);
```

Devuelve el número de posiciones que hay que recorrer en la lista hasta encontrar el elemento.

#### Recolocación

```
int ColisionesProducidas(TablaHash t, tipoelem elemento);
```

Devuelve el número de valores almacenados con ese mismo hash. Cambiará automáticamente en función de si es cuadrática o lineal.

```
int PasosAdicionalesInsercion(TablaHash t, tipoelem elemento);
```

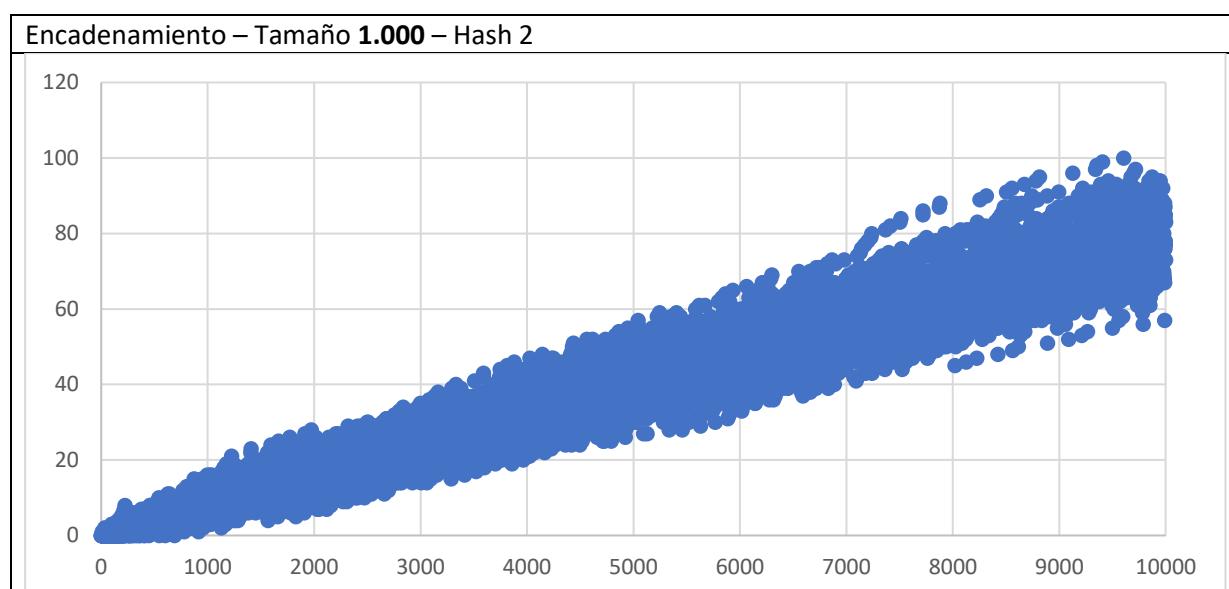
Devuelve el número de posiciones que hay que recorrer hasta encontrar una vacía o borrada, o -1 si está llena. Cambiará automáticamente en función de si es lineal o cuadrática.

```
int PasosAdicionalesBusqueda(TablaHash t, tipoelem elemento);
```

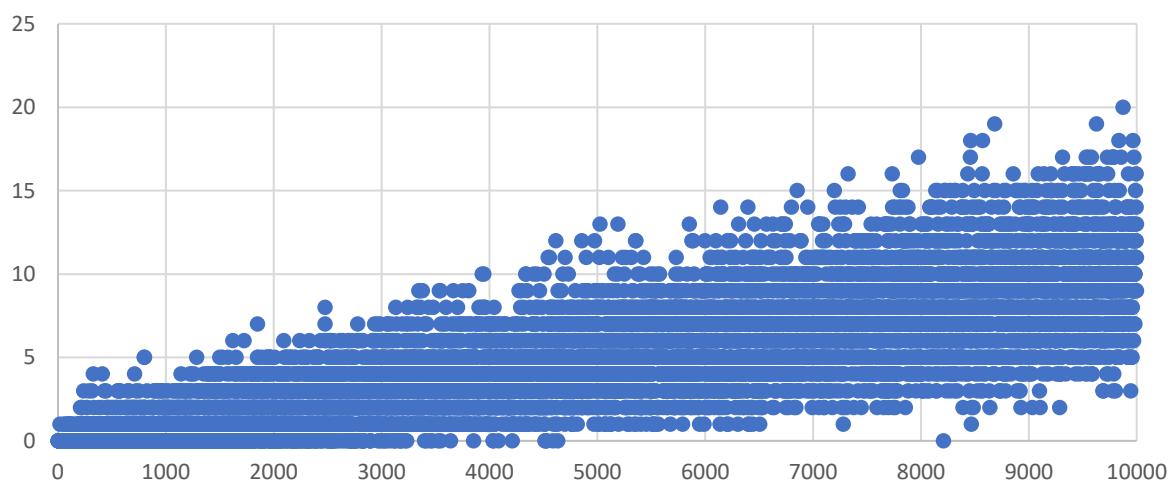
Devuelve el número de posiciones que hay que recorrer hasta encontrar el elemento, o -1 si no está. Cambiará automáticamente en función de si es cuadrática o lineal.

#### Influencia del tamaño (Tam) elegido para la tabla

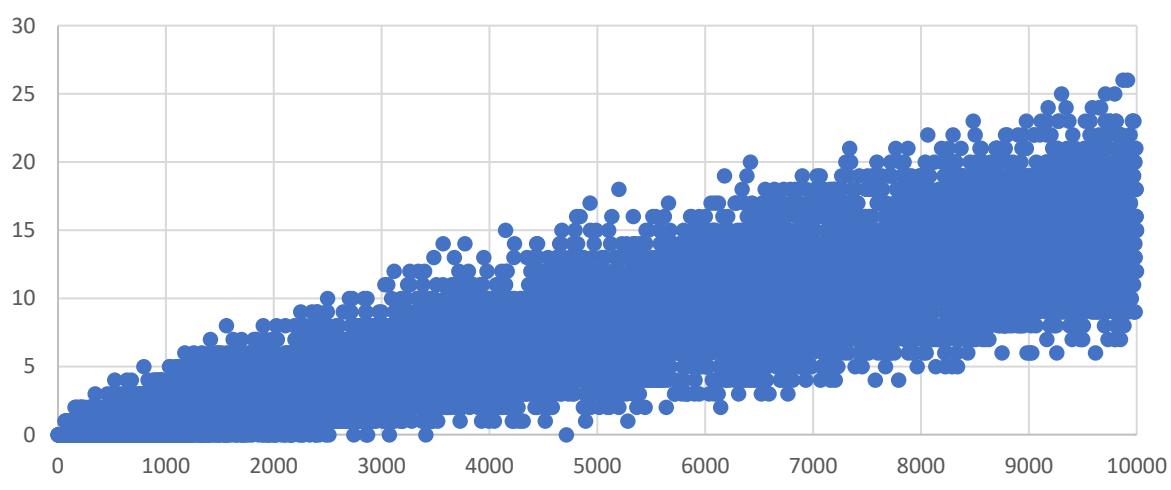
#### Encadenamiento



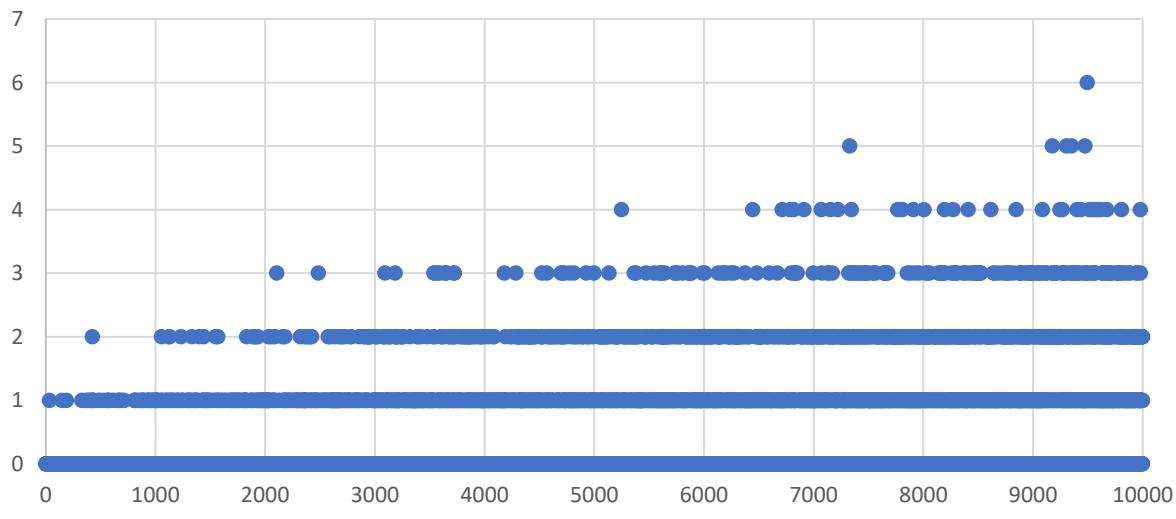
Encadenamiento – Tamaño **1.001** – Hash 2



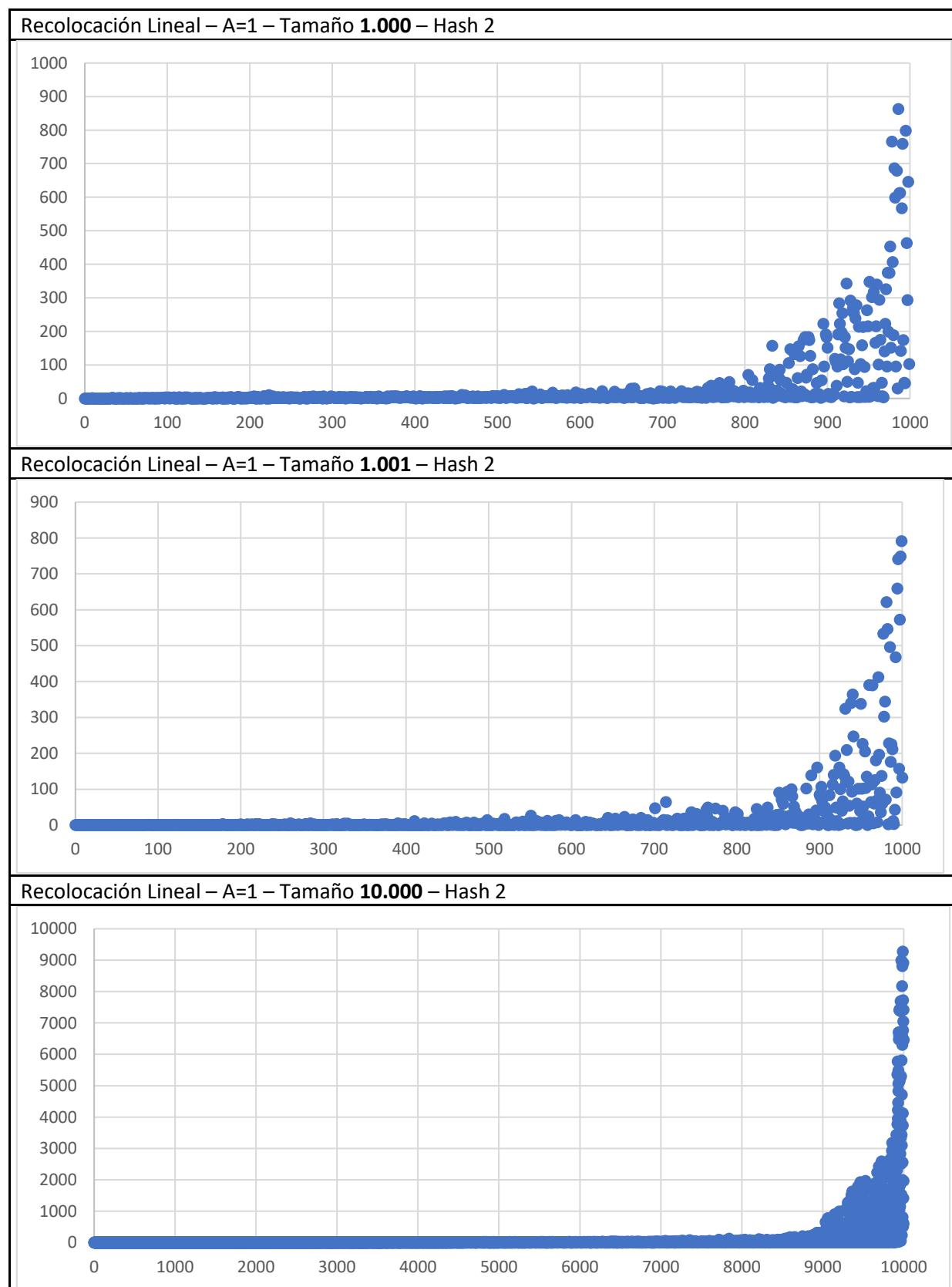
Encadenamiento – Tamaño **10.000** – Hash 2



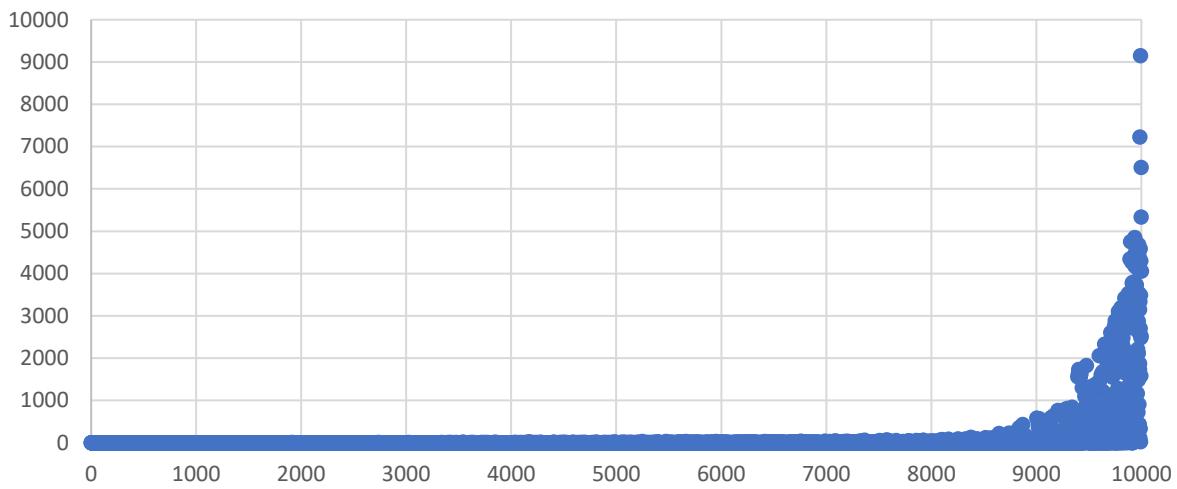
Encadenamiento – Tamaño **10.001** – Hash 2



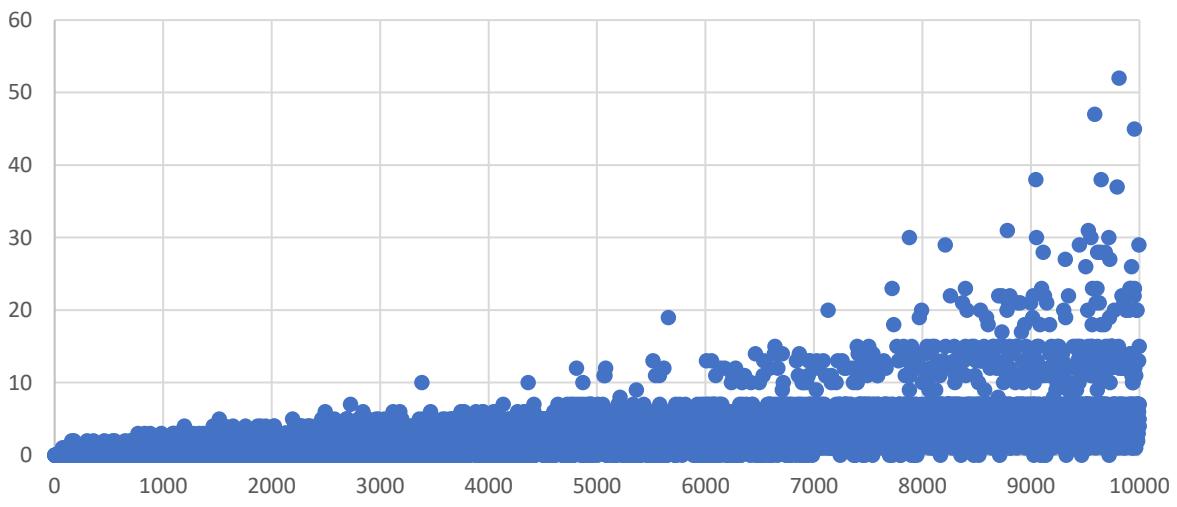
## Recolección Lineal



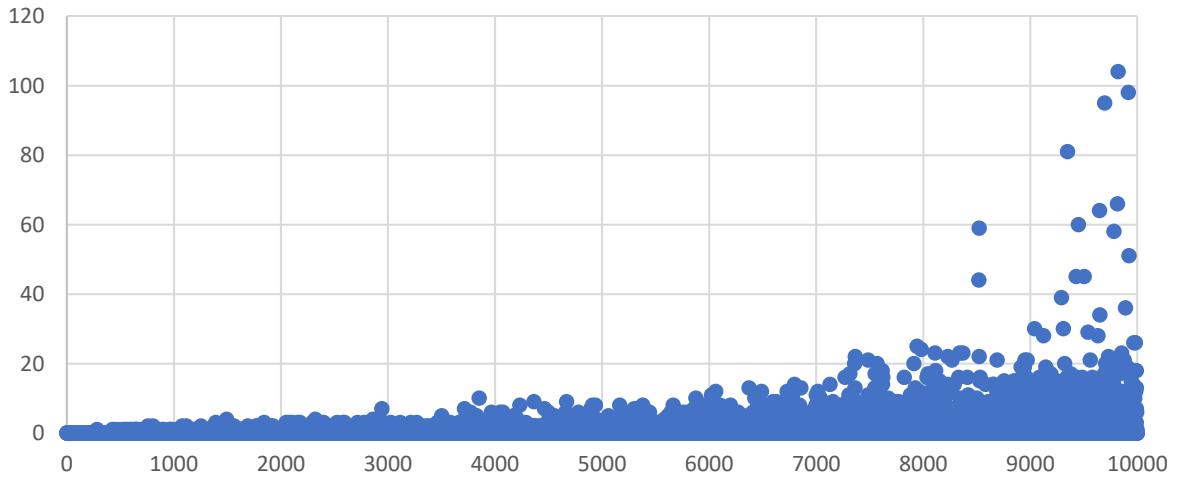
Recolección Lineal – A=1 – Tamaño **10.001** – Hash 2



Recolección Lineal – A=1 – Tamaño **15.000** – Hash 2



Recolección Lineal – A=1 – Tamaño **15.001** – Hash 2



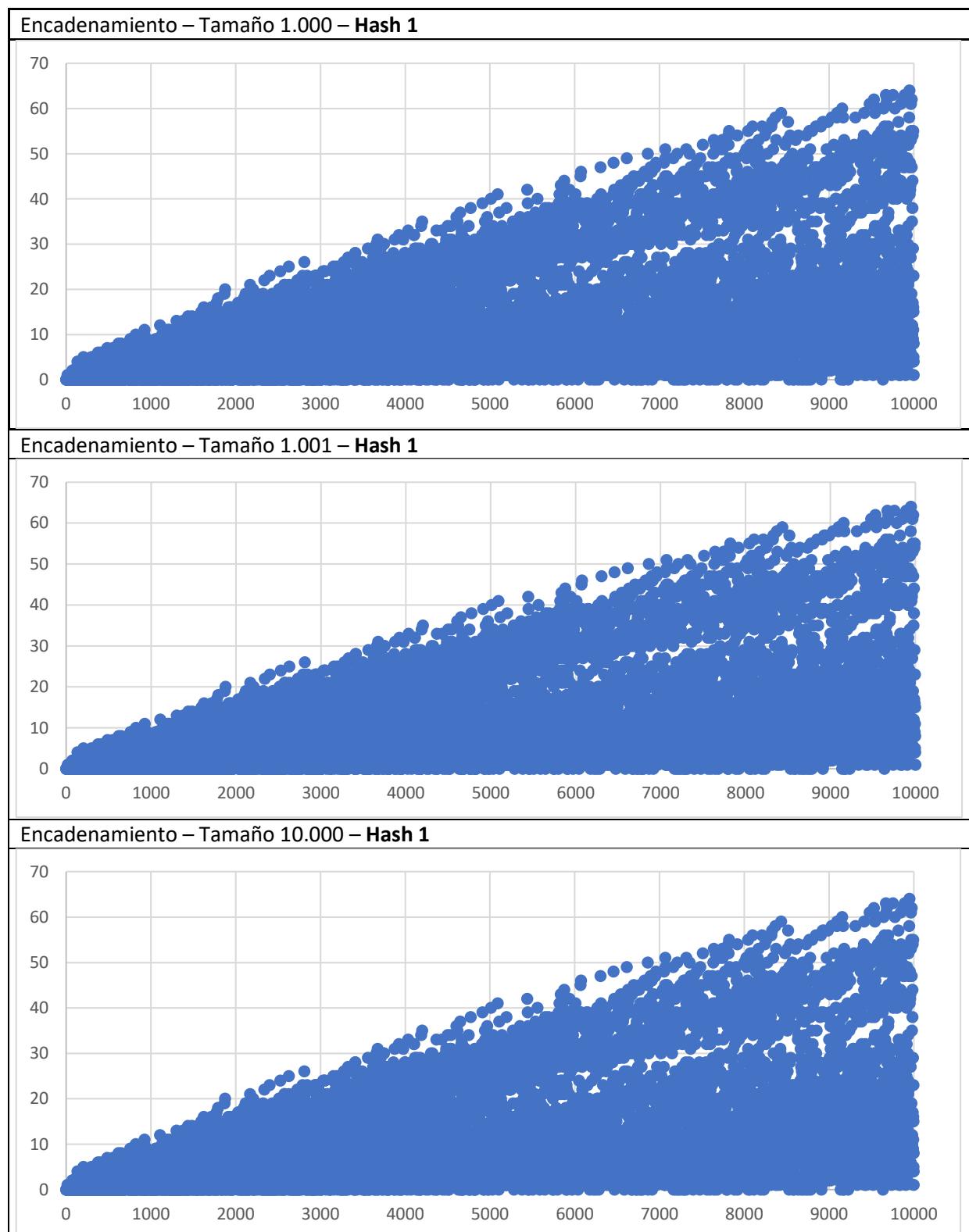
## Conclusiones

Como bien se puede observar en los gráficos, el método de encadenamiento con esta función hash se comporta especialmente bien en los casos en los que el tamaño de la tabla no tiene una gran cantidad de divisores (1.001, pero especialmente 10.001). En los otros dos casos, el de un tamaño de 10.000 no fue especialmente mala, pero en el de 1.001 sí que se comportó de una forma poco óptima comparado con los otros resultados.

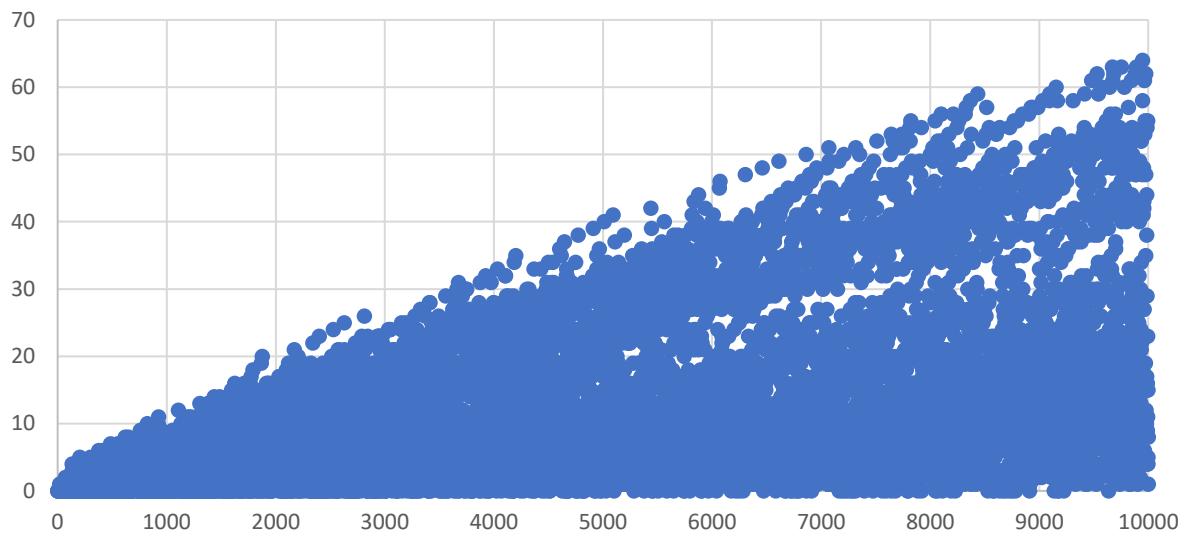
Sin embargo, en la recolocación lineal simple, se ha probado un par de valores más: 15.000/1, ya que se ha observado como cuando la tabla se va llenando, el número de colisiones incrementa considerablemente. En esta última prueba, se ha obtenido un muy buen resultado con el valor 15.000, mientras que con el 15.001 no se ha comportado tan bien (pero no ha sido mala del todo).

## Influencia de la elección de la función hash utilizada para obtener la posición del jugador en la tabla

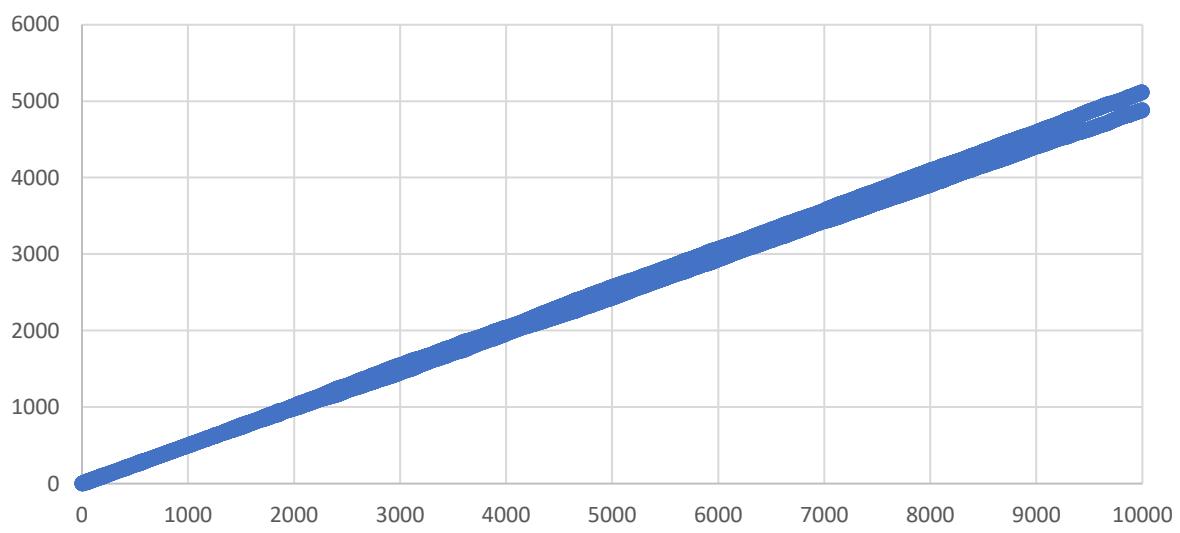
### Encadenamiento



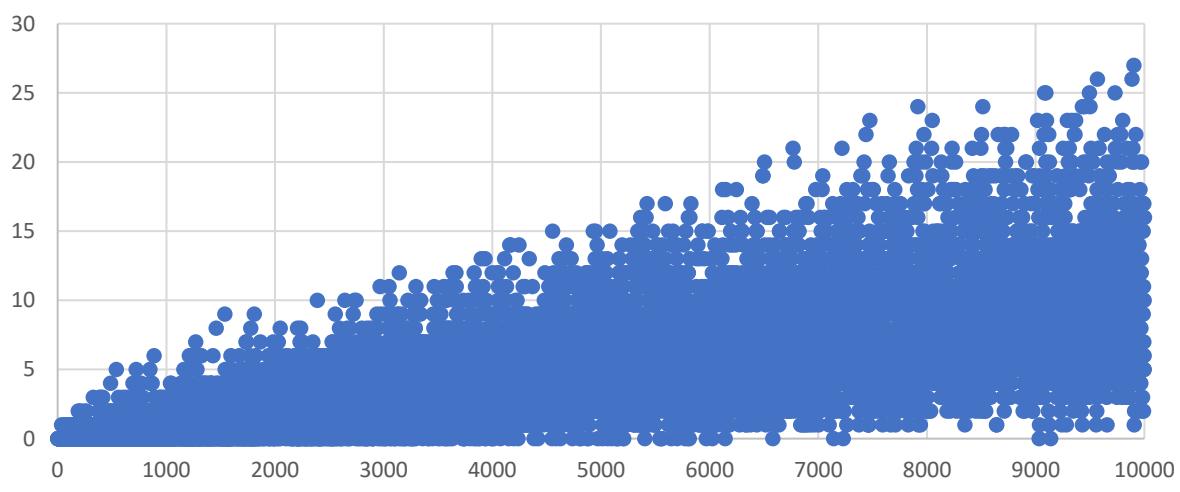
Encadenamiento – Tamaño 10.001 – Hash 1



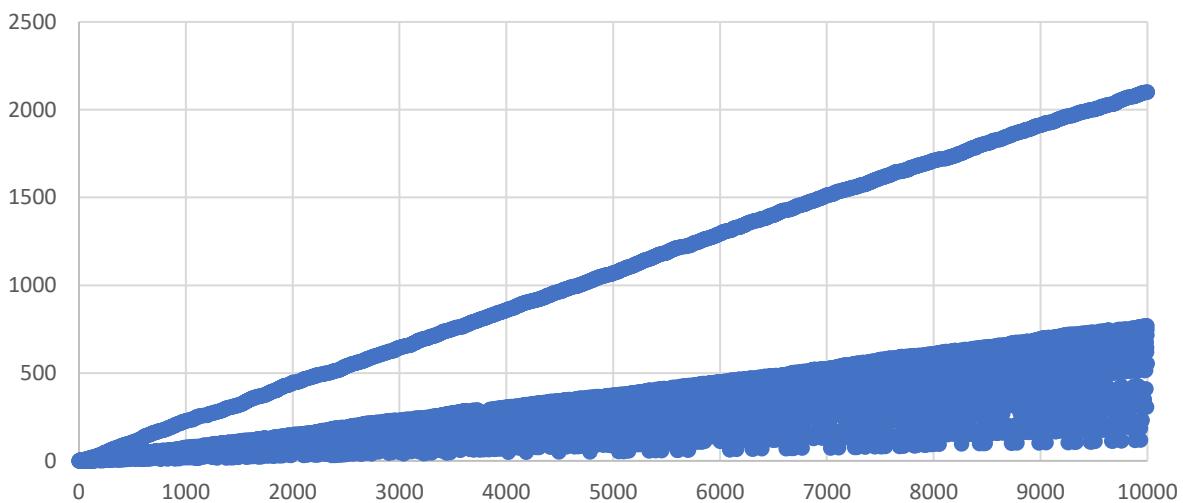
Encadenamiento – Tamaño 1.000 – Hash 3 – K=500



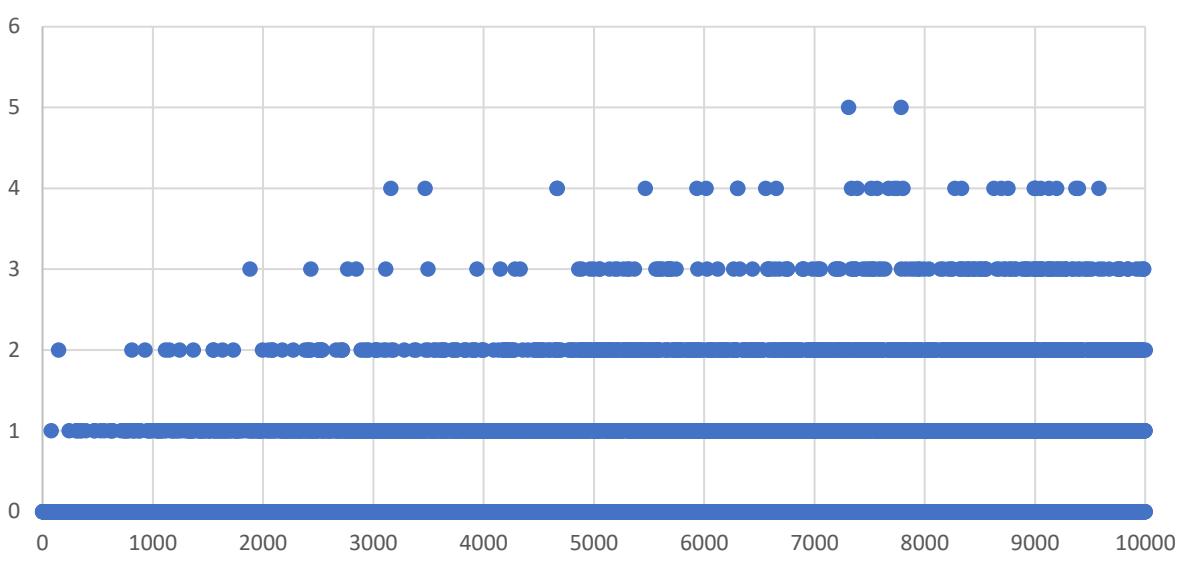
Encadenamiento – Tamaño 1.001 – Hash 3 – K=500



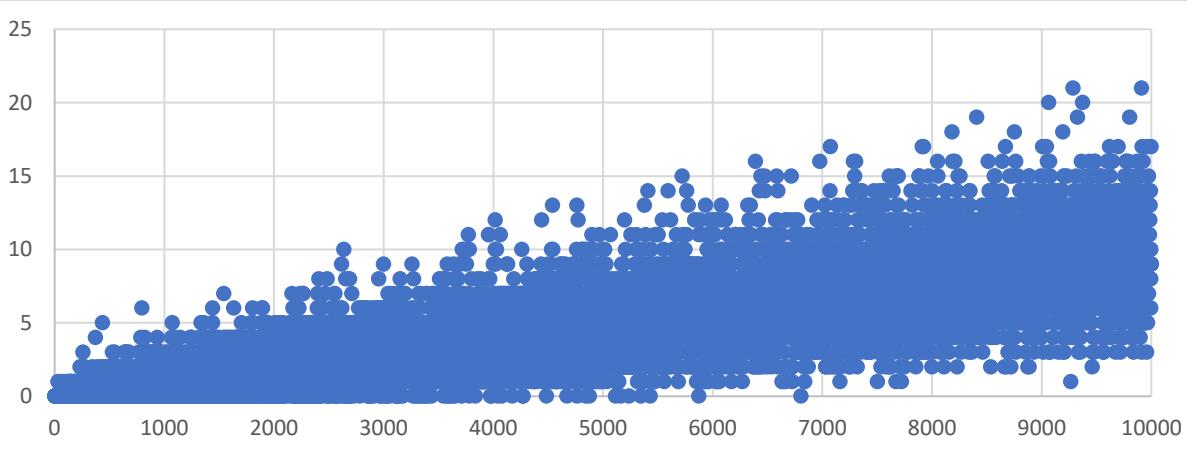
Encadenamiento – Tamaño 10.000 – Hash 3 – K=500



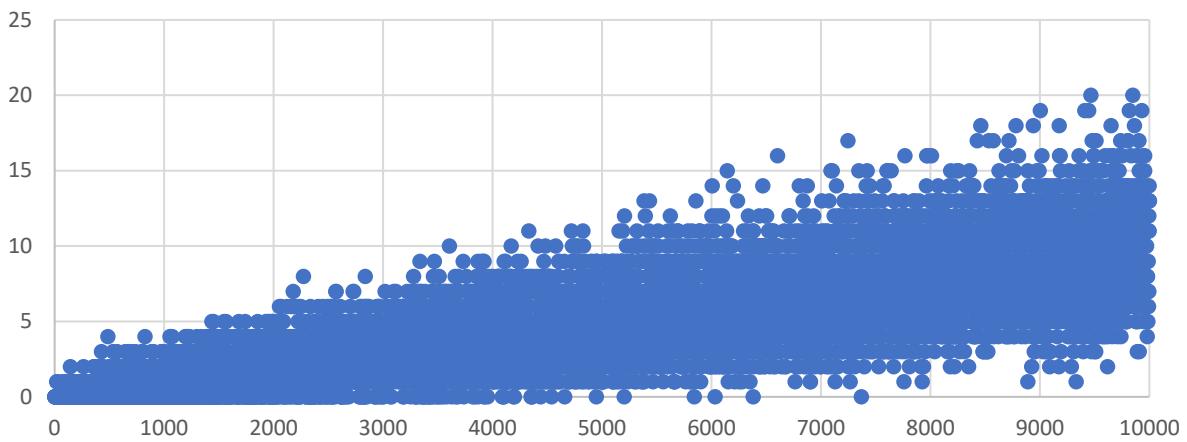
Encadenamiento – Tamaño 10.001 – Hash 3 – K=500



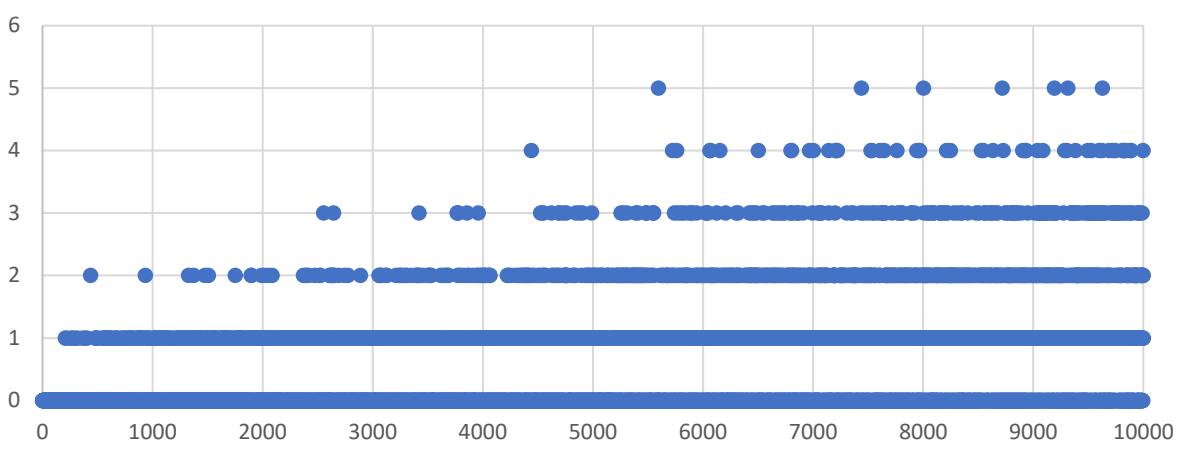
Encadenamiento – Tamaño 1.000 – Hash 3 – K=227



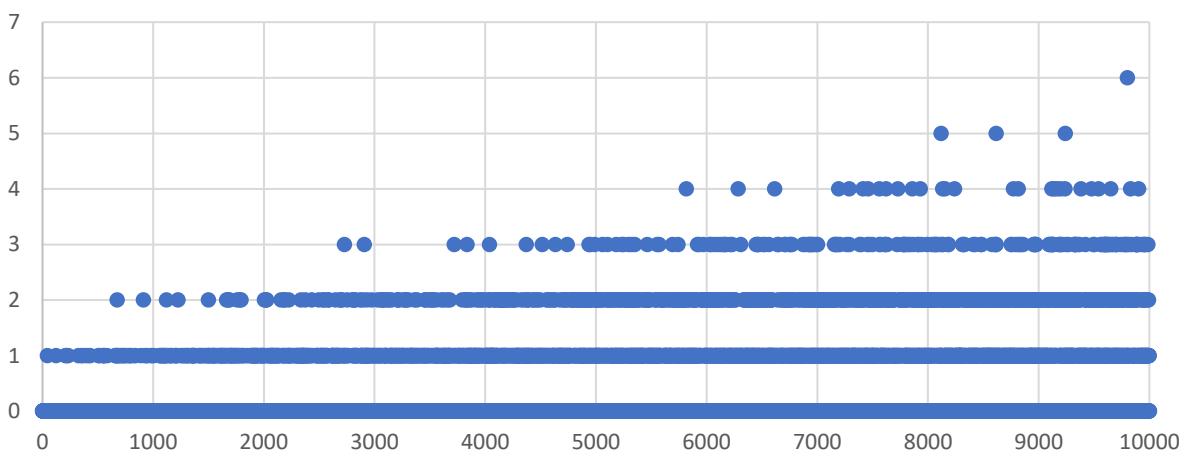
Encadenamiento – Tamaño 1.001 – Hash 3 – K=227



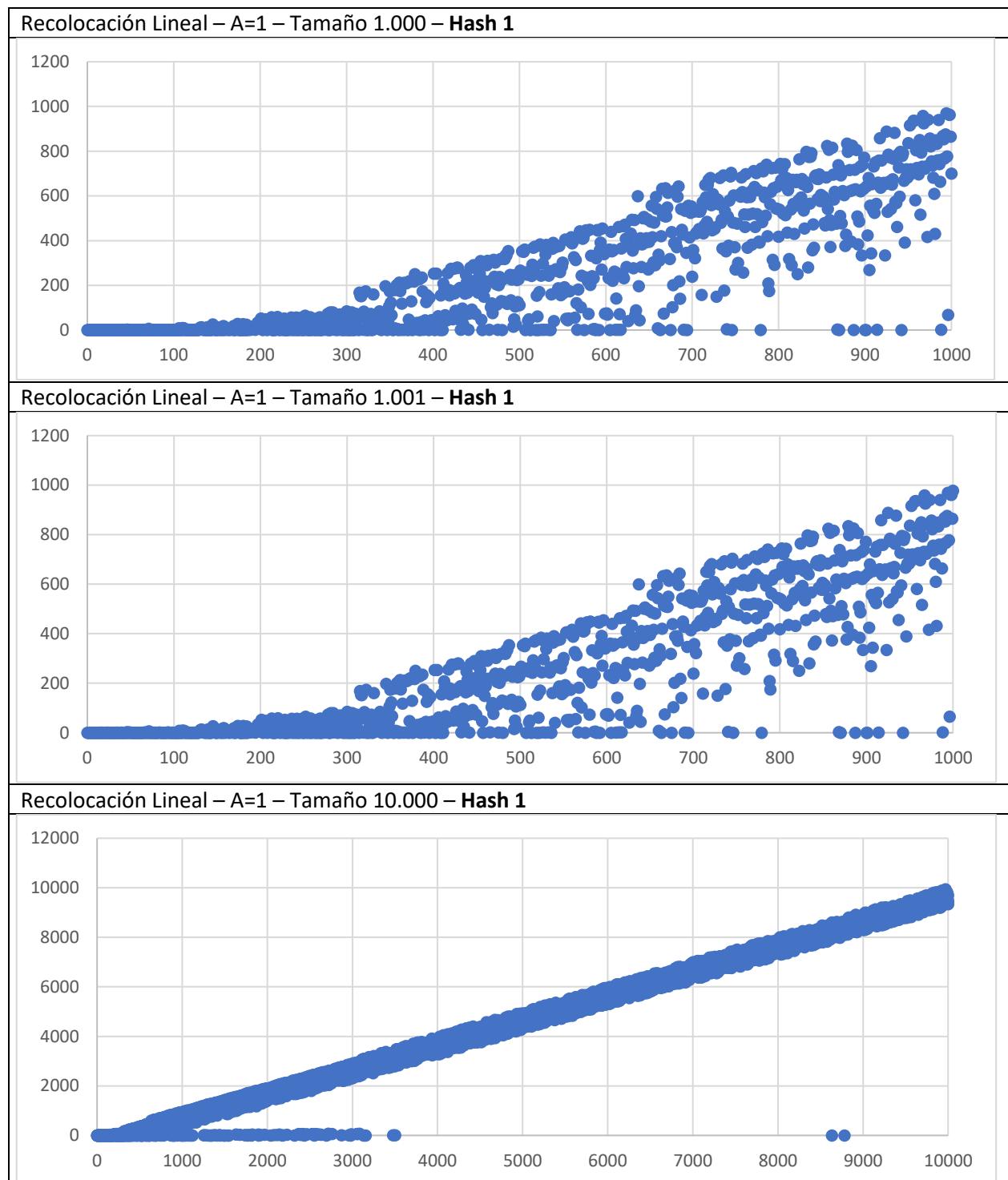
Encadenamiento – Tamaño 10.000 – Hash 3 – K=227



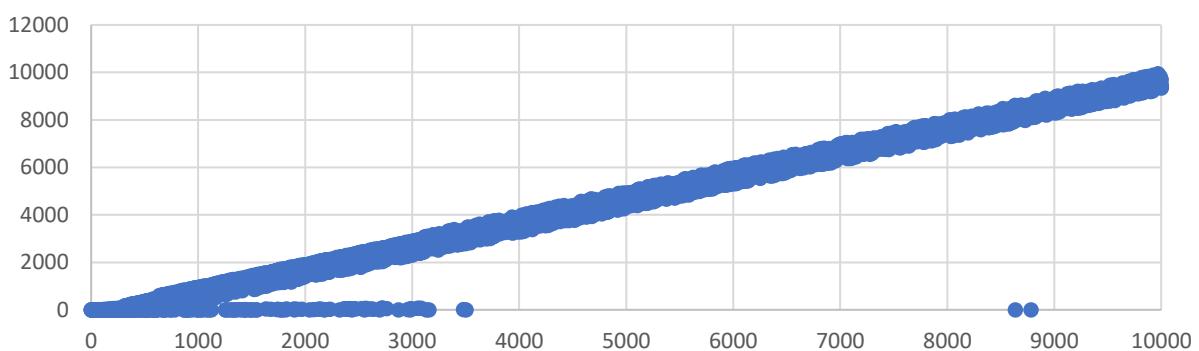
Encadenamiento – Tamaño 10.001 – Hash 3 – K=227



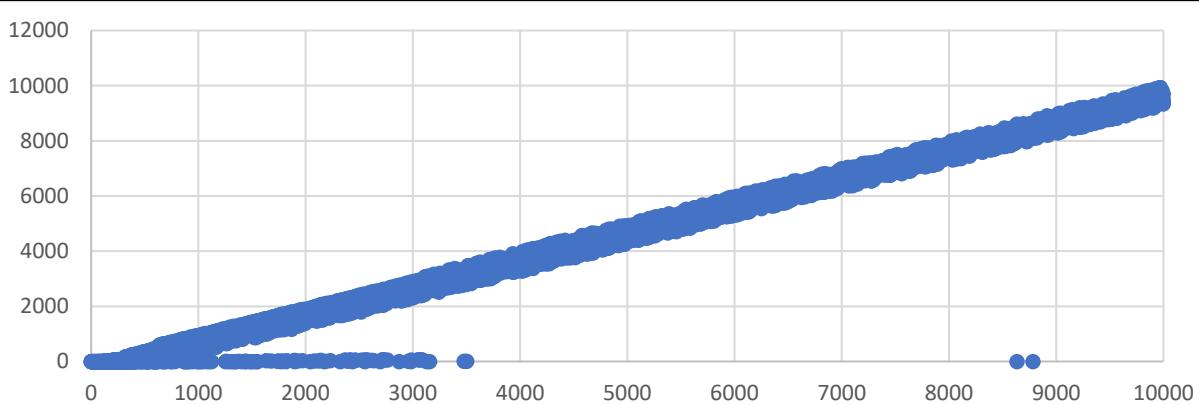
## Recolección Lineal



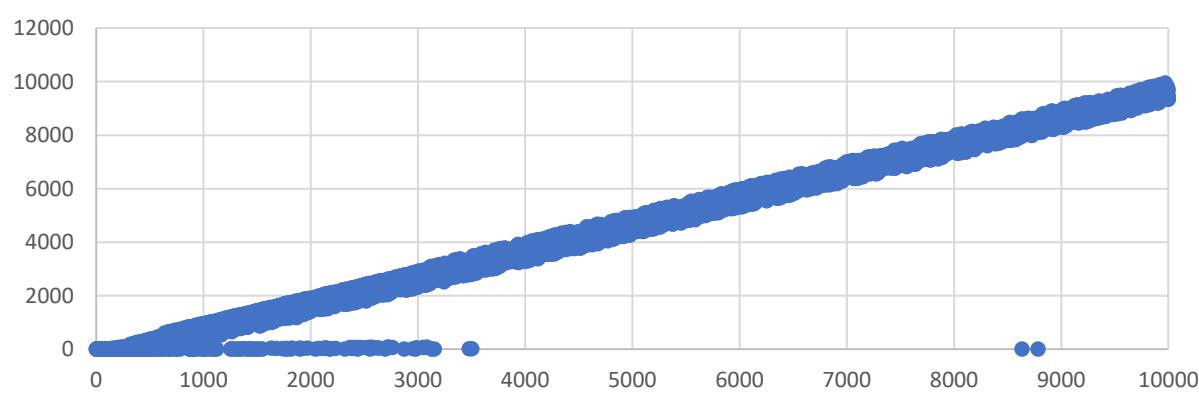
Recolección Lineal – A=1 – Tamaño 10.001 – Hash 1



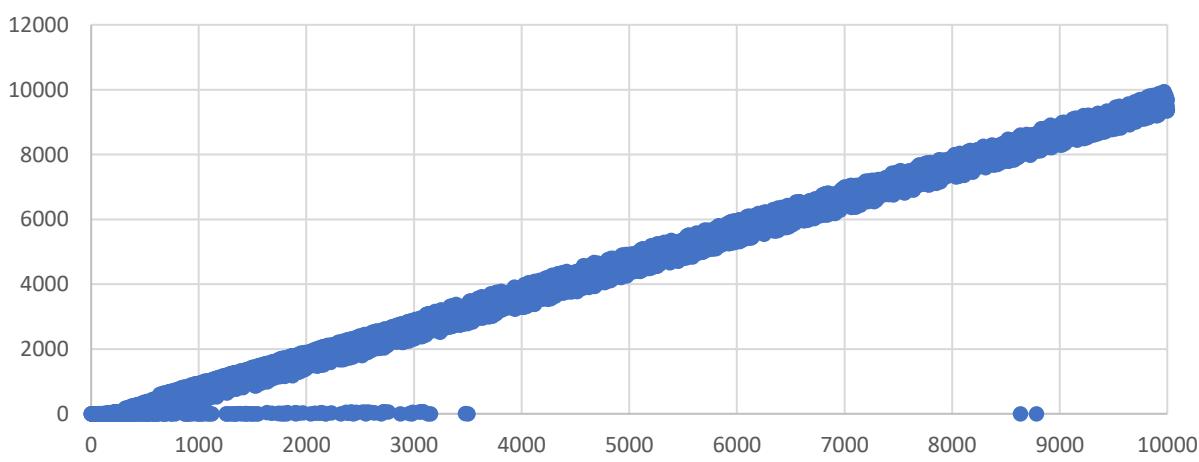
Recolección Lineal – A=1 – Tamaño 15.000 – Hash 1



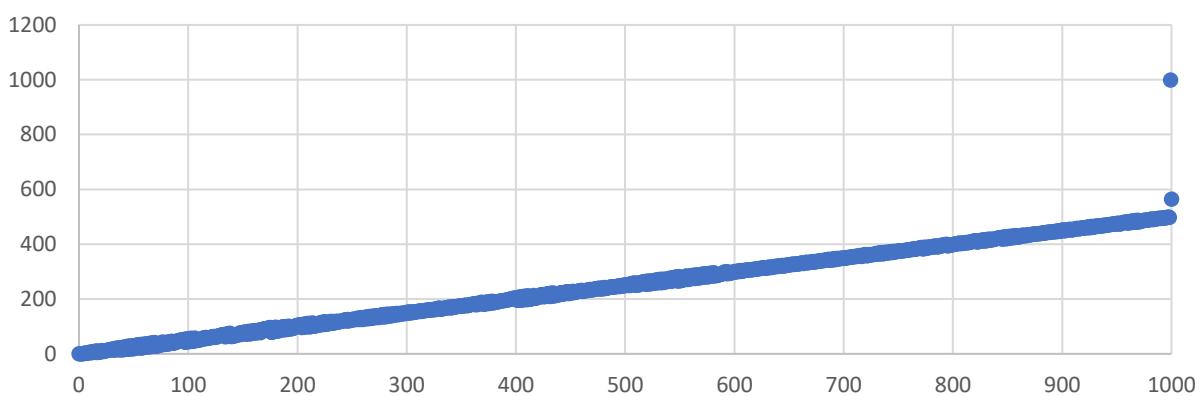
Recolección Lineal – A=1 – Tamaño 15.000 – Hash 1



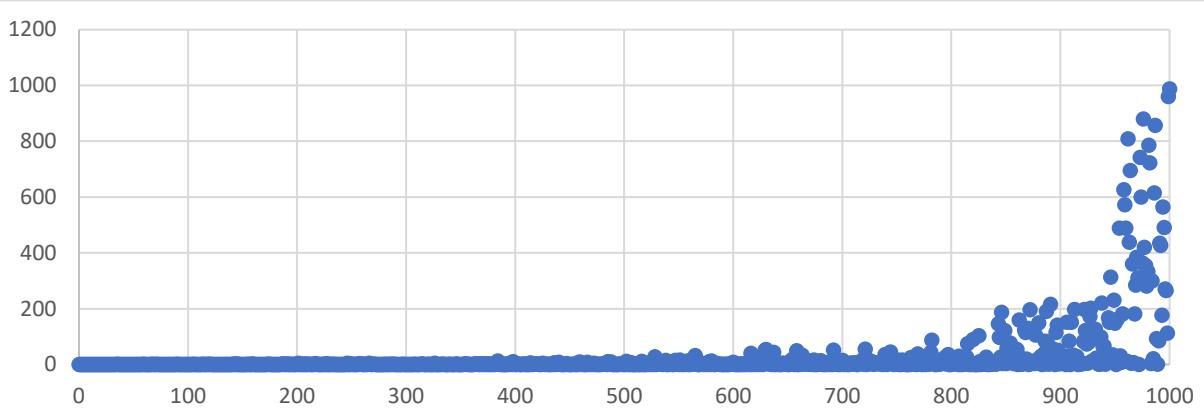
Recolección Lineal – A=1 – Tamaño 15.001 – Hash 1



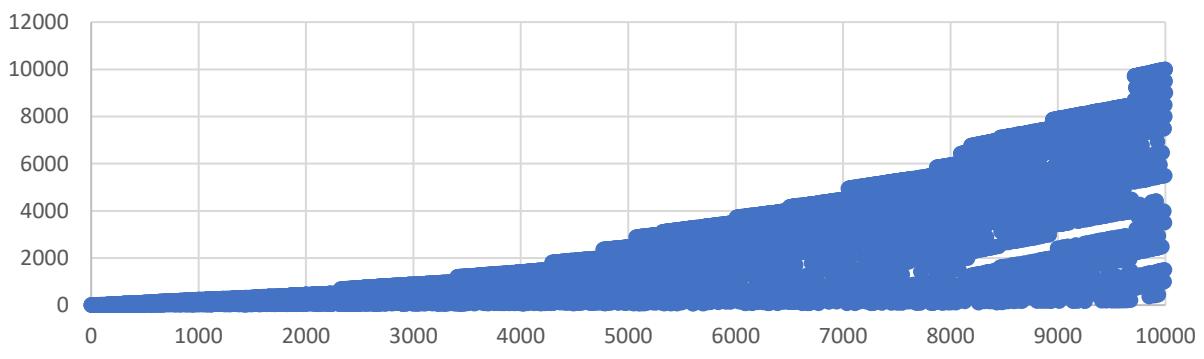
Recolección Lineal – A=1 – Tamaño 1.000 – Hash 3 – K=500



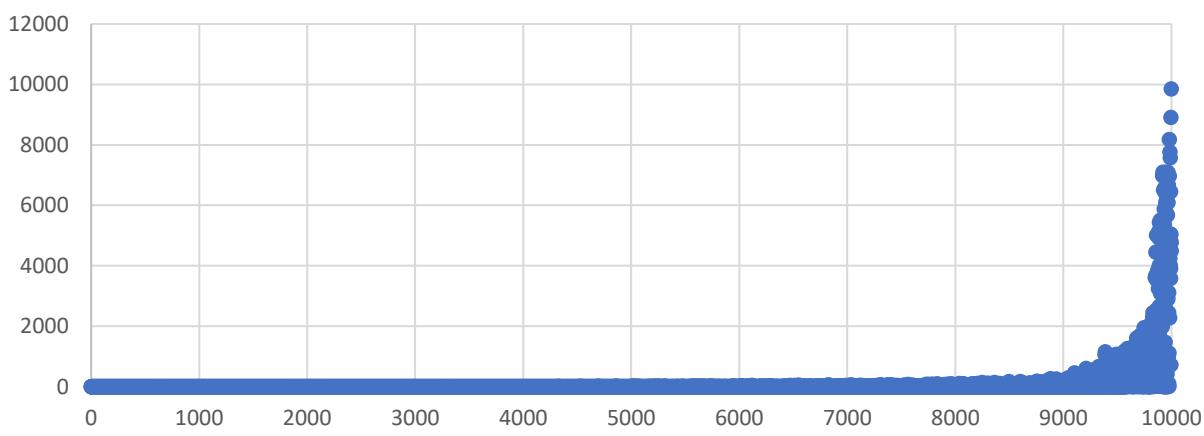
Recolección Lineal – A=1 – Tamaño 1.001 – Hash 3 – K=500



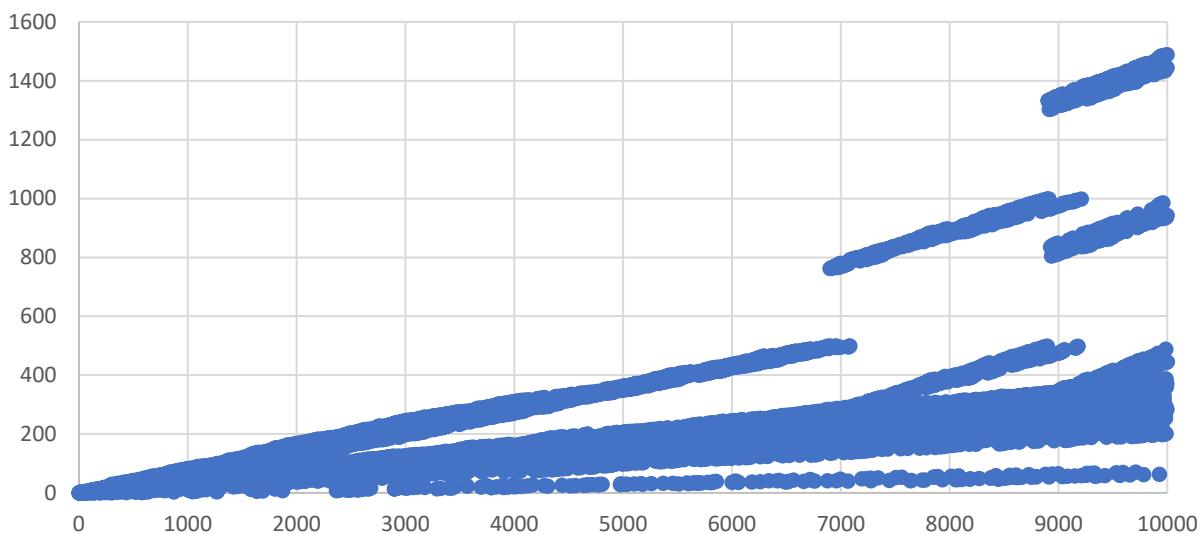
Recolección Lineal – A=1 – Tamaño 10.000 – Hash 3 – K=500



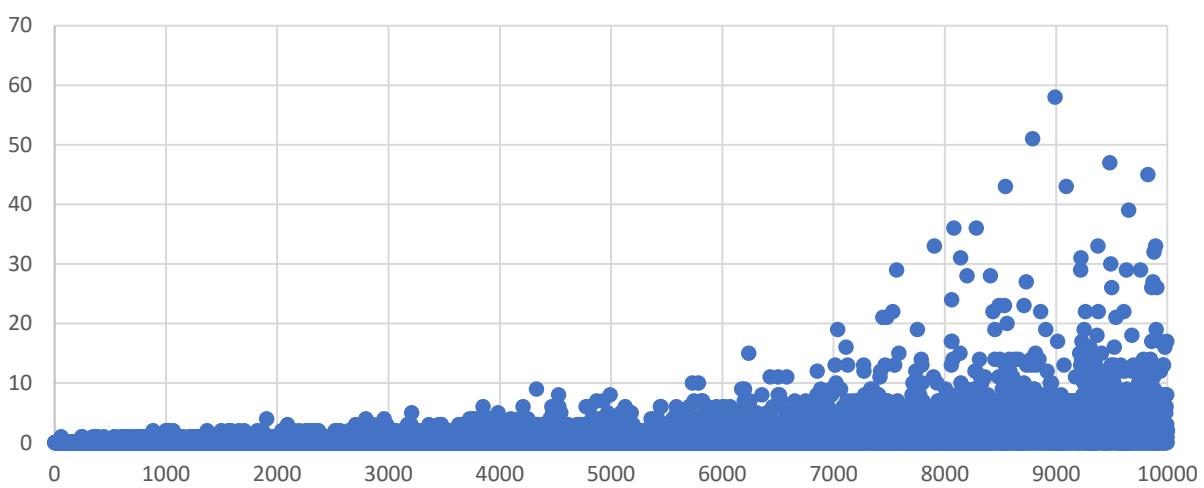
Recolección Lineal – A=1 – Tamaño 10.001 – Hash 3 – K=500



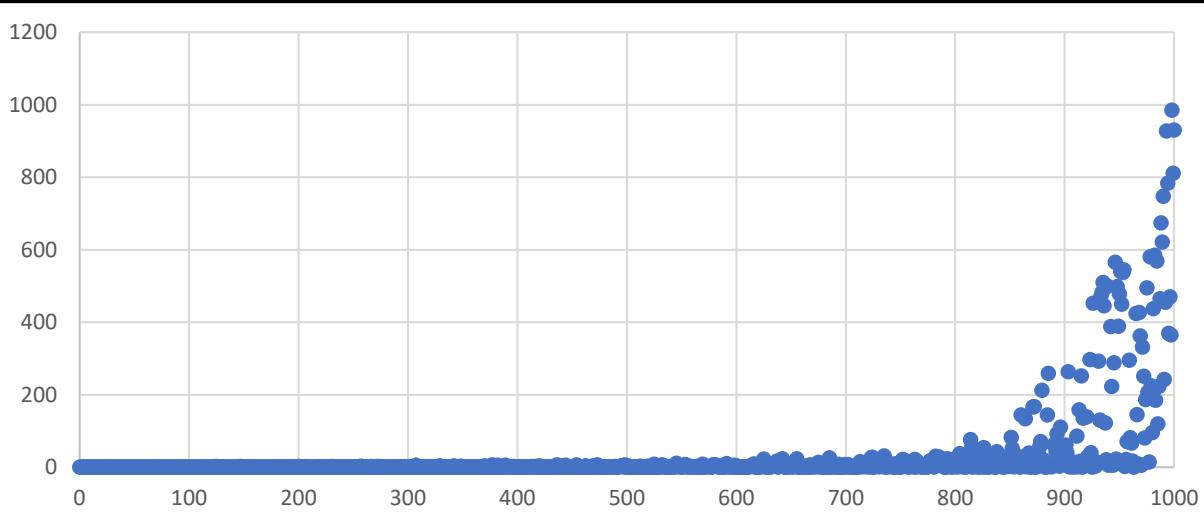
Recolección Lineal – A=1 – Tamaño 15.000 – Hash 3 – K=500



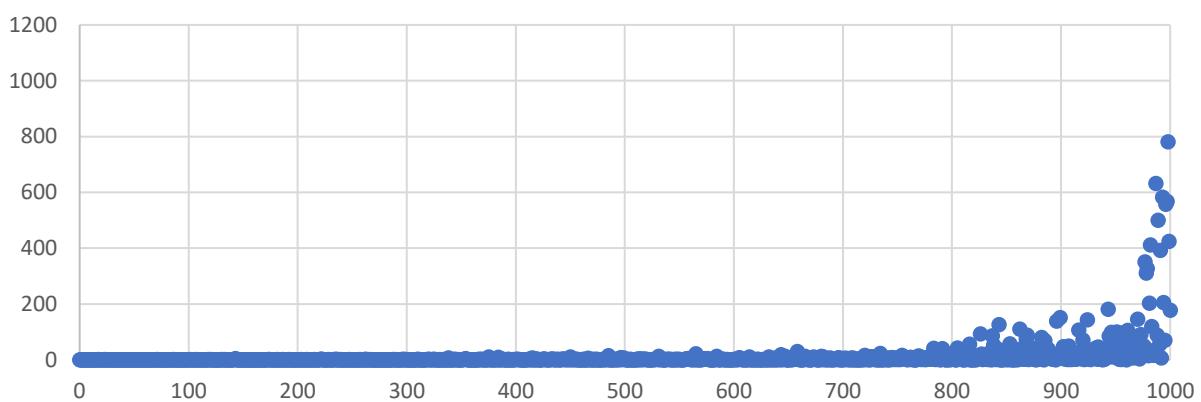
Recolección Lineal – A=1 – Tamaño 15.001 – Hash 3 – K=500



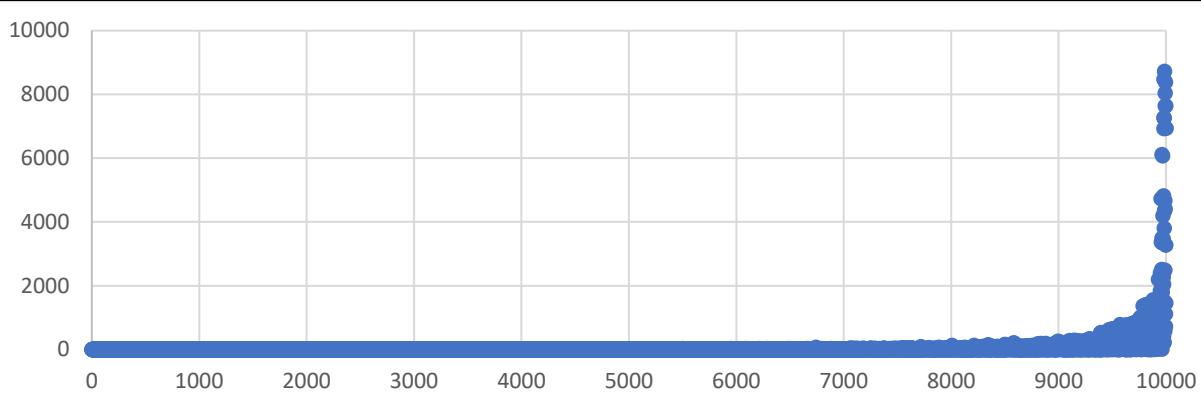
Recolección Lineal – A=1 – Tamaño 1.000 – Hash 3 – K=227



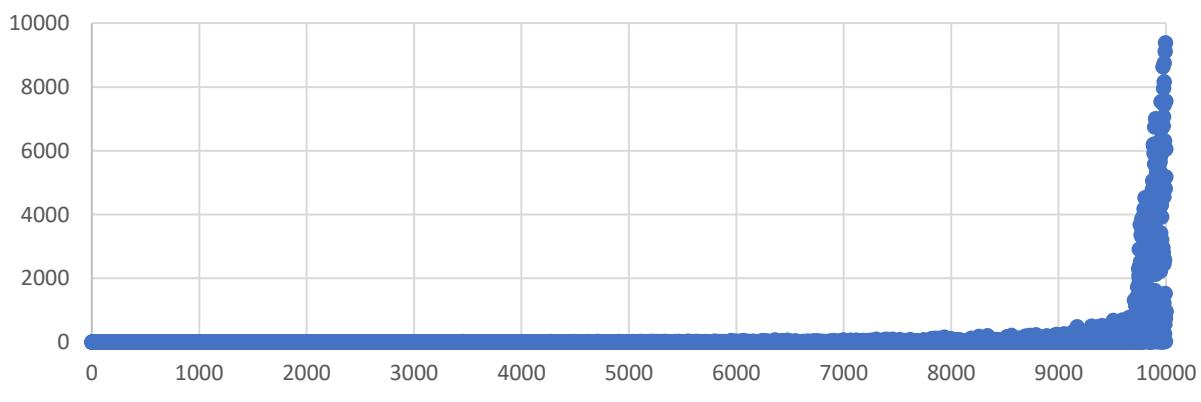
Recolección Lineal – A=1 – Tamaño 1.001 – Hash 3 – K=227



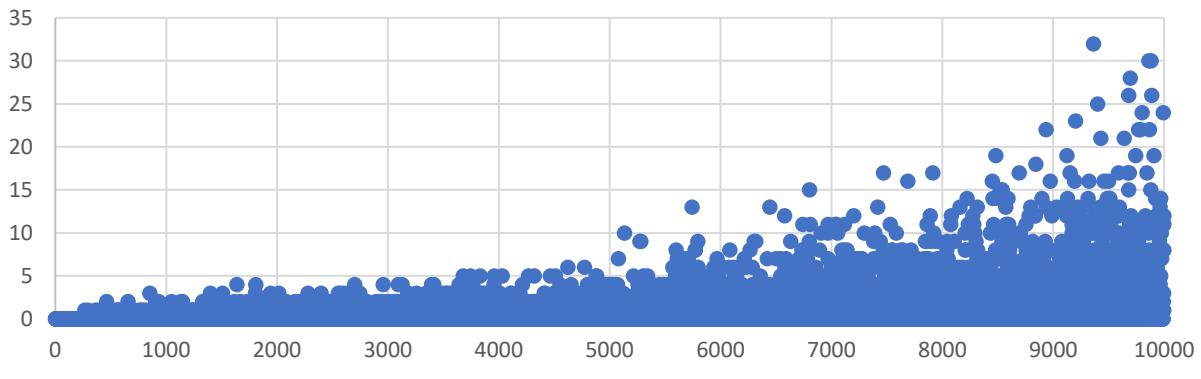
Recolección Lineal – A=1 – Tamaño 10.000 – Hash 3 – K=227



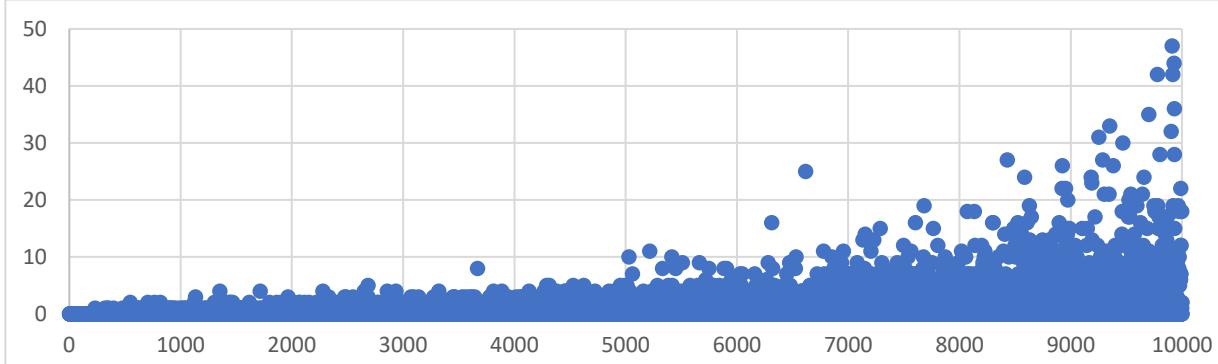
Recolección Lineal – A=1 – Tamaño 10.001 – Hash 3 – K=227



Recolección Lineal – A=1 – Tamaño 15.000 – Hash 3 – K=227



### Recolección Lineal – A=1 – Tamaño 15.001 – Hash 3 – K=227



### Conclusiones

Como bien se puede apreciar, el Hash 1 no es nada eficiente para este problema, ya que genera prácticamente el mismo número de colisiones para cualquier tamaño. Esto se debe, a que suma el valor ASCII de cada letra de la clave y obtiene el módulo del tamaño. Pero, como la clave es relativamente pequeña, nunca se llega a aplicar el módulo, lo que hace que el tamaño de la tabla no tome parte en el cálculo del hash, resultado en un algoritmo muy ineficiente.

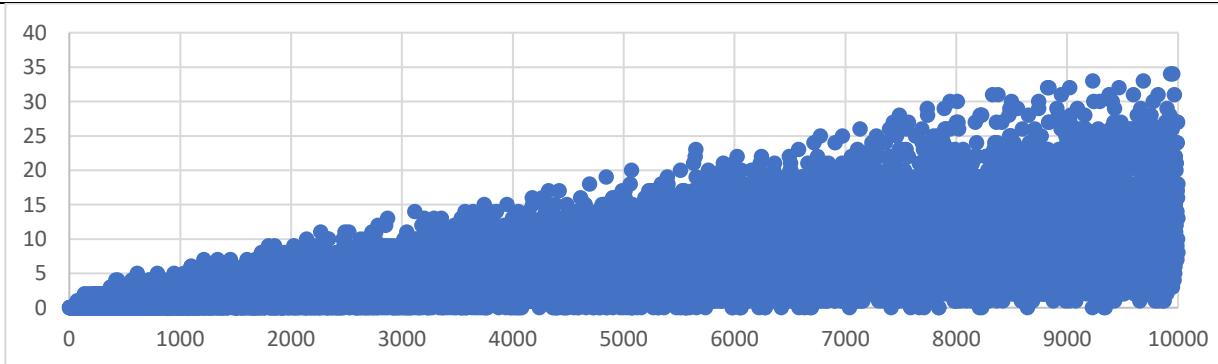
En el caso del Hash 3, se puede apreciar como hay una mejora considerable con respecto al anterior. Esta función depende de una constante K, la cual se comporta mejor con cuantos menos divisores tenga. Además, en el caso del encadenamiento, se obtuvieron muy buenos resultados ya con un tamaño de 1.000, y con la recolocación se observó cómo se comporta mejor si se deja un tamaño bastante superior al número de datos.

### Influencia de la elección de la clave utilizada para obtener la posición del jugador en la tabla

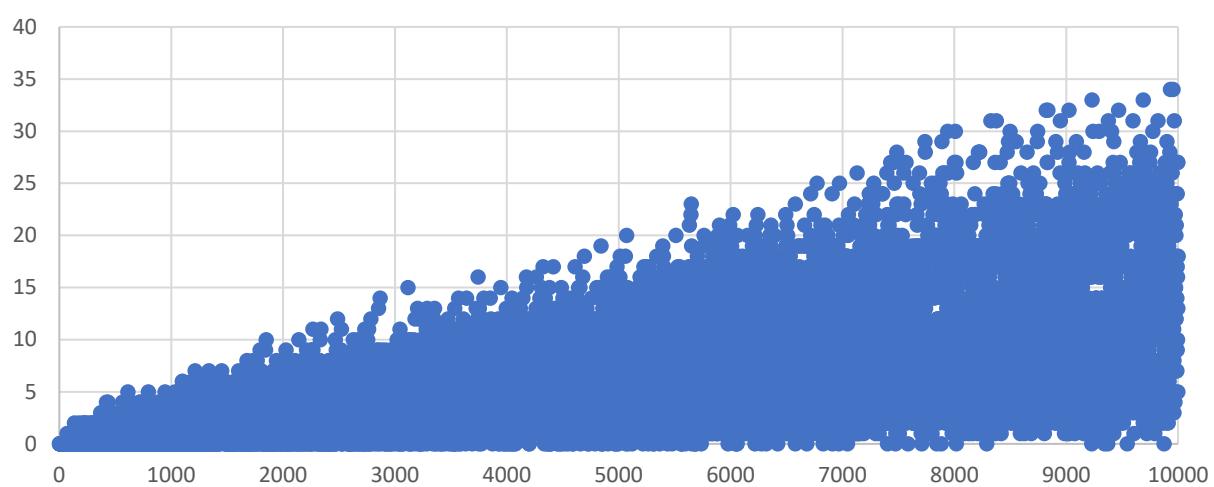
En este apartado, se cambiará la clave del nombre de usuario al email, y se espera que la función Hash 1 mejore considerablemente, ya que la longitud de la clave aumentará bastante. Por este motivo, tan sólo se realizarán pruebas con el Hash 1 (ya que se ha visto que el Hash 3 ya se obtienen muy buenos resultados).

### Encadenamiento

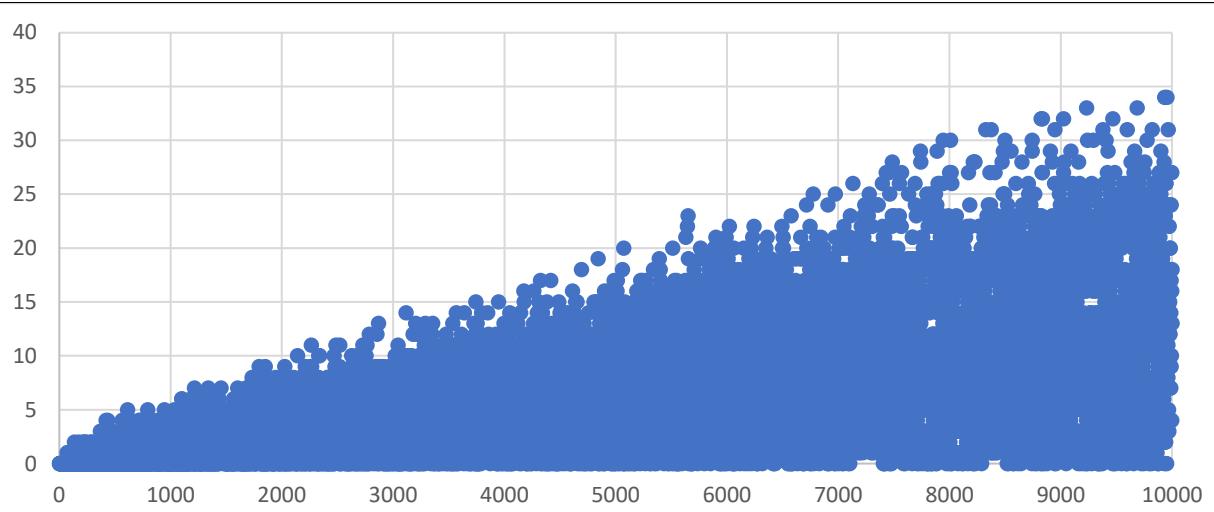
### Encadenamiento – Tamaño 1.000 – Hash 1 – Correo Electrónico



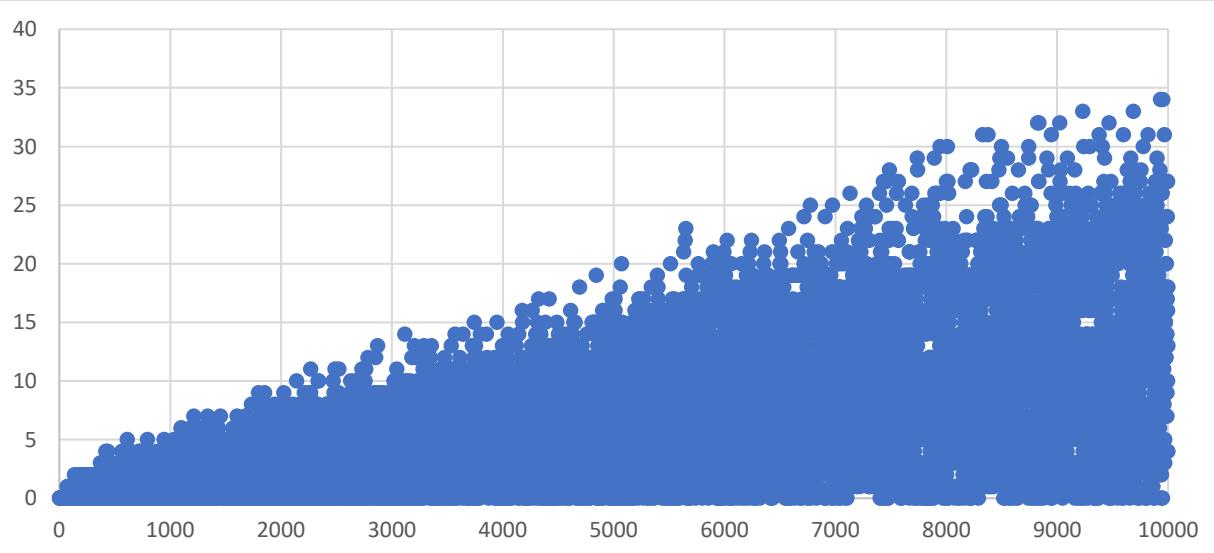
Encadenamiento – Tamaño 1.001 – Hash 1 – Correo Electrónico



Encadenamiento – Tamaño 10.000 – Hash 1 – Correo Electrónico

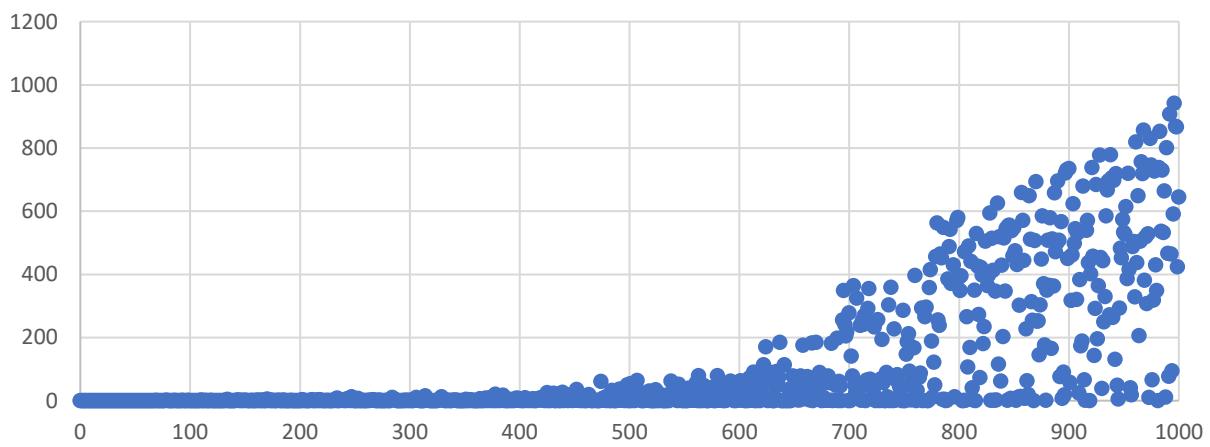


Encadenamiento – Tamaño 10.001 – Hash 1 – Correo Electrónico

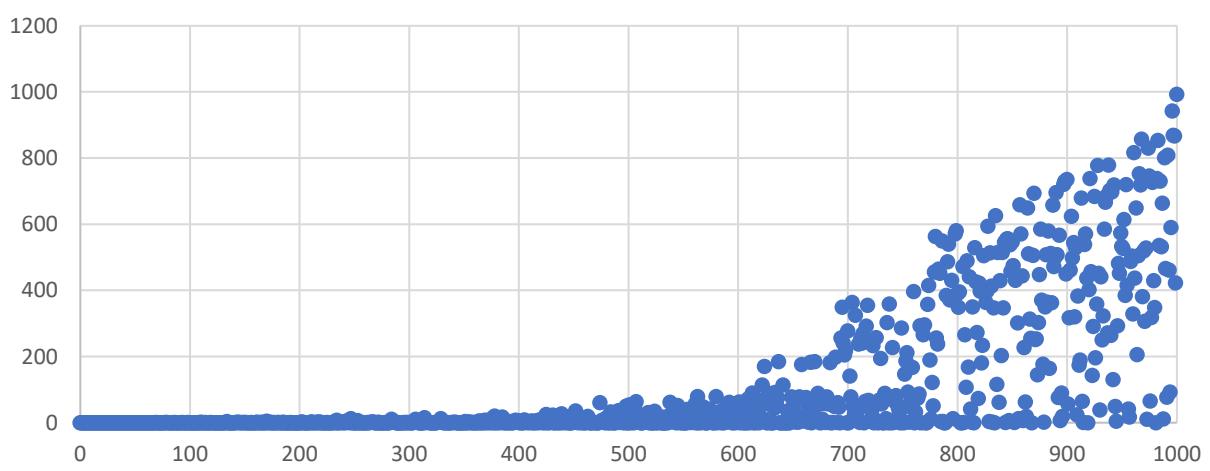


## Recolección Lineal

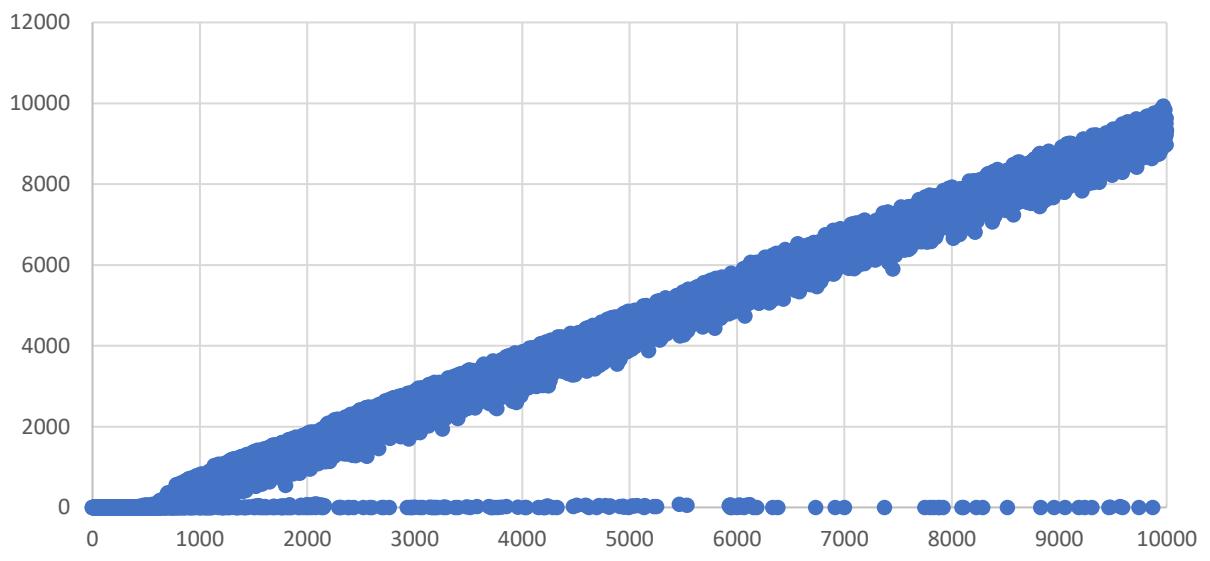
Recolección Lineal – A=1 – Tamaño 1.000 – Hash 1 – Correo Electrónico



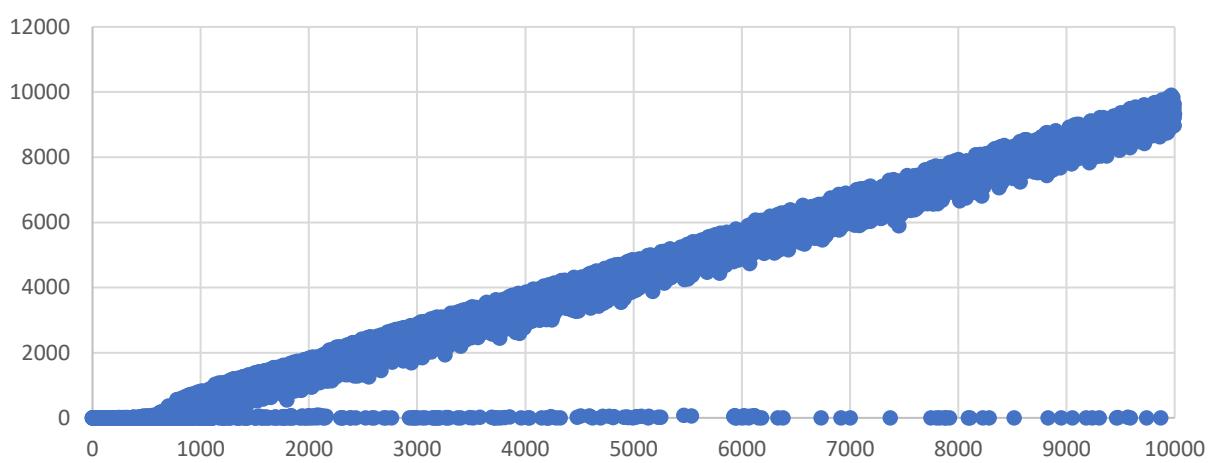
Recolección Lineal – A=1 – Tamaño 1.001 – Hash 1 – Correo Electrónico



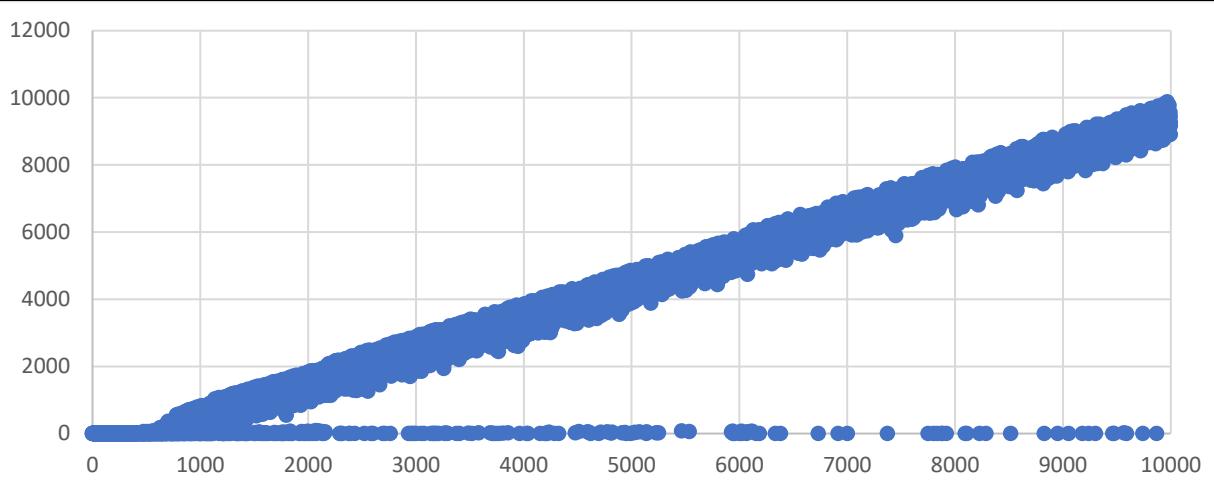
Recolección Lineal – A=1 – Tamaño 10.000 – Hash 1 – Correo Electrónico



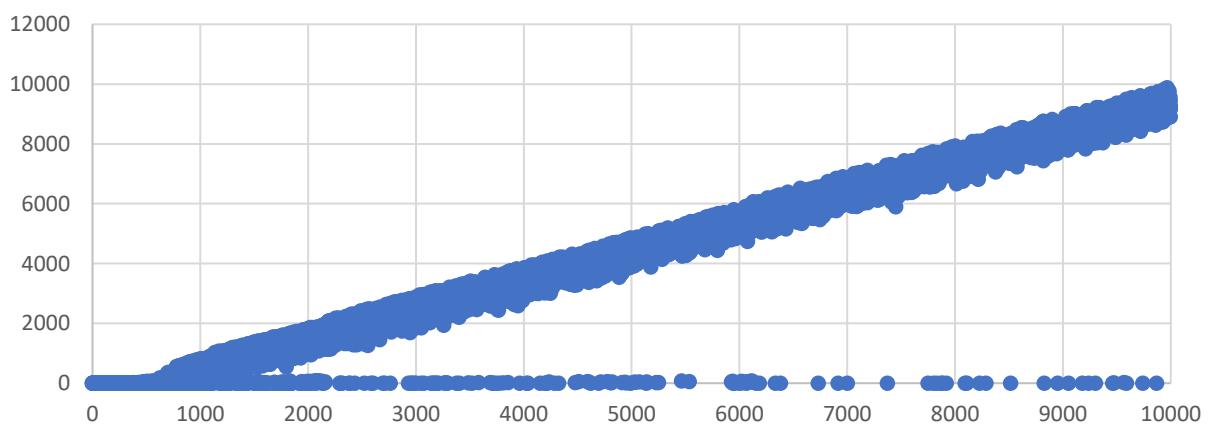
Recolocación Lineal – A=1 – Tamaño 10.001 – Hash 1 – Correo Electrónico



Recolocación Lineal – A=1 – Tamaño 15.000 – Hash 1 – Correo Electrónico



Recolocación Lineal – A=1 – Tamaño 15.001 – Hash 1 – Correo Electrónico

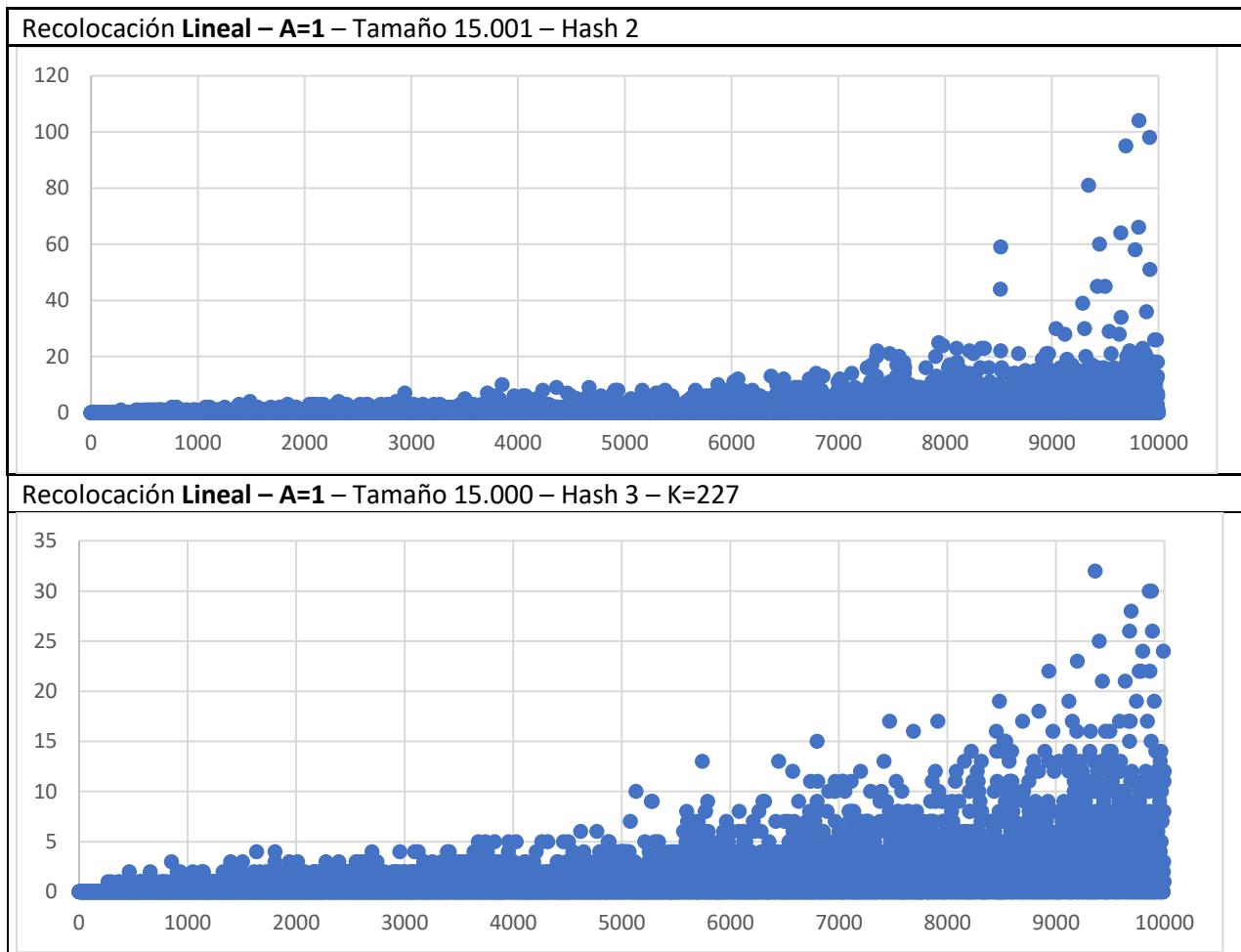


## Conclusiones

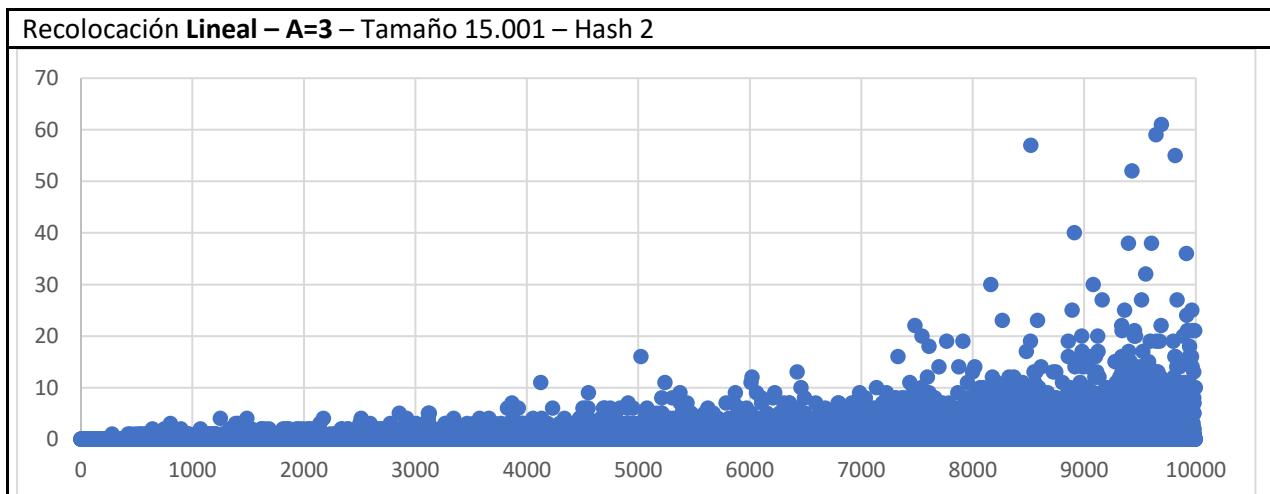
Como se puede apreciar, en encadenamiento mejora considerablemente, reduciendo casi a la mitad el número de colisiones frente a escoger el usuario (más corto) como clave. Sin embargo, en la recolocación lineal simple, no se aprecia una gran mejoría.

## Influencia de la elección de la estrategia de recolocación

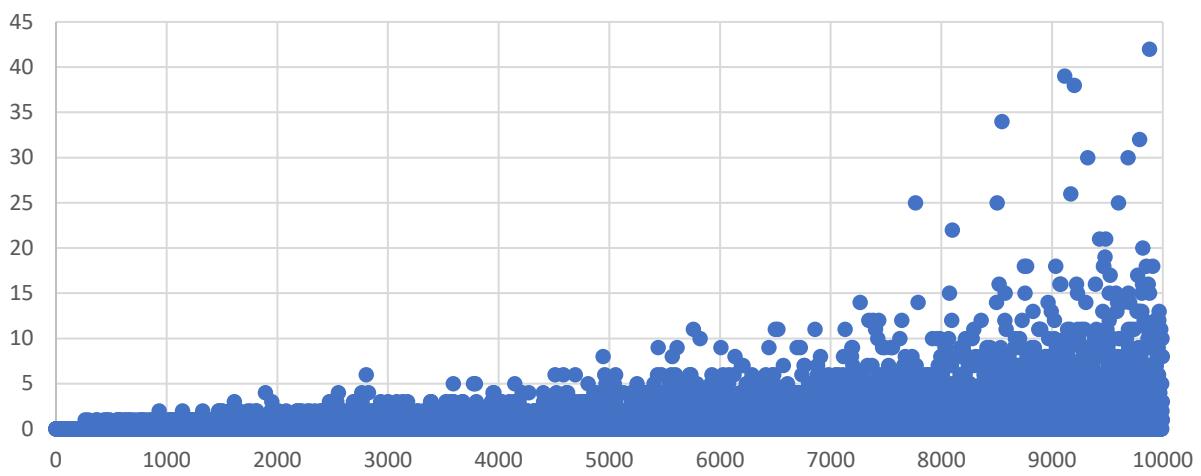
Lineal A = 1



Lineal A = 3

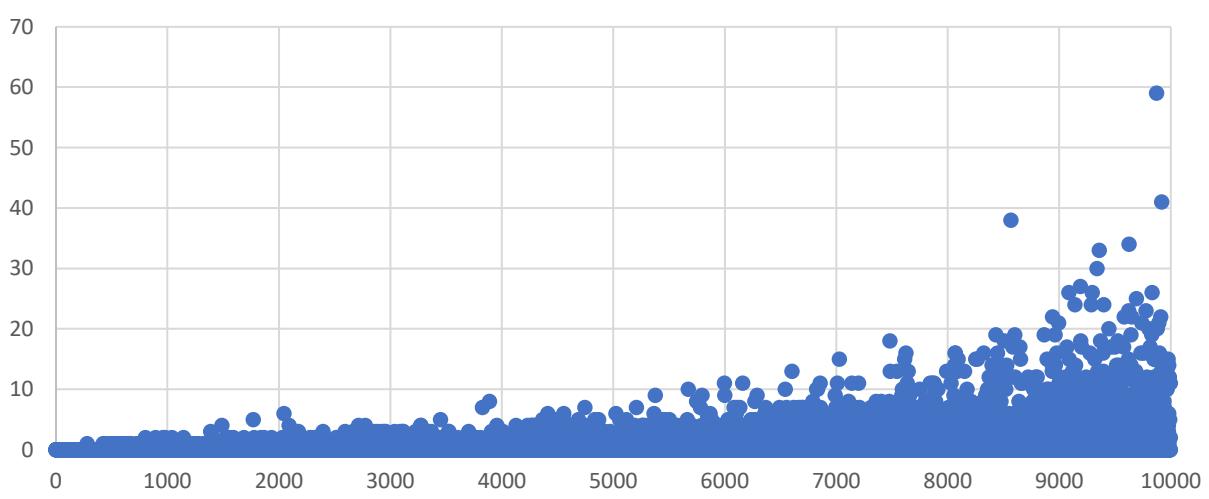


Recolección Lineal – A=3 – Tamaño 15.000 – Hash 3 – K=227

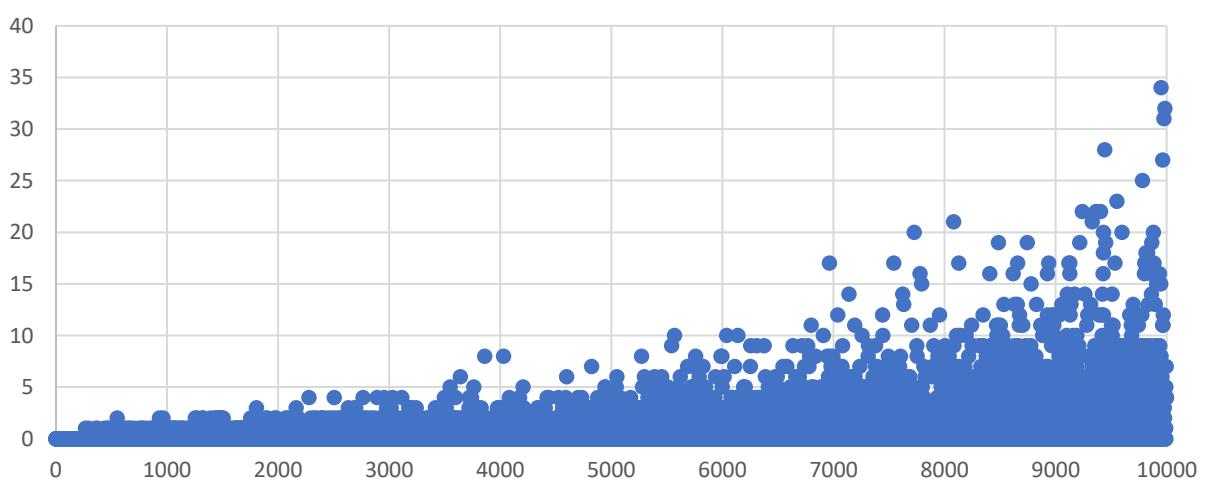


Lineal A = 7

Recolección Lineal – A=7 – Tamaño 15.001 – Hash 2

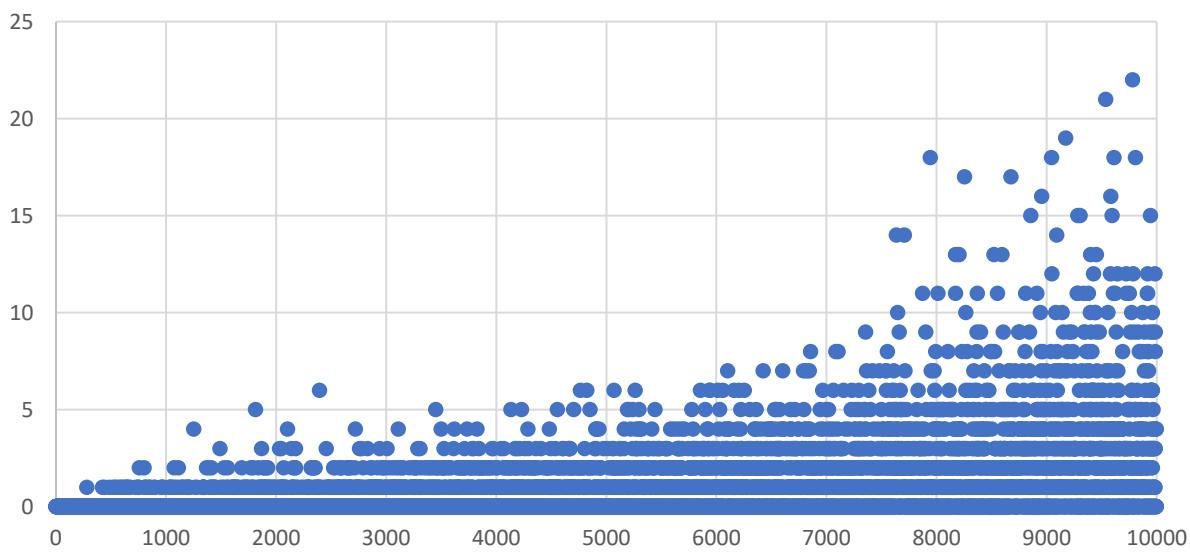


Recolección Lineal – A=7 – Tamaño 15.000 – Hash 3 – K=227

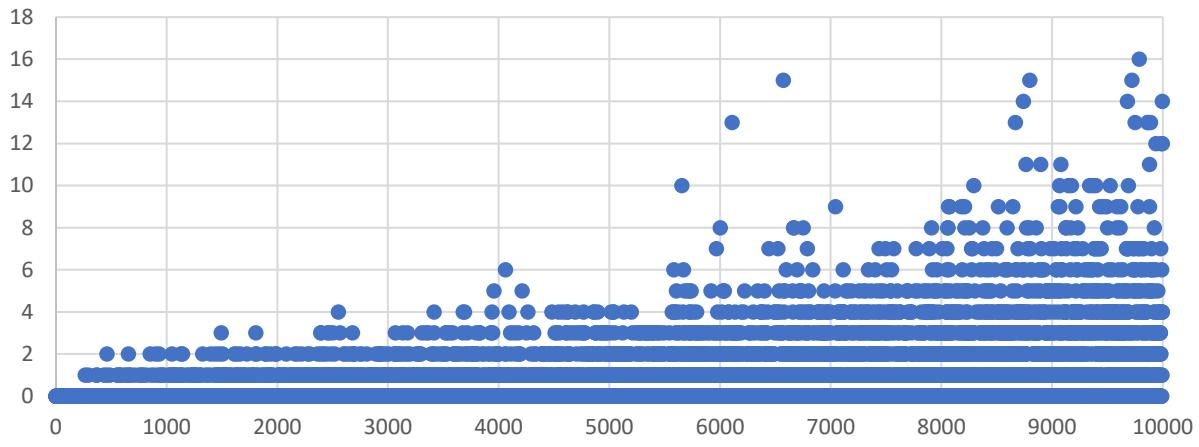


## Cuadrática

Recolocación **Cuadrática** – Tamaño 15.001 – Hash 2



Recolocación **Cuadrática** – Tamaño 15.000 – Hash 3 – K=227



## Conclusiones

Como bien se aprecia, cambiando la variable A en la recolocación lineal se puede reducir el número de colisiones, sobre todo con la función Hash 2 (ya que en la Hash 3 a veces incluso aumenta ligeramente).

Sin embargo, se consigue una mucho mejor optimización con la recolocación cuadrática, reduciendo a la mitad el número de colisiones con la función Hash 3, e incluso a un 75% en la función Hash 2. Esto hace que la recolocación cuadrática sea mucho más eficiente en estos dos algoritmos con estos tamaños especificados.

En la recolocación lineal, dado que se utiliza una constante A, puede resultar contraproducente si la función Hash devuelve varios resultados con el mismo hash, pero en la cuadrática, ya que esa “A virtual” cambia con cada búsqueda, se consigue más aleatoriedad y se consigue que haya menos conflictos.

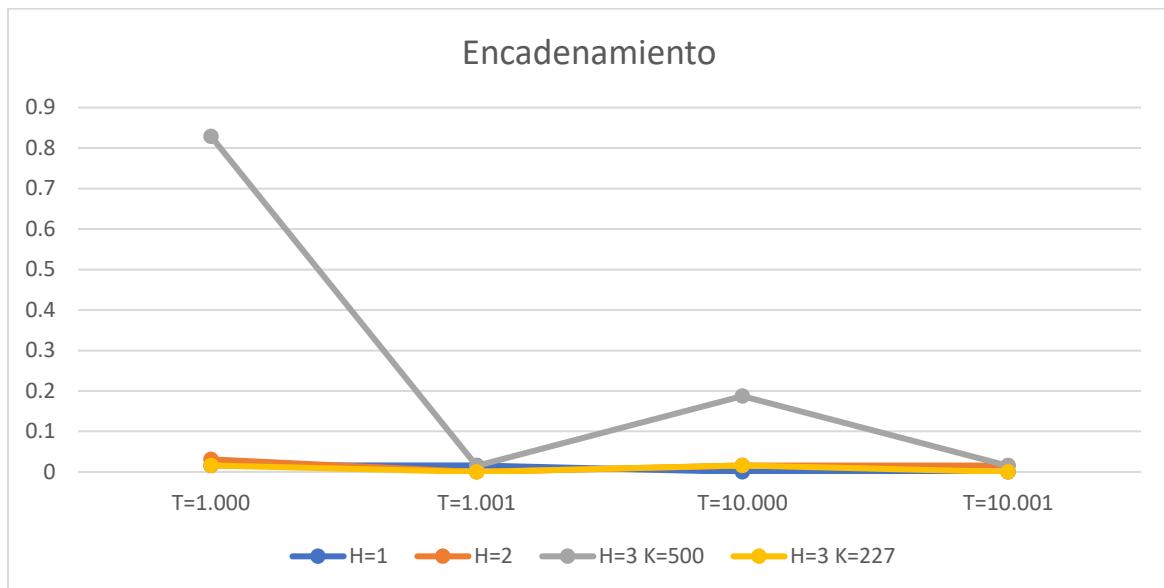
## Estimación de la eficiencia en el acceso a los datos (búsqueda)

Para este experimento, se han generado datos con todas las posibles combinaciones de las variables (Tam, HashF, A, K) y los tipos de tabla.

En el caso de la recolocación cuadrática, hubo varios casos en los que no se pudo probar debido a que generaba errores de tabla llena ya que no se recorría toda la tabla con ciertas combinaciones de tamaño.

### Encadenamiento

	T=1.000	T=1.001	T=10.000	T=10.001
H=1	0.016	0.016	0	0
H=2	0.031	0	0.016	0.016
H=3 K=500	0.829	0.015	0.187	0.015
H=3 K=227	0.016	0	0.016	0



En encadenamiento, se han obtenido resultados “relativamente malos” con un tamaño de tabla pequeño. Esto indica, que para una tabla hash que use encadenamiento, se ha de primar el tamaño de esta.

## Recolocación Lineal

**A=1**      T=10.000    T=10.001    T=15.000    T=15.001

H=1	0.469	0.422	0.422	0.531
H=2	0.031	0.015	0	0
H=3 K=500	0.218	0	0.047	0
H=3 K=227	0.015	0.016	0	0

**A=3**      T=10.000    T=10.001    T=15.000    T=15.001

H=1	0.187	0.266	0.219	0.218
H=2	0.016	0.032	0	0.016
H=3 K=500	0.234	0.016	0.141	0
H=3 K=227	0.031	0.016	0	0

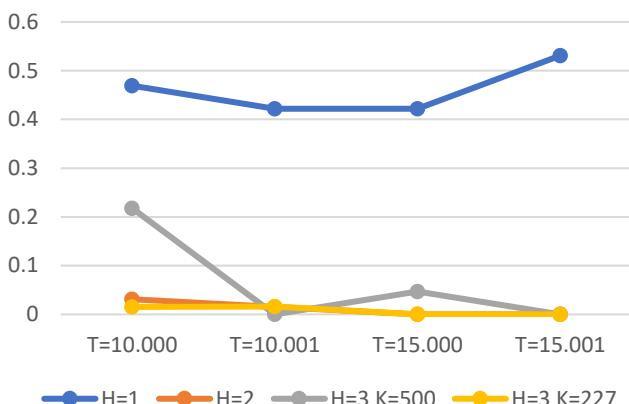
**A=7**      T=10.000    T=10.001    T=15.000    T=15.001

H=1	0.188	0.141	0.14	0.172
H=2	0.015	0.016	0	0
H=3 K=500	0.234	0	0.078	0
H=3 K=227	0.032	0.031	0	0

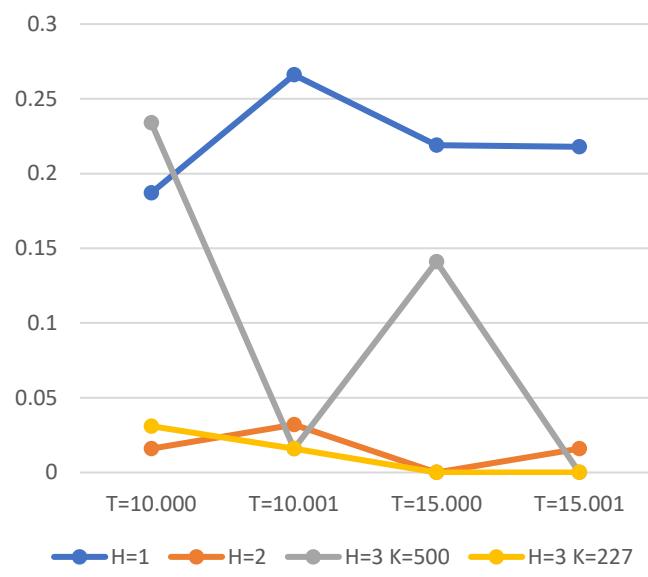
En recolocación Lineal, se puede observar como el factor A no afecta en gran medida a los resultados (tan solo se aprecian leves diferencias en la función Hash 3 con un valor de K que comparta varios múltiplos con el tamaño).

Se podría decir que, en recolocación lineal, lo más importante es la elección de la función hash.

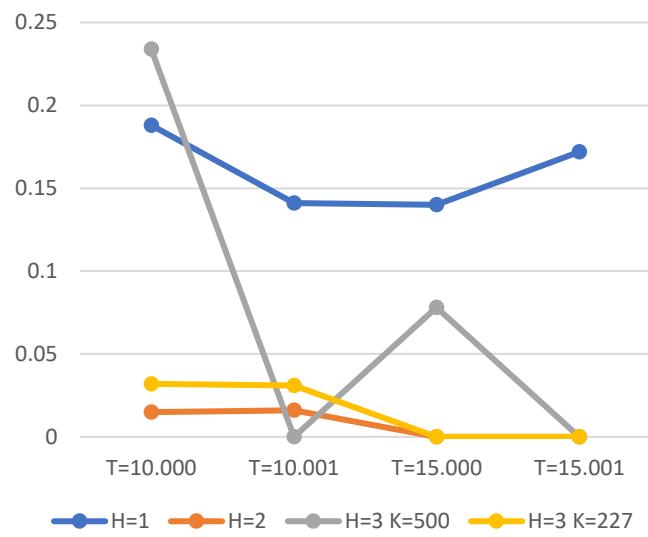
Recolocación Lineal A=1



Recolocación Lineal A=3

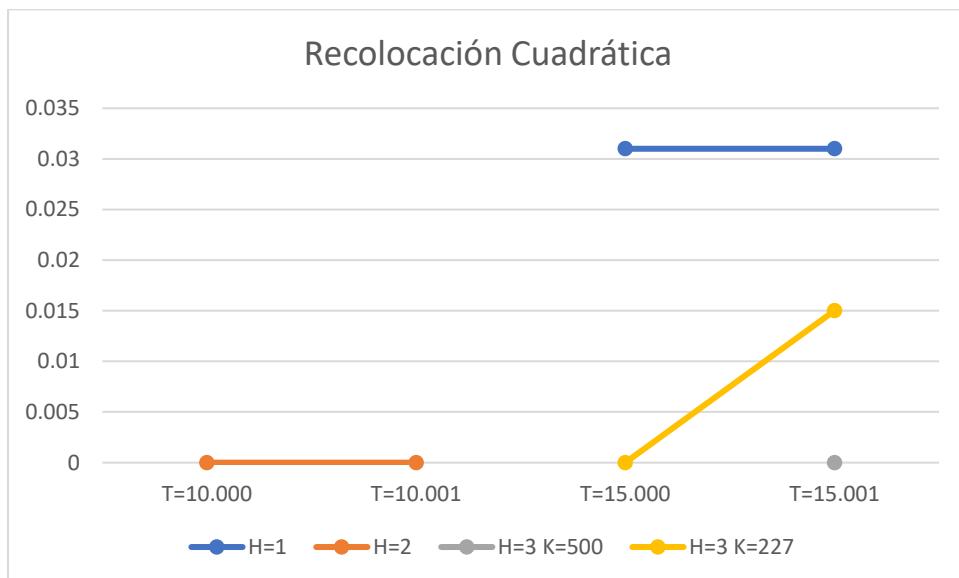


Recolocación Lineal A=7



## Recolocación Cuadrática

	T=10.000	T=10.001	T=15.000	T=15.001
H=1			0.031	0.031
H=2	0	0		
H=3 K=500			0	
H=3 K=227			0	0.015



En recolocación cuadrática, a pesar de los pocos valores, se puede ver como la función Hash 3 con un K primo va en aumento según el tamaño, mientras que con un K distinto parece que se mantiene constante. Se podría suponer que la función Hash 2 se mantendría constante a lo largo del tamaño.

En este caso, queda demostrado que en recolocación cuadrática es importante escoger un par de tamaño y función hash que no cause conflictos al recorrer la tabla, ya que si no sería ineficiente produciendo errores de tabla llena cuando no lo está en realidad.