# Trabajo Práctico Especial

**Materia:** Programación 3
**Segunda Etapa:** Eficiencia Computacional
**Integrantes:** Colamai Daniela
Martino Martínez Fabián

# INTRODUCCIÓN

En esta segunda etapa se nos pidió reducir la complejidad computacional de la búsqueda de usuarios al orden logarítmico (O(log n)) mediante la implementación de algún método que ordene la estructura, y otro de búsqueda logarítmica.
En la entrega anterior habíamos trabajado sobre dos estructuras (Listas y Arreglos de Listas), por lo que decidimos implementar para este nuevo requerimiento, la representación de usuarios mediante Arreglo de Listas a fin de facilitar las inserciones ordenadas en la estructura .Por otro lado, para optimizar las búsquedas, como estrategia, se decidió implementar la búsqueda binaria.

# DESARROLLO

Decidimos implementar un método que nos permita ingresar usuarios de manera ordenada en la estructura de menor a mayor, esto causa que aumente la complejidad temporal a O(n) siendo n la cantidad de elemento en el arreglo.
Se nos pidió también analizar el comportamiento de la propuesta elegida teniendo en cuenta los siguientes escenarios:

**a) Hay pocas altas de usuarios y muchas verificaciones de existencia**

Creemos haber aplicado una posible solución para este caso, más adelante daremos más detalles sobre esta estructura.

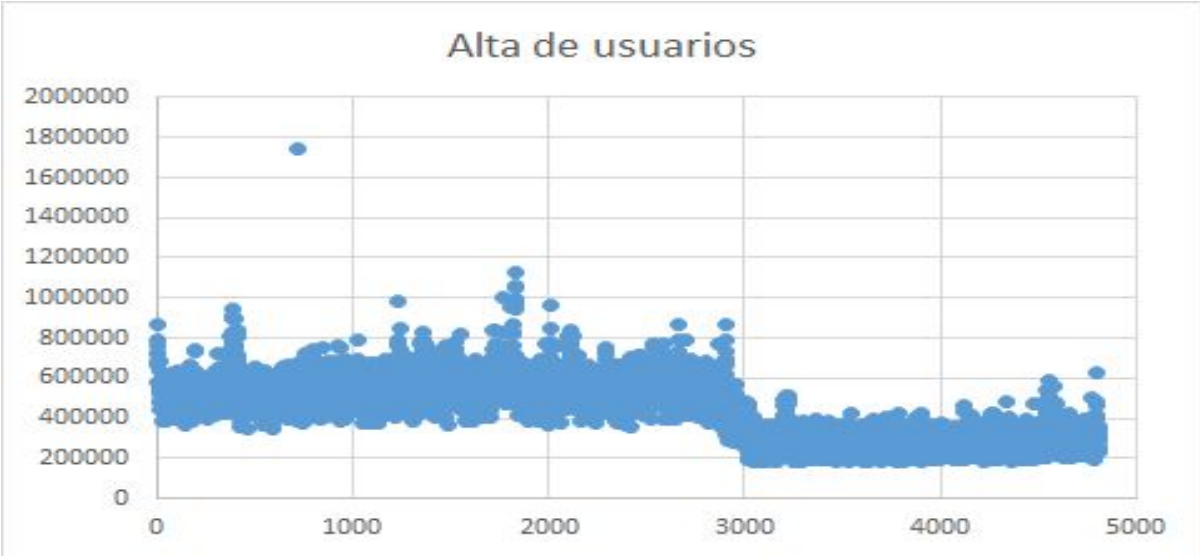
**b) Hay muchas altas de usuarios y pocas verificaciones de existencia**

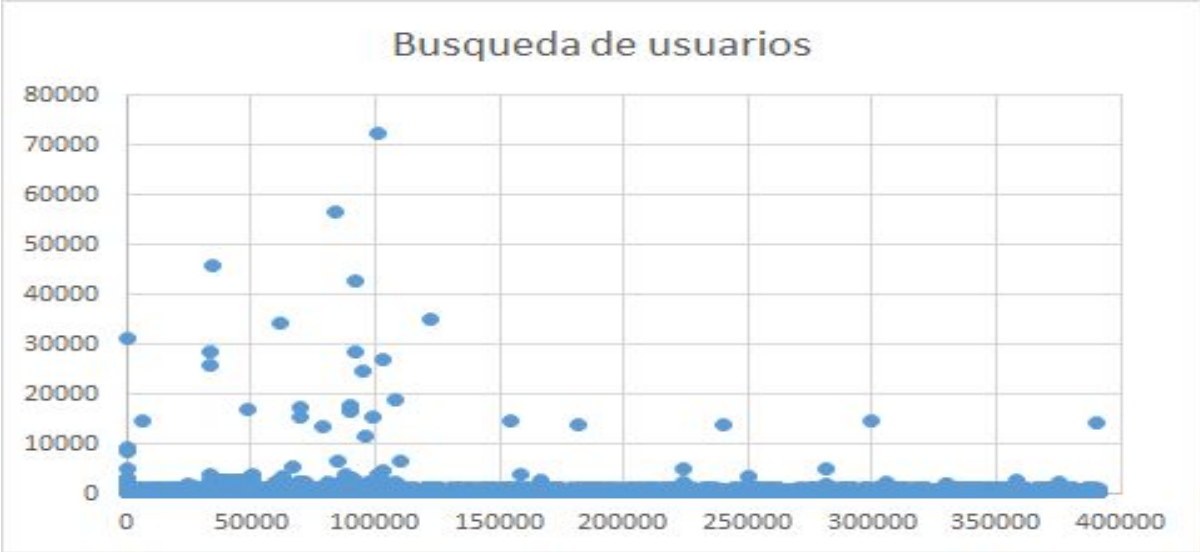
Para este caso es importante reducir la complejidad del alta lo más posible, por lo que el método de ordenamiento sería aplicado de manera diferente y reduciendo la cantidad de llamados para reducir la cantidad de veces que se recorra la estructura.
En el caso de la búsqueda si la diferencia entre altas y verificaciones es muy alta podría darse el caso de que sea más eficiente tener la estructura desordenada y realizar búsquedas secuenciales debido al costo del ordenamiento de los elementos.

CASO A:

Gracias a los siguientes gráficos, podemos apreciar la complejidad temporal tanto del alta como de la búsqueda de usuarios. El alta debe recorrer el arreglo entero ya que debe encontrar la posición a la que pertenece el nuevo usuario y realizar un corrimiento de los que poseen un ID mayor.

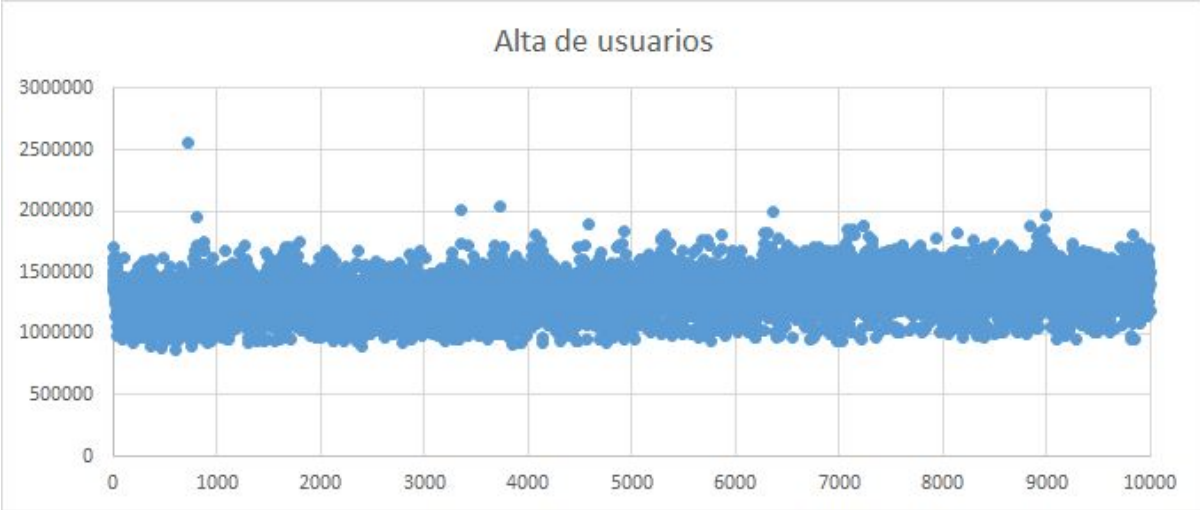
Debido a que la estructura se encuentra ordenada podemos ver que los tiempos de búsqueda son menores a los que serían en el tipo de búsqueda secuencial.

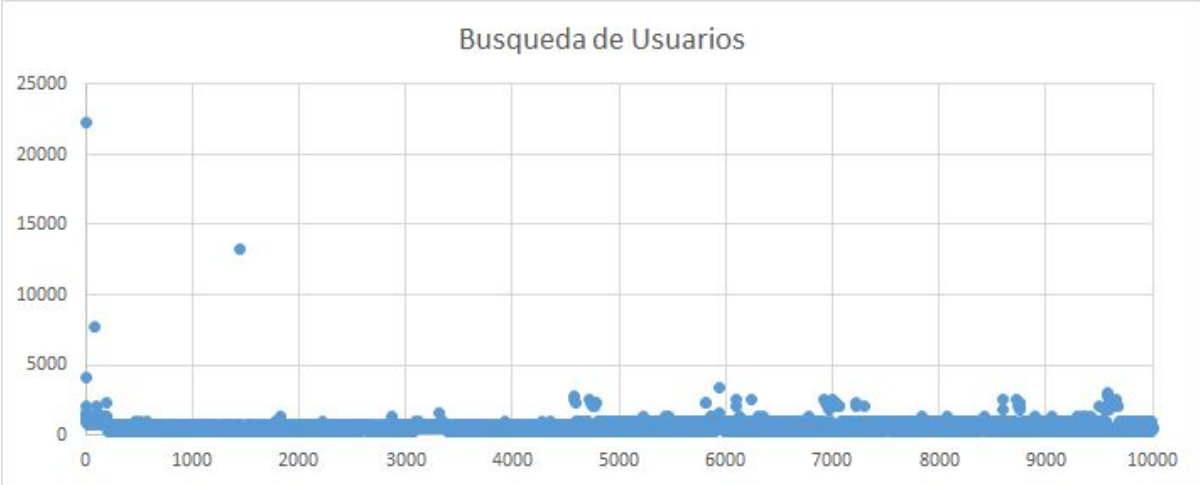


CASO B:

En este caso, los beneficios que posee el CASO A no se pueden ver tan claramente debido a que la complejidad que trae el dar de alta tantos usuarios opaca a la complejidad que posee la búsqueda. Los tiempos de búsqueda de ambos son similares gracias al formato de búsqueda binaria, pero el ordenar el arreglo cada vez que se da el alta de un nuevo usuario, genera un aumento notable en el tiempo de ejecución.



Luego del análisis de los casos a) y b), se nos planteó evaluar en el punto a) la utilización de la estructura de árbol binario de búsqueda.
Es posible usar un árbol binario para cumplir con las consignas de este trabajo pero solo si este se encuentra balanceado, la complejidad de la búsqueda sería la misma pero el problema se da a la hora de mantener el árbol balanceado cuando se da un alta.
Este punto no sería conveniente realizarlo con un árbol de búsqueda binaria ya que sería más costoso por el hecho que ya habíamos mencionado, de mantener el árbol balanceado.
Se puede en un árbol binario de búsqueda efectuar ciertos controles para mantener la estructura balanceada.
El grado de balance depende del orden en que se van a insertar los nodos.
En el árbol la altura de su subárbol izquierdo y de su subárbol derecho no deben diferir en más de 1.
En caso de que no se encuentre balanceado es necesario cambiar el nodo raíz y reordenar el árbol para que queden parejas la cantidad de subárboles de cada lado.
Si quisiéramos crear un método que en un árbol binario nos permita realizar una inserción de manera que el árbol quede de manera balanceada sería insertando de manera ordenada y a continuación se realiza una revisión para ver si el árbol sigue balanceado, de ser negativo se ejecutaría el método para balancear el árbol. El balanceado funcionaria de la siguiente manera:

- se corrobora cuál es la rama más grande y se toma como pivote
- chequea si añadiendo la cantidad de elemento perteneciente a la otra rama el pivote podría quedar como un árbol balanceado. De ser negativo se repite el proceso hasta encontrar un pivote válido o llegar a la hoja que “se encuentra en el centro”.
- se reorganiza el árbol utilizando el pivote como la nueva raíz para que tengo la misma cantidad de subárboles en cada rama.