Programación 3 - TUDAI 2017

Trabajo Práctico Especial

Integrantes del Grupo:

Marcelo, Prado

Eduardo, Bravo

Fecha:

11/05/2017

Introducción:

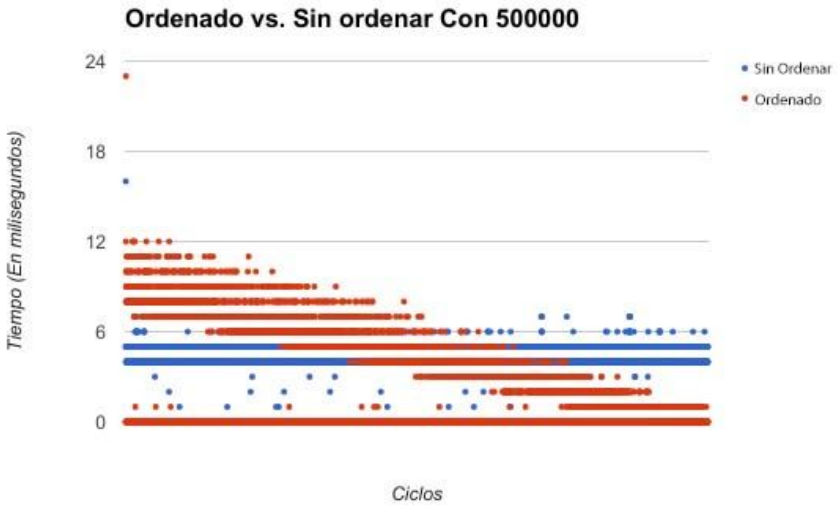
Se nos pide reducir la complejidad computacional (el tiempo insumido) de la operación al orden logarítmico (O(log n)) mediante la implementación de algún método que ordene la estructura y otro una búsqueda logarítmica que pueden ser una búsqueda binaria u otra técnica.
Seleccione la estructura de representación de los usuarios y una estrategia que optimice la operación de verificación de existencia (búsqueda). Implemente estas operaciones siguiendo la propuesta elegida y justifique.

Decisiones de diseño:

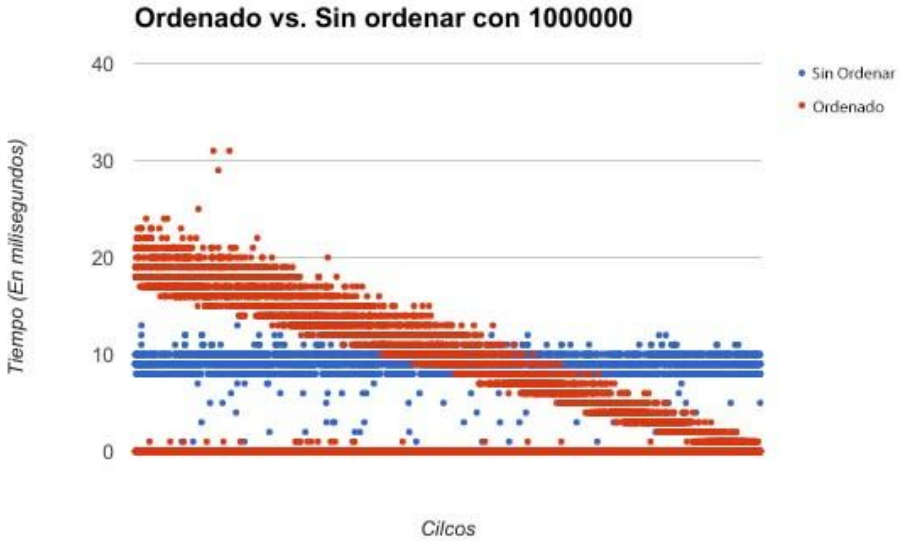
Ante la solicitud de reducir los tiempos computacionales, hemos decidido usar la estructura de Arreglo para mejorar dichos tiempos. El algoritmo de ordenamiento que usaremos es quicksort sobre la estructura y los datos de búsquedas los ordenamos de la misma manera, de esta forma creamos una estrategia (greedy) de búsqueda por rango, Con esta estrategia NO recorremos el arreglo completo cada vez que buscamos un nuevo objeto. Por lo contrario, al tener las búsquedas ordenadas, una vez que encontramos un objeto en nuestro arreglo, ya no es necesario fijarnos en los elementos anteriores del arreglo.
Para aplicar esta técnica agregamos al objeto Arreglo un nuevo método que nos recorre el arreglo desde un índice a otro pasados por parámetros y a su vez dicho método nos retorna la posición del arreglo en el caso que lo contenga.

Resultados de la técnica aplicada:

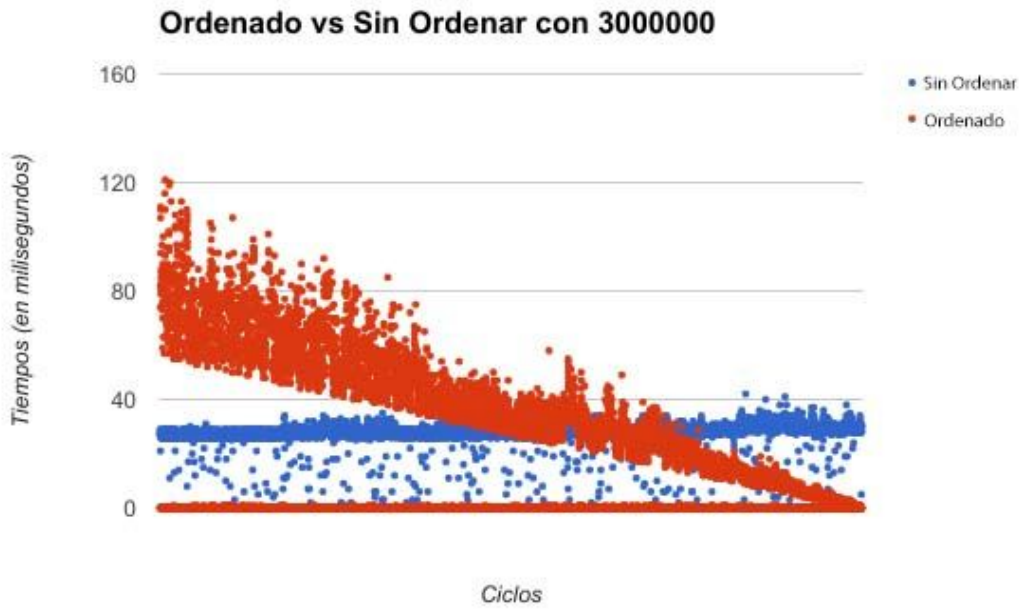
Con 500000 Usuarios			
	Peor Busqueda	Promedio Búsqueda	Total Busqueda
Sin Ordenar	16 miliseg	4 miliseg	Minutes: 0 Seconds: 44 Milisec: 388
Ordenada	23 miliseg	3 miliseg	Minutes: 0 Seconds: 31 Milisec: 66



Con 1000000 Usuarios			
	Peor Busqueda	Promedio Búsqueda	Total Busqueda
Sin Ordenar	13 miliseg	9 miliseg	Minutes: 1 Seconds: 31 Milisec: 174
Ordenada	31 miliseg	6 miliseg	Minutes: 1 Seconds: 8 Milisec: 961



Con 3000000 Usuarios			
	Peor Busqueda	Promedio Búsqueda	Total Busqueda
Sin Ordenar	42 miliseg	28 miliseg	Minutes: 4 Seconds: 41 Milisec: 937
Ordenada	121 miliseg	25 miliseg	Minutes: 3 Seconds: 51 Milisec: 376



En modo de conclusión vemos que con la técnica aplicada sobre los distintos dominios de entrada reduce el tiempo total y promedio de las búsquedas. En las primeras busquedas resulta más costoso buscar en los productos ordenados pero avanzando lo ciclos, tiempo disminuye considerablemente ya que la búsqueda va reduciendo el tamaño del arreglo con los datos.

Al analizar el comportamiento de la propuesta elegida teniendo en cuenta los siguientes escenarios:

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

Notamos que en el caso a) nuestra propuesta se mantiene con tiempos estables ya que al no haber demasiadas altas el arreglo no crece pero en el escenario b) la situación cambia

porque al crecer la cantidad de usuarios nuestra estructura crece y la búsqueda cada vez tarda más en encontrar los elementos.

Para mejorar este ultimo escenarios nuestra solución sería implementar un árbol binario balanceado.
Se define el factor de balance como el alto del subárbol derecho menos el alto del subárbol izquierdo. Entonces en un árbol AVL, todos los nodos cumplen la propiedad de tener valores del factor de balance iguales a: -1, 0, ó +1. Sea h n el mínimo número de nodos en un árbol AVL de altura h dada, que se encuentra en su peor caso de desbalance, si se agrega un nodo, tal que la nueva altura sea (h+1), dejan de ser AVL.

Preguntas:

1) Será posible utilizar un árbol binario de búsqueda para el caso a) ?

Sin problemas, buscando una buena raíz para garantizar un buen balanceo.

3) Sería conveniente hacerlo?

Seria una buena opción para las búsquedas.

4) Qué pros y contras le ve a esa solución?

Entre los pro encontramos el costo minimo de búsqueda e inserción y entre los contras podemos encontrar la dificultad del balanceo.

5) Será posible encontrar alguna forma de garantizar que cuando cargamos los datos en una
estructura de árbol, éste resulte en un árbol balanceado? Si fuera posible, describa cómo lo haría?

Si es posible, usando Operaciones de re-balanceo con rotaciones a derecha o a izquierda.