Desempeño Comparativo de Algoritmos de Ordenamiento

**Por:** *Santiago Serna Zapata.*

Siempre ha sido bien conocido el hecho de que no es lo mismo que te cuenten algo a experimentarlo *de facto.* Pues bien, este hecho se evidenció en la experiencia del autor al realizar la actividad. A continuación, se procederá a describir dicha experiencia con sus momentos más sobresalientes:

Como primer paso, para poder describir bien las posibles dificultades vividas, así como momentos remarcables, se debe mencionar brevemente la metodología seguida: primero, se comenzó con la codificación de ciertas operaciones necesarias para la prueba, tales como realizar TestingSort o imprimir Estadisticas, las cuales permiten la ejecución de la prueba dependiendo del tamaño de los arreglos input, así como del algoritmo usado, para que finalizada la misma, se imprimieran la fecha y hora de inicio, final y el tiempo de ejecución junto con las operaciones realizadas en el algoritmo. Finalizado esto, se procedió a crear el archivo con los arreglos input, y fue aquí donde surgió la primera dificultad. Los arreglos usados fueron un par desordenado y otro ordenado para cada tamaño, el cual varió de 10, 100,1000, 10000 y 1’241.118 elementos. Estos fueron creados con Excel© haciendo uso de la función ALEATORIO.ENTRE( ). Al momento de guardarlo como documento de texto (.txt) se supone que los símbolos quedan delimitados por tabulaciones, dejando saltos de líneas solo donde debería para hacer más fácil su lectura con las funciones de *stdlib.h.* Sin embargo, en las pruebas del proceso de lectura del archivo, se imprimían los elementos del arreglo que se iba llenando y para sorpresa, estos elementos eran números muy grandes. Nada que ver con los generados con Excel.

Después de revisar estos elementos, se llegó a la conclusión de que Excel convertía los caracteres numéricos de la hoja literalmente en los símbolos que representaban el sistema de codificación ANSI, por lo que *fscanf* los leía como un numero aparentemente muy grande, que solo era la concatenación de varios números enteros seguidos. En base a esto, se decidió pegar directamente los números de Excel a un Notepad abierto directamente para irlos guardando, pues así se pegaban directamente como los valores de la hoja misma. Todo transcurría sin problema, pero conforme aumentaba el tamaño de números a pegar, el Notepad tardaba unos segundos en poder pegarlos (pues se saturaba el portapapeles). Cuando llego el turno del millón de datos, el Notepad se ralentizó unos buenos segundos tan siquiera para guardar.

Para colmo, esta pesadilla estaba lejos de terminar. Debido a que esta vez no se guardaban los datos desde el mismo Excel (quien se encarga de acomodar con espacios y newlines donde corresponde), ahora era el turno de la acomodación manual. Como solo se usó un mismo archivo para todos los arreglos, las newlines eran necesarias como un punto de referencia para saber, dependiendo de la prueba, desde donde comenzar a leer. Así que como sospecharan, el gestar el archivo tomó un buen tiempo. ¿Por qué el autor decidió usar un solo archivo en vez de uno para cada tamaño y así usar la macro EOF como referente de lectura? Es una muy buena pregunta. Sencillamente, el autor quería practicar la lectura de archivos. En serio.

Cuando llego el momento de probar la operación que leería y llenaría el arreglo, se encontró un problema conforme el tamaño aumentaba. Para el arreglo de más de un millón de elementos, el compilador arrojaba una advertencia por traspasar el tamaño permitido de memoria usada en el stack. Por esto, se descubrió que había una constante definida internamente en el compilador llamada STACK, la cual limitaba la cantidad de bytes disponibles en la memoria con el mismo nombre. Así que cabían dos posibilidades: o modificar esta constante en el compilador, o reservar la memoria de forma dinámica ya fuera con las funciones *malloc* o *calloc* o con declaración estática del arreglo con el especificador *static.* Se optó por el uso de las funciones, puesto que hacían más reutilizable el código.

Finalmente, ya teniendo los programas ejecutables para las pruebas, se procedió a ejecutarlos con el fin de recoger las estadísticas necesarias para un análisis comparativo de las eficiencias de cada algoritmo. Pero justo cuando se ejecutó el algoritmo burbuja con el arreglo de más de un millón de elementos, el autor recordó que no había programado un mecanismo para cancelar el algoritmo en el transcurso de una hora en caso de no haber terminado la ejecución antes. Fue demasiado tarde. Gracias al tamaño del arreglo, y a la mala eficiencia del burbuja, fueron unas cuatro horas de preocupación por saber si el burbuja se había *crasheado* en ejecución o de verdad estaba tardando tanto. Así que después de aquel percance, el autor comprobó por si mismo dos cosas: Primero, el bubblesort de verdad apesta; segundo, nunca ejecutar una prueba de eficiencia algorítmica que tenga límite temporal (en este caso era de una hora) sin asegurarse de que a la hora se termine.

Es aquí donde el autor corroboró los rumores que circulan respecto al bubblesort de primera mano, pues mientras este duró mas de cuatro horas con un arreglo de 1’ 241 184 elementos (que incluso estaba ordenado), el Quicksort, por su parte, tardó en promedio 0,7 segundos para el mismo arreglo. Sencillamente, la diferencia es abismal.