**Reflexión Unidad temática 9: Algoritmos de ordenación**

He terminado esta unidad, con una mezcla de contento, y por que no, nervios, no por la misma, sino por la cercanía de esta con el último parcial.

Hablando de la unidad en sí, ciertamente me gustó mucho. Los algoritmos de ordenación, son sin duda uno de los campos más trabajados (si no el que más) en lo que refiere a las ciencias de la computación.

Sin dudas mi favorito es HeapSort, literalmente es mejor (en teoría), que el sobrevalorado de QuickSort. Lastima que en la práctica, este último termine siendo más efectivo.

De todos modos, no se puede negar la magnificencia de los benditos “max-heaps”.

HeapSort no solo es eficiente ((n log n) en todos los casos), sino que además es in-place, lo que significa que no ocupa espacio adicional (complejidad espacial O(1)).

En adición, su funcionamiento es bastante simple, y hace muchísimo sentido. Sin dudas es necesario entender de arboles binarios para comprender HeapSort, pero, en mi caso, ya teniendo esos dichosos conocimientos, fue un trámite comprender su lógica,

Quizás una desventaja que tiene es que no es estable, es decir, que no mantiene el orden relativo de elementos con la misma clave. Pero a decir verdad, el infame de QuickSort tampoco es estable.

Insertion sort es el más elemental de todos, fácil de entender y programar; pero su aplicación se reduce a casos en donde los elementos, además de no ser numerosos, deberían estar lo más ordenados posibles, para que usar Insertion sort tenga sentido. Si se cumple esto último, nos podemos acercar a un caso promedio de casi O(n), lo cual es una joyita.

De hecho, Java, cuando la cantidad de elementos es < 47, en vez de recurrir al pérfido de QuickSort, recurre a un simple InsertionSort.

ShellSort es un intento de mejora de InsertionSort. Pero a diferencia de este último, no es estable.

Sin embargo, con una buena secuencia de incrementos (como Sedgwick), ShellSort puede tener un peor caso de O(n\*\*(4/3)), lo cual no está nada mal. Además de ser in-place.

BubbleSort no sirve para nada.

SelectionSort está bastante cerca de la inutilidad de BubbleSort, pero es, a mi gusto, el algoritmo de ordenación más fácil de entender y programar. Al igual que el burbuja, en todos los casos es O(n\*\*2), lo cual es paupérrimo. Pero al menos ambos son in-place.

Ya mencione al moroso de QuickSort. La verdad es que es de los más utilizados en la práctica, ya que, con una buena elección de pivote, su Big O es casi asegurado O(n log n).

A mi no me cae bien porque le hace sombra a “Chad HeapSort”, pero no se puede negar su importancia. Su complejidad espacial, si se elige un buen pivote, es de O(log n), lo cual es considerablemente poco,

Pasando a los tres chiflados, toca hablar de Counting Sort, Radix Sort y BucketSort.

CountingSort al principio me caía mal, porque lo vi muy rebuscado; pero después de comprender su funcionamiento, me retracto completamente. Si se asume que todos los elementos son enteros de 1…N, este algoritmo puede tener Big O(n), siempre y cuando no haya una diferencia sustancial entre los valores de las claves. Es decir, que si la clave más grande es mayor a, digamos, 20 (a papel), estamos en el horno; pues estaríamos desperdiciando mucha memoria, propio de tener que cargar un arreglo adicional muy grande. Ya que en CountingSort, necesitamos de extra overhead para su funcionamiento, y el tamaño del arreglo en cuestión es igual al valor del elemento más grande del conjunto numérico a ordenar.

Empero, su complejidad espacial es lineal.

BucketSort, también conocido como BinSort, o, por que no, UrnasSort; es un algoritmo que también depende de conocer los datos con los que vamos a trabajar. Este en particular, funciona generando cubetas, o “urnas”, en donde colocaremos los elementos del array a ordenar. Una vez tengamos los elementos dentro de los cubos, los ordenaremos por separado (usando cualquier algoritmo conocido, generalmente InsertionSort); una vez terminemos, los concatenamos, formando el arreglo completo.

Big O (n), si se dividen bien los elementos. Y O(n + k), donde n son los elementos y k las cubetas; de complejidad espacial.

Por último tenemos al rarito de RadixSort. Al principio no lo comprendía muy bien, pero ahora si. jeje.

Básicamente ordena el conjunto de elementos, procesando sus componentes individuales. Es decir, si trabajamos con un arreglo numérico con números de hasta 3 cifras; primero revisa las centenas, luego decenas y luego unidades, y las ordena en ese orden. Al final, tendremos el array ordenado. También sirve para ordenar elementos compuestos (por ejemplo fechas. EJ: 14/12/2005). En este caso, trabajamos primero con los años, luego los meses y luego los días.

Tanto su complejidad espacial como temporal son lineales.

En resumen, la verdad es que esta unidad me ha gustado muchísimo. Fue en la que mejor me fue en el iRat, y es que no es para menos, ya que para prepararme para este realice un resumen de hasta 18 hojas, “Pode crer?”.

Pese a ser la unidad más extensa del curso, yo no la note muy pesada la verdad.

**Conclusión conclusiva**: Todo joya, que tenga buen dia. Y aguante HeapSort.