INSTITUTO POLITÉCNICO NACIONAL

Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas.

MATERIA:

- Analisis y diseño de algoritmos

MAESTRA:

- Erika Sánchez Femat

ALUMNA:

- Naylin Yusseth Martínez Mireles

MÉTODO QUICKSORT

Índice

1. INTRODUCCIÓN

1.1. ¿QUÉ ES?

El método de ordenamiento QuickSort es actualmente el más eficiente y veloz de los métodos de ordenación interna. Es tambien conocido con el nombre del método rápido y de ordenamento por partición.
Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de QuickSort por la velocidad con que ordena los elementos del arreglo.
Es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar "n" elementos en un tiempo proporcional a "n Log n".
Este método fue creado por el científico británico Charles Antony Richard Hoare, tambien conocido como Tony Hoare en 1960, su algoritmo Quicksort es el algoritmo de ordenamiento más ampliamente utilizado en el mundo.

2. DESARROLLO

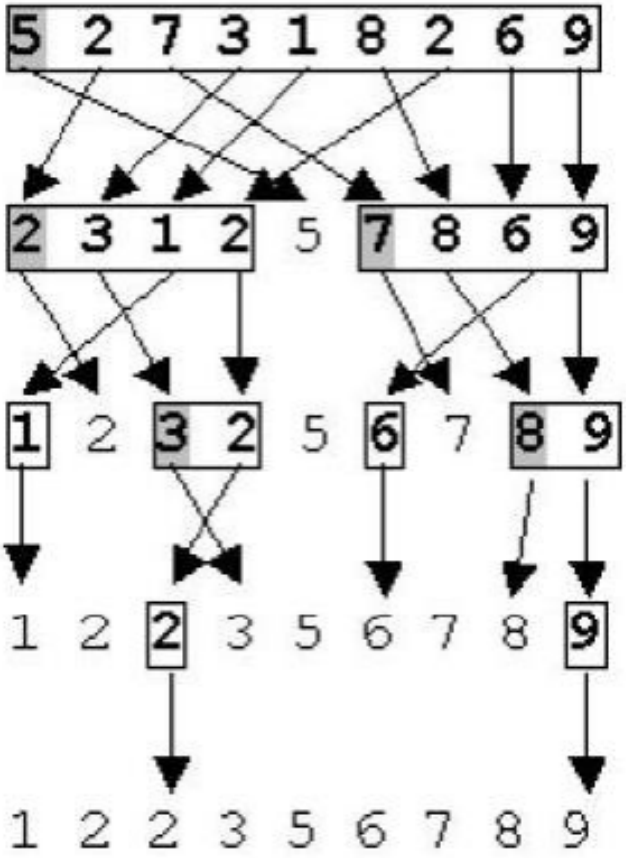
2.1. ¿CÓMO FUNCIONA?

Elegimos un elemento de la lista de elementos a ordenar, al que llamaremos "pivote".
Lo que el algoritmo desea hacer es lo siguiente:
Se toma un elemento "x" de una posición cualquiera del arreglo. Se trata de ubicar a "x" en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentran a su izquierda sean menores o iguales a "x" y todos los elementos que se encuentren a su derecha sean mayores o iguales a "x".

Se repiten los pasos anteriores pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de "x" en el arreglo.

Reubicar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.
La eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

Ejemplo 1.



2.2. COMPLEJIDAD DEL ALGORITMO

Tiene una complejidad promedio de O(n log n) y es uno de los algoritmos más utilizados para grandes volúmenes de datos.

3. ANALISIS DE CASOS

3.1. Mejor caso

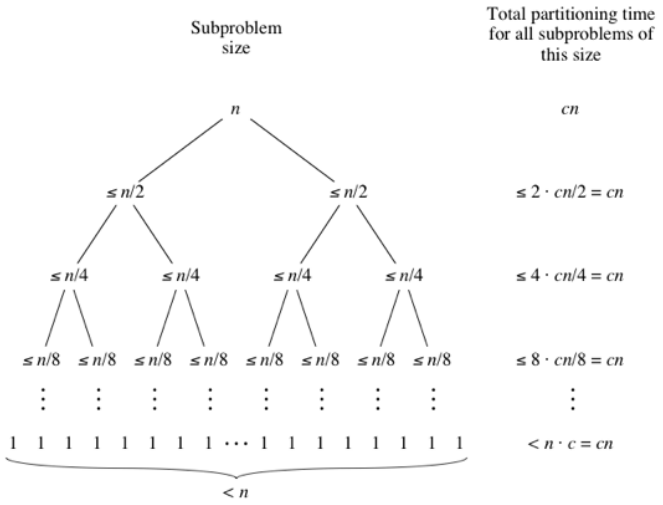
El mejor caso del ordenamiento rápido ocurre cuando las particiones están balanceadas tan uniformemente como sea posible: sus tamaños son iguales o difieren en 1.

El primer caso ocurre si el subarreglo tiene un número impar de elementos, el pivote está justo en medio después de hacer la partición y cada partición tiene $\frac{(n-1)}{2}$ elementos.

El otro caso ocurre si el subarreglo tiene un número par n de elementos y una partición tiene $\frac{n}{2}$ elementos mientras que la otra tiene $\frac{n}{2-1}$.

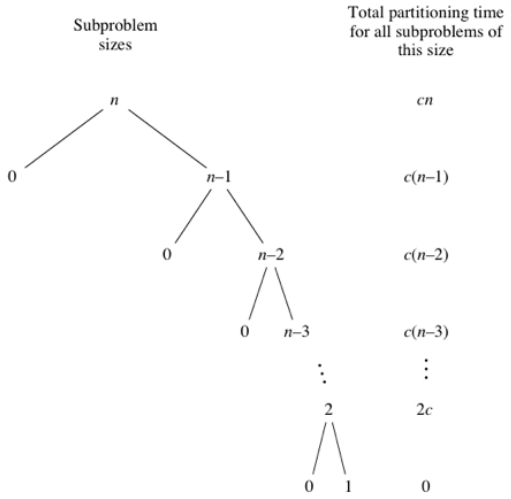
En cualquiera de estos casos, cada partición tiene a lo más $\frac{n}{2}$ elementos, y el árbol de los tamaños de los subproblemas se parece mucho al árbol de los tamaños de los subproblemas para el ordenamiento por mezcla, con los tiempos para hacer las particiones que se ven como los tiempos de mezcla:

Al usar notación O grande, obtenemos el mismo resultado que para un ordenamiento por mezcla: $= O(n \log_2 n)$.



3.2. Peor caso

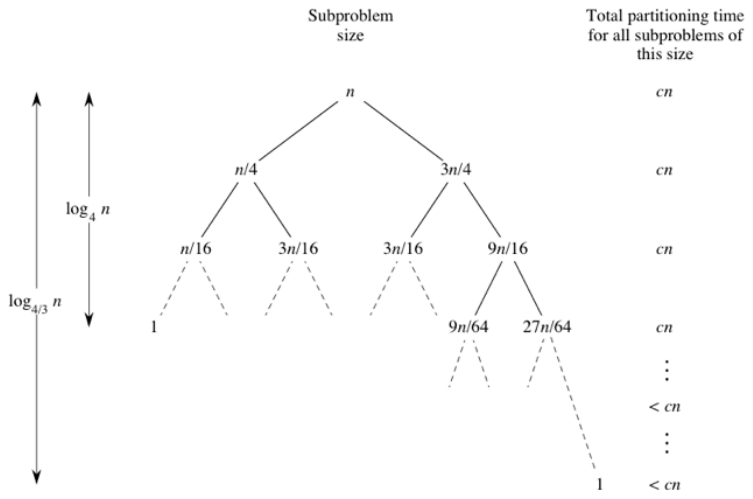
Cuando el ordenamiento rápido siempre tiene las particiones más desbalanceadas posibles, entonces la llamada original tarda un tiempo çn"para alguna constante c, la llamada recursiva sobre n-1 elementos tarda un tiempo c(n-1), la llamada recursiva sobre n-2 elementos tarda un tiempo c(n-2), y así sucesivamente.



3.3. Caso promedio

Mostrar que el tiempo de ejecución del caso promedio también es $=O(n \log_2 n)$ requiere unas matemáticas bastante complicadas, así que no lo vamos a hacer. Pero podemos obtener un poco de intuición al ver un par de otros casos para entender por qué podría ser $O(n \log_2 n)$.
Una vez que tengamos $O(n \log_2 n)$, la cota de $O(n \log_2 n)$ se sigue porque el tiempo de ejecución del caso promedio no puede ser mejor que el tiempo de ejecución del mejor caso.
Primero, imaginemos que no siempre obtenemos particiones igualmente balanceadas, pero que siempre obtenemos una razón de 3 a 1
Es decir, imagina que cada vez que hacemos una partición, un lado obtiene $\frac{3n}{4}$ elementos y el otro lado obtiene $\frac{n}{4}$ para mantener limpias las matemáticas, no vamos a preocuparnos por el pivote.

Entonces, el árbol de los tamaños de los sub-problemas y los tiempos para hacer las particiones se vería así:



Para analizar el caso promedio, sea $T(n)$ que denota el número de pasos necesarios para llevar a cabo el quick sort en el caso promedio para n elementos. Se supondrá que después de la operación de división la lista se ha dividido en dos sublistas. La primera de ellas contiene s elementos y la segunda contiene $(n-s)$ elementos. El valor de s varía desde 1 hasta n y es necesario tomar en consideración todos los casos posibles a fin de obtener el desempeño del caso promedio.
Para obtener $T(n)$ es posible aplicar la siguiente fórmula:

$T(n) = Promedio(T(s) + T(n-s)) + cnconl <= s <= n$

donde cn denota el número de operaciones necesario para efectuar la primera operación de división. Cada elemento es analizado antes de dividir en dos sublistas la lista original. Al resolver matemáticamente, se obtiene una eficiencia $O(nlogn)$.

4. CONCLUSIÓN

El algoritmo de ordenamiento rápido permite ordenar una lista en menor tiempo promedio en comparación a otros algoritmos como bubblesort, sin embargo, en el peor caso la ganancia es el mismo. Así mismo, es necesario considerar la ubicación del pivote en cada iteración, ya que éste determina los elementos menores a la izquierda, y los valores mayores a la derecha, con el objetivo de reducir la complejidad del algoritmo al usar la técnica de divide y vencerás.
Es un algoritmo muy eficiente que nos ayuda a poder ordenar de manera más rapida que otros metodos.

5. REFERENCIAS

https://bitcu.co/algoritmo-de-ordenamiento-quicksort/
https://bitcu.co/algoritmo-de-ordenamiento-quicksort/
https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-4.pdf
http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro9/mtodo_quick_sort.html