INSTITUTO POLITÉCNICO NACIONAL

Unidad Profesional Interdisciplinaria de Ingeniería
Campus Zacatecas.

Materia:
Análisis y Diseño de Algoritmos

Práctica 01: Análisis de Casos

Docente:
Erika Sánchez Femat

Nombre del alumno:
Dalia Naomi García Macías

Fecha de entrega:
12 de Octubre de 2023

Índice

1. Introducción

Desde que existe la ciencia de la computación, uno de los mayores problemas con los que los ingenieros se encontraban en su día a día, era el de ordenar listas de elementos. Por su causa, diversos algoritmos de ordenación fueron desarrollados a lo largo de los años y siempre existió un intenso debate entre los desarrolladores sobre cual de todos los algoritmos de ordenación era el más rápido.

El debate finalizó abruptamente en 1960 cuando Sir Charles Antony Richard Hoare, nativo de Sri Lanka y ganador del premio Turing en 1980, desarrolló el algoritmo de ordenación QuickSort casi por casualidad mientras ideaba la forma de facilitar la búsqueda de palabras en el diccionario.

El algoritmo QuickSort, en esapñol método de ordenamiento rápido, será el que se analizará durante la práctica, basa en la técnica de "divide y vencerás" por la que en cada recursión, el problema se divide en subproblemas de menor tamaño y se resuelven por separado (aplicando la misma técnica) para ser unidos de nuevo una vez resueltos.

2. Desarrollo de la Práctica

2.1. ¿Qué es?

Quicksort es un algoritmo de ordenación que utiliza el primcipio de divide y vencerás, fue desarrollado por Antony R. Horae en 1959.

El método de ordenamiento QuickSort es actualmente el más eficiente y veloz de los métodos de ordenación interna.

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".

2.1.1. Caracteristicas

En la práctica, es el algoritmo de ordenación más rápido conocido, su tiempo de ejecución promedio es O(n log (n)), siendo en el peor de los casos O(n2), caso altamente improbable. El hecho de que sea más rápido que otros algoritmos de ordenación con tiempo promedio de O(n log (n)) (como SmoothSort o HeapSort) viene dado por que QuickSort realiza menos operaciones ya que el método utilizado es el de partición.

2.2. Idea central del algoritmo

La idea central de este algoritmo consiste en 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.

2.3. ¿Cómo funciona? (Explicación mas detallada)

El proceso es el siguiente:

1. Del vector de entrada se elige uno de sus elementos (eje) que dividirá al vector en dos sub-vectores, uno de ellos con elementos menores que el eje y otra con elementos mayores o iguales.
2. Después de elegir el eje se ordenan todos los elementos alrededor del mismo, por un lado los de valor más pequeño antes que él y los mayores detrás de él.
3. Los pasos anteriores se aplican a cada uno de los sub-vectores de forma recursiva.
4. Todos los sub-vectores se combinan para formar el vector ordenado.

El eslabón más débil del algoritmo es la elección del eje. ¿Cómo elegimos el mejor eje?, difícil en un vector desordenado. El elemento de valor intermedio sería el mejor, pero encontrarlo es costoso, entonces debemos tomar alguna de estas opciones:

- El primer elemento del vector
- El último elemento
- La mediana
- Cualquier otro elemento aleatorio.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

Ejemplo:

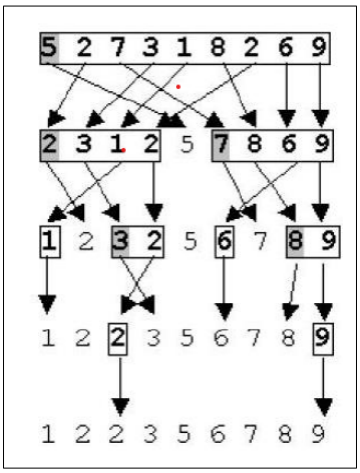

Figura 1: Ejemplo del funcionamiento del algoritmo

2.4. Complejidad

La complejidad del algoritmo Quicksort puede variar según varios factores, y algunos de los criterios que pueden influir en su complejidad incluyen:

- **Selección del pivote:** La elección del pivote es un factor crítico en el rendimiento de Quicksort. Siempre es mejor elegir un pivote que divida aproximadamente la lista en dos partes iguales. En el peor caso, si el pivote es el elemento más pequeño o más grande en cada partición, el rendimiento se degrada y se acerca a $O(n^2)$. Sin embargo, si el pivote se elige de manera eficiente, el rendimiento es $O(n\ log\ n)$.
- **Distribución inicial de los elementos:** La distribución de los elementos en la lista original también puede influir en la complejidad. Si los datos ya están ordenados o casi ordenados, Quicksort puede ser menos eficiente y acercarse al peor caso. Por otro lado, si los datos están distribuidos al azar, es más probable que el rendimiento se acerque a $O(n\ log\ n)$.
- **Implementación específica:** La implementación exacta del algoritmo Quicksort puede variar. Hay diferentes formas de implementarlo, y algunos detalles de la implementación, como la elección del pivote y la estrategia para manejar elementos iguales al pivote, pueden afectar el rendimiento.
- **Tamaño de la lista:** La complejidad del algoritmo Quicksort puede variar según el tamaño de la lista a ordenar. Para listas pequeñas, puede superar a algoritmos más eficientes, como el algoritmo Merge Sort. Sin embargo, para listas extremadamente grandes, el rendimiento puede verse afectado por la necesidad de memoria y la caché del sistema.

2.4.1. Mejor caso

El mejor caso de quick sort ocurre cuando X divide la lista justo en el centro. Es decir, X produce dos sublistas que contienen el mismo número de elementos. En este caso, la primera ronda requiere n pasos para dividir la lista original en dos listas. Para la ronda siguiente, para cada sublista, de nuevo se necesitan $\frac{n}{2}$ pasos (ignorando el elemento usado para la división). En consecuencia, para la segunda

ronda nuevamente se requieren $2\frac{n}{2} = n$ pasos. Si se supone que $n = 2^P$, entonces en total se requieren $p(n)$ pasos. Sin embargo, $p = log_2\ n$. Así, para el mejor caso, la complejidad temporal del quick sort es $O(n\ log\ n)$.

2.4.2. Caso promedio

El peor caso del quick sort ocurre cuando los datos de entrada están ya ordenados o inversamente ordenados. En estos casos, todo el tiempo simplemente se está seleccionando el extremo (ya sea el mayor o el menor). Por lo tanto, el número total de pasos que se requiere en el quick sort para el peor caso es: $n + (n-1) + ... + 1 = \frac{n}{2}(n+1) = O(n^2)$.

2.4.3. Peor caso

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 5 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:

$T(n) = Promedio(T(s) + T(n-s)) + cn\ con\ l <= s <= n$

Donde cn equivale al 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 $O(n\ log\ n)$.

3. Conclusiones

Gracias a la siguiente prática, se analizó de manera teorica el algoritmo de Quicsort, el más eficaz y eficiente en cuanto a lagoritmos de ordenamiento. Y podremos tomar la desicion correcta caundo deseemos usar uno.

4. Referencias

- https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-4.pdf
- https://numerentur.org/quicksort/
- https://www.genbeta.com/desarrollo/implementando-el-algoritmo-quicksort
- http://aniei.org.mx/paginas/uam/CursoAA/curso_aa_19.html#:~:text=El%20mejor%20caso%20de%20quick,lista%20original%20en%20dos%20listas.