Taller método maestro

Parte 1: Resolución Matemática.

1) $T(n) = 4 \cdot T\left(\frac{n}{2}\right) + O(n^2)$

Identificar: $a = 4$, $b = 2$, $f(n) = O(n^2)$, $d = 2$

Comparar $f(n)$ con $n^{\log_b a}$: $\log_2 4 = 2$ entonces $f(n) = n^{\log_b a}$ $(n^2 = n^2)$

Complejidad = $O(n^2 \log n)$

2) $T(n) = 3 \cdot T\left(\frac{n}{3}\right) + O(n)$

Identificar: $a = 3$, $b = 3$, $f(n) = O(n)$, $d = 1$

Comparar: $\log_3 3 = 1$ entonces $f(n) = n^{\log_b a}$ $(n = n)$

Complejidad = $O(n \log n)$ "Caso 2 porque son iguales"

3) $T(n) = 5T\left(\frac{n}{2}\right) + O(n \log n)$

Identificar: $a = 5$, $b = 2$, $f(n) = O(n \log n)$, $d = 1$

Comparar: $\log_2 5 = 2,3219\ldots$ como es menor que $O(n \log n)$ entonces $n^{\log_2 5}$.

Complejidad: $O(n^{\log_2 5})$

Parte 2: implementación Algorítmica.

1) Merge_sord:

$T(n) = 2T(n/2) + O(n)$

identificar: $a=2, b=2, f(n) = O(n), d=1$

Comparar: $\log_b a = \log_2 2 = 1$ / entonces $f(n) \circ n^d = n^{\log_b a}$ "$n=n$"

Complejidad: $O(n \log n)$ "caso 2"

2) Busqueda binaria

$T(n) = T(n/2) + O(1)$

identificar: $a=1, b=2, f(n) = O(1), d=0$

Comparar: $\log_b a = \log_2 1 = 0$ / $f(n) \circ n^d = n^{\log_b a}$ "$1=1$"

Complejidad: $O(\log n)$

3) Quick sord "mejor caso"

$T(n) = 2T(n/2) + O(n)$

identificar: $a=2, b=2, f(n) = O(n), d=1$

Comparar: $\log_b a = \log_2 2 = 1$ / $f(n) \circ n^d = n^{\log_b a}$ "$n=n$"

Complejidad: $O(n \log n)$

En el peor caso Quick sort tiene una complejidad de $O(n^2)$.
En este caso, el pivote o punto medio no divide uniformemente la lista. Es decir un lado de la lista tendrá un tamaño de $n-1$ y el otro 1.
La recurrencia es: $T(n) = T(n-1) + O(n)$
Esto solo se puede solucionar con método de expansión.

Parte 3: ~~Complejidad~~ Explicación y Solución Extendida en PDF.

Analisis de complejidad con Metodo maestro.

Solución Matematica

$$T(n) = 2T(n/2) + O(n)$$

$a = 2,\ b = 2,\ f(n) = O(n),\ d = 1$

Comprobacion: $\log_b a = \log_2 2 = 1 \;/\; f(n) = n^{\log_b a}$ "$n = n$"

Complejidad: $O(b \log n)$

Con la solucion de este problema, se logra identificar que dependiendo el tamaño de la entrada "n", la complejidad y el tipo de busqueda el algoritmo aumentara o disminuira su tiempo de ejecución.

Usando busqueda binaria el tiempo de ejecucion aumenta segun sea el tamaño de la entrada "n" en este caso el numero de registros, dicho tiempo aumenta a medida que aumentan los niveles y su costo.

Parte 3:

Análisis de Complejidad con el Método Maestro

Teniendo en cuenta los requerimientos del problema en la **parte 4** se requiere diseñar un algoritmo que realice una búsqueda en la tabla **personas** para encontrar a todas las personas que cumplan con una condición específica, como por ejemplo:

- Buscar todas las personas cuyo nombre comience con una letra específica.
- Buscar todas las personas nacidas antes de una fecha específica.

¿Cuál sería el tiempo de ejecución de este algoritmo, dependiendo de cómo se realice la búsqueda en la base de datos?

Según lo anterior el planteamiento o diseño del algoritmo seria:

Si la búsqueda se hace mediante **búsqueda lineal**, se sabe que el tiempo de ejecución del algoritmo será lineal en base a la entrada o mejor dicho n, es decir en este caso son n datos o registros, el algoritmo recorrerá cada uno de esos por ende tendrá un costo lineal de O(n), que quiere decir que crece a medida que la entrada también lo hace.

Si la búsqueda se hace mediante **divide y vencerás**, se sabe que se deberá dividir la tabla o el registro a la mitad, dando como resultado 2 tablas o registros separados, en los cuales se hará dicha búsqueda recursivamente para al final combinar los resultados ordenados, esto tendría un costo de **O(n).**

Dicho esto ya se puede formular la recurrencia ya que se tiene:

Subproblemas: **2T(n/2)**

Costo: **O(n)**

Resultado de recurrencia: **2T(n/2)+O(n)**

Para saber el tiempo de ejecución, se debe utilizar el método maestro para saber la complejidad.

Según el método maestro **aT(n/b)+f(n): a = 2, b = 2, d = 1, f(n) = O(n)**

Se calcula con **n^log_b (a): n^log_2 (2) = n^1= n**

Se compara con **f(n)** o **n^d** con **n^log_b (a): n = n,** eso demuestra una complejidad de caso 2 **O (n log n)**

Esto determina que con divide y vencerás el tiempo de ejecución crece más rápido que **O(n)** pero menos que **O(n^2),** es decir crece en niveles dependiendo de la entrada.

Para finalizar, se logra identificar que dependiendo la entrada y la complejidad, el tiempo de ejecución aumentara dependiendo de la entrada, es decir que si se realiza una búsqueda en n registros, dependiendo la cantidad de estos registros y la complejidad del algoritmo, el tiempo de ejecución aumentara, ya que entre más aumente la complejidad más operaciones o niveles se deben hacer.