Faber Alexis Solis Gamboa - 2259714
Jhojan Serna Henao - 2259504
George Chamorro Patiño - 2259521
Tecnología en Desarrollo de Software - 2724

Taller 3

Análisis y desarrollo de Algoritmos 1

Punto 1: Análisis de algoritmo divide y vencerás:(Stooge Sort)

- Explicación:
  - Divide: En este algoritmo dividimos el tamaño del arreglo entre 3, para obtener los primeros ⅔ iniciales y los ⅔ finales
  - Conquistar: Recursivamente vamos dividiendo los arreglos resultantes hasta llegar a arreglos de tamaño 2 o menos, la recursividad se realiza por medio de los primeros ⅔ , luego por otra recursividad de los últimos ⅔. Combinar: Al llevar a cabo los llamados recursivos se verifica con otro llamado recursivo que el los primeros ⅔ se hayan ordenado.
- Complejidad:
  - Teórica:

    Sea:

$$T(n) = 3 * T(\frac{n}{3/2}) + \mathcal{O}(n^0) \in \mathcal{O}(n^{\log_{3/2} 3})$$

    Donde por el método del maestro cumple en el caso1:

    $T(n) = \Theta(n^{\log_{3/2} 3})$

    $f(n) = n^0 = 1$

    Si 1 es $\mathcal{O}(n^{\log_{3/2} 3-\epsilon})$ donde $\epsilon > 0$

    $\mathcal{O}(n^{\log_{3/2} 3-\epsilon})$

    $\mathcal{O}(n^{2.709 - \epsilon})$ puede existir un valor de ϵ donde cumpla que $n^0=1$

    Solución → $T(n) =\Theta(n^{\log_{3/2} 3})$

Entrada (n)	Complejidad $\mathcal{O}(n^{\log_{3/2} 3})$
10	512.28
100	262435.5003
1000	134441662.4

Faber Alexis Solis Gamboa - 2259714
Jhojan Serna Henao - 2259504
George Chamorro Patiño - 2259521
Tecnología en Desarrollo de Software - 2724

10000	$6.88723918x10^{10}$

- Práctica:

Entrada (n)	Tiempo de ejecución
10	0.000000 segundos
100	0.000000 segundos
1000	0.000994 segundos
10000	0.002007 segundos

Podemos observar que la complejidad nos da una curva de crecimiento de costos mucho más alta que el tiempo de ejecución encontrado, pero también se observa el patrón de crecimiento de ambos, a mayor cantidad de datos de entrada los costos se incrementarán.

Con este algoritmo (Stooge Sort) nos dimos cuenta de que la complejidad de este es demasiado alta, aunque pudimos optimizarlo al principio el rendimiento que se observaba era el mismo esperado por la complejidad teórica donde en diferentes ocasiones lanzaba excepciones de runtime.

Punto 2: Diseño algoritmo Divide y vencerás: (Quick Sort)

- Explicación:
  - Dividir (Divide): La función quicksort toma un arreglo (arr) y elige un elemento como pivote.Luego, divide el arreglo en tres partes: elementos menores que el pivote, elementos iguales al pivote y elementos mayores que el pivote.
  - Conquistar (Conquer): Se aplica recursivamente el algoritmo quicksort a las sublistas izquierda y derecha (los elementos menores y mayores que el pivote, respectivamente). En este paso, se resuelven los subproblemas más pequeños.
  - Combinar (Combine): La combinación ocurre en la declaración return quicksort(left) + middle + quicksort(right), donde las sub listas ordenadas se combinan para formar el arreglo final ordenado.
- Complejidad:
  - Teórica:

Faber Alexis Solis Gamboa - 2259714
Jhojan Serna Henao - 2259504
George Chamorro Patiño - 2259521
Tecnología en Desarrollo de Software - 2724

$T(n) = 2T\left(\frac{n}{2}\right) + n \qquad n^{\log_b a} \rightarrow n^1$

$f(n) = n \qquad 2)\ n\ es\ \Theta(n)$ ✓

$T(n) = \Theta(n^{\log_b a} \log(n)) = \Theta(n \log(n))$

Entrada (n)	Complejidad
10	10
100	200
1000	3000
10000	40000

- Práctica:

Entrada (n)	Tiempo de ejecución
10	0.0 segundos
100	0.0 segundos
1000	0.0019567012786865234 segundos
10000	0.008516550064086914 segundos

El tiempo de ejecución es fiel de acuerdo al rendimiento mostrado en la complejidad teórica, ambos casos muestran un rendimiento muy óptimo. También se muestra la relación de crecimiento de ambos, a mayor cantidad de datos de entrada mayor será el costo,

Este algoritmo resultó completamente eficiente en cuanto al rendimiento, no sólo ordena una lista de datos por medio de la estrategia divide y vencerás, sino que además se le

Faber Alexis Solis Gamboa - 2259714
Jhojan Serna Henao - 2259504
George Chamorro Patiño - 2259521
Tecnología en Desarrollo de Software - 2724

implementó la capacidad de hallar la moda y todo esto lo logra hacer en tiempos muy buenos para cantidades de datos muy altas.

Punto 3: Comparación ejecución algoritmos:

- QuickSort: O(n \log n)\ en promedio.
- Insertion Sort: O(n^2)\ en el peor de los casos.
- Merge Sort: O(n \log n)\.

- **QuickSort:**

Entrada (n)	Tiempo Real (seg.)	Complejidad (O(n log(n)))	Constantes
10	0.0	10	0
50	0.0	84.9485	0
100	0.0	200	0
500	0.002506	1349.485	1,857*10^-6
1000	0.005518	3000	1,83933*10^-6
2000	0.008514	6602.05999	1,2896*10^-6
5000	0.019133	18494.85002	1,0345*10^-6
10000	0.034761	40000	0,00000086

Constante **= 6,2929E-07**

- **Insertion Sort:**

Entrada (n)	Tiempo Real (seg.)	Complejidad (O(n^2))	Constantes
10	0.0	100	0
50	0.0	2500	0
100	0.000999	10000	0,000000099
500	0.019624	250000	0,000000078
1000	0.072312	1000000	0,000000072

Faber Alexis Solis Gamboa - 2259714
Jhojan Serna Henao - 2259504
George Chamorro Patiño - 2259521
Tecnología en Desarrollo de Software - 2724

2000	0.273253	4000000	0,000000068
5000	1.712527	25000000	0,000000068
10000	6.934455	100000000	0,000000069

Constante **= 5,71084E-08**

- **Merge Sort:**

Entrada (n)	Tiempo Real (seg.)	Complejidad (O(n log(n)))	Constantes
10	0.0	10	0
50	0.0	84.9485	0
100	0.0	200	0
500	0.001998	1349.485	1,48056*10^-6
1000	0.003008	3000	1,00267*10^-6
2000	0.007002	6602.05999	1,06058*10^-6
5000	0.016278	18494.85002	8,80137*10^-7
10000	0.035189	40000	0,00000087

Constante **= 6,62959E-07**

Observando las diferentes constantes de las implementaciones de cada algoritmo se observa que el más eficiente fue el InsertionSort, mientras que el resto de algoritmos tuvieron un rendimiento muy similar, Curiosamente el algoritmo InsertionSort tiene un crecimiento cuadrático y los otros un crecimiento linealítmico, y según la notación Big-O la curva de crecimiento del InsertionSort en algún punto para la entrada de datos los costos serán mayores en comparación a MergeSort Y QuickSort.

Al llevar a cabo este análisis entre lo teórico y lo práctico, pudimos identificar más fácilmente los patrones de crecimiento en función de la entrada de datos e implementación del código, lo que permitió ser más precisos con los resultados, pero la elección del algoritmo depende

de muchos factores, y los resultados pueden variar según la aplicación específica y las características del conjunto de datos.