Taller #3

Santiago Aníbal Carrillo Torres (2259465)

Edwar Yamir Forero (2259664)

Juan Eduardo Calderon Jaramillo (2259671)

ANÁLISIS Y DISEÑO DE ALGORITMOS I

CARLOS ANDRES DELGADO SAAVEDRA

UNIVERSIDAD DEL VALLE

TULUÁ, VALLE DEL CAUCA

DICIEMBRE 2023

Punto 1: Análisis algoritmo Divide y Vencerás (StoogeSort).

La estrategia de divide y vencerás para el algoritmo Stooge Sort se encargará de dividir un array en 3 partes, donde cada una de ellas se entregará ordenada de menor a mayor para que luego sea combinada y se obtenga la matriz ordenada.

En este caso, cuando se **divide** en el algoritmo Stooge Sort se divide la matriz en 3 partes, donde la primera divide los primeros ⅔ del arreglo, la segunda divide los últimos ⅔ del arreglo y por último la tercera división que de nuevo divide los primeros ⅔ del arreglo.

Vencer y combinar se refiere al momento donde las dos primeras divisiones del array principal se ordenan de forma ascendente. Esto ocurre en la validación (arreglo [i] > arreglo [j]) lo que quiere decir que si hay un valor mayor que está tras uno de menor valor, se realizará un intercambio de las posiciones de los dos valores en los cuales la condición fue verdadera.

La tercera y última división, se refiere al **vencer** ya que es la encargada de verificar o validar que los datos del primer tercio se encuentren ordenados y por lo tanto los del segundo tercio también lo estarán.

Cálculo Complejidad:

Complejidad teórica O(n^2):

Análisis: Correspondiente a la práctica, el algoritmo por sí mantendrá su complejidad, pero esta misma será un factor negativo al momento de realizar operaciones a gran escala como lo puede ser al momento de ingresar diversos arreglos en un mismo tiempo, ya que su eficiencia decaerá mientras los arreglos sean de mayor tamaño.

Análisis y conclusiones:

Santiago: En lo que se visualizó al momento de realizar en el ordenador la prueba, Stooge Sort logra ordenar arreglos hasta de tamaño 1000 de manera aceptable, pero al momento de resolver un arreglo tamaño 10000 pasados 15 minutos aún no se lograba completar. Este tiempo puede llegar a cambiar dependiendo de la manera en como pueda llegar a estar organizado el arreglo aleatorio, ya que en una siguiente prueba realizada, al cabo de 2 minutos todos los arreglos ya se habían ordenado de manera exitosa.

Edwar: Al ejecutar el código de Stooge Sort con los cuatro diferentes tamaños (10,100,1000,10000), se puede evidenciar que al organizar los tres primeros tamaños de matrices da como resultado un nivel de tiempo aceptable, pero al llevar a cabo la ejecución de la matriz de tamaño 10000, la organizacion del datos de la matriz es muy desfavorable en cuanto al tiempo y esto da como conclusión, que la

implementación de este algoritmo es óptima para arreglos menores de 1000 datos, aunque también es importante aclarar que el tiempo se ve influenciado por la organización de la matriz, es decir depende de si la matriz está en su peor, intermedio o mejor estado en cuanto organización de sus datos.

En la siguiente tabla se puede evidenciar el tiempo de ejecución:

Stooge-Sort			
	Tiempo en Milisegundos		
Tamaño Matriz	Prueba 1	Prueba 2	
10	0,0164292	0,0001588	
100	0,0020491	0,0018104	
1000	0,0070245	0,0030207	
10000	0,0130614	0,0127806	Promedio
total	0,0385642	0,0177705	0,02816735

Punto 2: Diseño de algoritmo Divide y Vencerás.

Se usa la técnica vista en el punto 3, llamada quicksort para lograr primero ordenar los elementos del menor al mayor, para que de esta forma no se necesite recorrer la toda la matriz para cada elemento del array, provocando que su complejidad sea O(n^2).

En cuánto a **dividir** el método “quicksort” se encarga de elegir un pivote dentro del array enviado, el cual será el eje de los cambios que se realizarán, donde menores se ubican a la izquierda y mayores a la derecha. Luego de hacer estos cambios se **combina**

Para el siguiente paso, se inicia el conteo de cada uno de los elementos en “encontrarModa”, que usará el array ordenado de menor a mayor, para contar cada elemento, mientras que el anterior sea igual a este, una vez se realice la comparación, la función se encargará de comparar si la frecuencia actual es mayor que la anterior, para que así el elemento al cual pertenece su frecuencia sea agregado al List que devolverá la moda.

Cálculo Complejidad:

La complejidad de este algoritmo sería O(n*log(n) + m*log(m)) donde (m<=n) n es el tamaño total de la matriz dada y m es el número de elementos únicos que posee dicha matriz. Dentro de la función “encontrarModa” la complejidad sería de O(n+m) n siendo la iteración del array y m siendo la iteración HasMap. Mientras que log (n) y log(m) se halla dentro de la función “modaDivideYVencerás” la cual se encarga de dividir en mitades cada arreglo que se ingresa durante la recursividad.

Análisis y conclusiones:

Santiago: Para la ejecución de este algoritmo, su complejidad es de como promedio O(n*log(n)) esto se logra gracias a que la técnica QuickSort, ordena los datos de manera ascendente, lo que desencadena en una mejor eficiencia al momento de realizar cada uno de los conteos de cada elemento. Es decir, que para contar cada uno de los elementos del array, se verifica que el dato actual sea igual al siguiente, lo que inhibe al problema a comparación de otro tipo de soluciones, recorrer el array completo por cada elemento distinto para hallar su frecuencia. En cuánto al tiempo de ejecución se tuvo un promedio de 0,033 segundos, donde se puede resaltar que la primer ejecución, donde existía un tamaño de matriz menor, la mayoría de veces tardó más que las matrices de tamaño 100 en adelante

Edwar: En este algoritmo se implementa el método de ordenamiento QuickSort para organizar un arreglo de enteros y luego determinar la moda de dicho arreglo ordenado. QuickSort, en términos teóricos, tiene una complejidad de tiempo de O(n^2) en el peor caso y O(nlogn) en el caso promedio y mejor caso. La elección del pivote y la recursión en QuickSort también influyen en su eficiencia práctica,

siendo más eficiente en arreglos parcialmente ordenados o de pequeño tamaño. La operación de encontrar la moda, realizada después del ordenamiento, tiene una complejidad lineal O(n). En la ejecucion, la eficiencia puede variar según la distribución de los datos y otros factores. Es crucial considerar tanto la complejidad teórica como la práctica para evaluar el rendimiento del algoritmo en diferentes escenarios, destacando la importancia de la elección del pivote en QuickSort y su impacto en la eficiencia general del algoritmo.

Tabla de Pruebas:

Algoritmo Moda - Divide y vencerás				
	Tiempo (Segundos)			
Tamaño Matriz	Prueba 1	Prueba 2	Prueba 3	Promedio
10	0,012342301	0,0238633	0,0003425	
100	0,0025651	0,002635	0,002635	
1000	0,007685201	0,0108858	0,0037765	
10000	0,006675099	0,0130527	0,0134015	
Total	0,029267701	0,0504368	0,0201555	0,033286667

Punto 3: Comparación ejecución algoritmos

Quicksort

QuickSort			
Entrada (n)	Tiempo Real (seg)	Complejidad O(n log(n))	Constates
10	0,0000047	10	0,000000469900000000000
10	0,0000035	10	0,000000350000000000000
10	0,0000039	10	0,000000390000000000000
50	0,0000076	84,94850022	0,000000089465970330 3024
50	0,0000135	84,94850022	0,000000158919815718 3000
50	0,0000171	84,94850022	0,000000201298433243 1800
100	0,0000518	200	0,000000259000000000000
100	0,0000244	200	0,000000122000000000000
100	0,0000209	200	0,000000104500000000000
500	0,0001168	1349,485002	0,000000086551536187 7720
500	0,0000579	1349,485002	0,000000042905256380 7534
500	0,0000503	1349,485002	0,000000037273478341 1381
1000	0,0000967	3000	0,000000032233000000000
1000	0,0001057	3000	0,000000035233333333333
1000	0,0001011	3000	0,000000033700000000000
2000	0,0002116	6602,059991	0,000000032050602429 8392
2000	0,0003123	6602,059991	0,000000047303417480 3345
2000	0,0003425	6602,059991	0,000000051877747316 7293
5000	0,0003687	18494,85002	0,000000019935279257 0797
5000	0,0006084	18494,85002	0,000000032895643883 9363
5000	0,0006713	18494,85002	0,000000036296590629 9909
10000	0,0008091	40000	0,000000020227500000000
10000	0,0013416	40000	0,000000033540000000000
10000	0,0014735	40000	0,000000036837500000000
		Total	0,0000001134977127

Este algoritmo, teóricamente tiene complejidad O(n * log(n)), para este caso, de las 24 pruebas que se realizaron, se obtuvo que el algoritmo crece con una constancia promedio de 0.000000113 lo que muestra que para este tipo de casos su funcionamiento es bastante bueno, ya que sus valores pocas veces llegan al límite de su complejidad. Esta puede llegar a ser la mejor opción para este tipo de casos comparado a las demás, ya que la tardanza en segundos es la más baja, lo que implica que al momento de que exista un gran flujo de datos, QuickSort obtendrá un mejor rendimiento. En conclusión, esta es la mejor alternativa, en cuánto a ordenamiento se refiere, ya que su promedio de constante es el segundo más bajo y sus tiempos de ejecución son los mejores.

Insertion-Sort			
Entrada (n)	Tiempo Real (seg)	Complejidad O(n^2)	Constates
10	0,000004699	100	0,00000004699000000
10	0,000001701	100	0,00000001701000000
10	0,000001600	100	0,00000001600000000
50	0,000021299	2500	0,00000000851960000
50	0,000019900	2500	0,00000000796000000
50	0,000012400	2500	0,00000000496000000
100	0,000078500	10000	0,00000000785000000
100	0,000048200	10000	0,00000000482000000
100	0,000043699	10000	0,00000000436990000
500	0,004758500	250000	0,00000001903400000
500	0,000447299	250000	0,00000000178919600
500	0,000437200	250000	0,00000000174880000
1000	0,000460600	1000000	0,00000000046060000
1000	0,001855900	1000000	0,00000000185590000
1000	0,000253300	1000000	0,00000000025330000
2000	0,000401200	4000000	0,00000000010030000
2000	0,000402001	4000000	0,00000000010050025
2000	0,000387700	4000000	0,00000000009692500
5000	0,003888300	25000000	0,00000000015553200
5000	0,001918200	25000000	0,00000000007672800
5000	0,001808399	25000000	0,00000000007233596
10000	0,012610399	100000000	0,00000000012610399
10000	0,007735899	100000000	0,00000000007735899
10000	0,007108000	100000000	0,00000000007108000
		Total	0,000000006020756675

En cuánto al InsertionSort se tuvo un promedio de constante (0.0000000060) más bajo que los demás, pero en la práctica se mostró como el tiempo de ejecución es mucho mayor al de otros algoritmos. Su crecimiento fue de (n^2) y algunas veces hasta menor que (n) y comparado a los demás tal vez crece de menor forma. Pero el valor promedio de constante, también puede ser algo relativo, ya que es baja comparada a la demás, pero los tiempos que marca en segundos son mucho mayores, lo que significa que mientras más datos existan más demora y por lo tanto puede llegar a causar problemas durante su funcionamiento. En conclusión, este algoritmo no sería factible en caso de que existiera un gran tráfico de datos.

Merge-Sort			
Entrada (n)	Tiempo Real (seg)	Complejidad O(n log(n))	Constates
10	0,0000060	10	0,000000600000000000000
10	0,0000062	10	0,000000620000000000000
10	0,0000059	10	0,000000590000000000000
50	0,0000450	84,94850022	0,000000529732719061001
50	0,0000274	84,94850022	0,000000322548366717143
50	0,0000295	84,94850022	0,000000347269226939990
100	0,0000923	200	0,000000461500000000000
100	0,0000515	200	0,000000257500000000000
100	0,0000386	200	0,000000193000000000000
500	0,0000902	1349,485002	0,000000066840313049118
500	0,0001042	1349,485002	0,000000077214641016831
500	0,0000803	1349,485002	0,000000059504181129093
1000	0,0036498	3000	0,000001216600000000000
1000	0,0002099	3000	0,000000069966666666667
1000	0,0001792	3000	0,000000059733333333333
2000	0,0002705	6602,059991	0,000000040972060289563
2000	0,0005205	6602,059991	0,000000078839029133891
2000	0,0003885	6602,059991	0,000000058845269584086
5000	0,0005878	18494,85002	0,000000031781874376433
5000	0,0014753	18494,85002	0,000000079768151581149
5000	0,0016612	18494,85002	0,000000089819598323463
10000	0,0036320	40000	0,000000090799975000000
10000	0,0019836	40000	0,000000049590000000000
10000	0,0012750	40000	0,000000031875000000000
		Total	0,000000250987516925073

Este algoritmo ofrece el peor promedio de la constante (0.00000025) dentro de los estudiados, pero no significa que posee el peor rendimiento de todos en cuanto a tiempo de ejecución, ya que está muy por debajo de los tiempos que marcó el InsertionSort. Su complejidad (n log(n)) es aceptable, ya que no sobrepasa casi nunca este valor de crecimiento y por lo tanto puede ser una solución factible. Es por estas razones que se hacen necesarias este tipo de pruebas, debido a que este algoritmo se puede denotar como la opción promedio de las 3 estudiadas, debido a que no es mejor que el Quicksort, pero tampoco ofrece tiempos tan altos como el InsertionSort. En resumen, MergeSort puede llegar a ser elegida para estos casos, ya que su rendimiento es casi el mismo comparado al QuickSort.