Taller 3

Divide y Vencerás

Fundamentos de Análisis y Diseño de Algoritmos

Daniel Camelo 202159908 - 2724
Jhon Stiven Mafla 202160295 - 2724
Santiago Perez 201968142 - 2711

Universidad del Valle
Sede Tuluá
2023 - II

Punto 1

Divide y vencerás es un paradigma que consiste en resolver un problema dividiéndolo en subproblemas más pequeños, en el algoritmo StoogeSort se implementan los siguientes pasos:

a) Primero el algoritmo compara el primer elemento del arreglo con el último elemento del arreglo, donde si el primer elemento es mayor que el último; estos se intercambian de posición
b) Donde después realizamos recursividad, separando el arreglo en segmentos más pequeños donde nuevamente compara el primero con el último elemento del segmento y se intercambian si es necesario y nuevamente se vuelve a separar el arreglo en otro segmento donde se vuelve a comparar el primer elemento con el último elemento hasta lograr que los segmentos del arreglo estén en solución trivial.
c) Por último, se recorre totalmente el arreglo verificando que todos los elementos del arreglo se encuentren en el orden correcto; de no ser así, se vuelve a iniciar desde el primer paso.

La complejidad teórica del algoritmo StoogeSort es:

- (10^2.71) = 512,861384
- (100^2.71) = 263026,7992
- (1000^2.71) = 134896288,3
- (10000^2.71) = 6,918309709x10^10

Punto 2

a) Para implementar este algoritmo primero se divide el vector en subvectores en el punto medio
b) Después se aplica la recursión para encontrar la moda de cada subvector, se combinan las modas respectivas
c) Posteriormente se cuenta la frecuencia se localiza la máxima y se agrega al conjunto de modas combinas todos los elementos con frecuencia máxima
d) Mientras que la complejidad teórica sugiere O(nlogn), donde n es el tamaño del arreglo original, la complejidad práctica podría variar y depender de las características específicas de la implementación y del entorno de ejecución. La optimización y el perfilado del código podrían ser útiles para evaluar el rendimiento real del algoritmo en situaciones específicas.

La complejidad teórica para el algoritmo de la moda es:

- (10*log(10)) = 10
- (100*log(100)) = 200
- (1000*log(1000)) = 3000
- (10000*log(10000)) = 4000

Punto 3

Size	QuickSort	Insertion-Sort	Merge-Sort
10	0.000022	0.000007	0.000023
10	0.000017	0.000007	0.000019
10	0.000016	0.000008	0.000017
50	0.000071	0.000065	0.000099
50	0.000108	0.000097	0.000110
50	0.000091	0.000083	0.000104
100	0.000219	0.000388	0.000262
100	0.000155	0.000226	0.000230
100	0.000127	0.000289	0.000162
100	0.000129	0.000196	0.000151
500	0.000607	0.005374	0.000911
500	0.000669	0.005693	0.000929
500	0.000640	0.005410	0.000841
1000	0.001417	0.026583	0.001919
1000	0.001406	0.023890	0.001846
1000	0.001259	0.023316	0.001920
2000	0.002588	0.094056	0.005977
2000	0.003933	0.095553	0.004077
2000	0.002561	0.150717	0.007360
5000	0.010015	1	0.021074
5000	0.008996	1	0.019386
5000	0.009197	0.746053	0.011446
10000	0.011412	2	0.024166
10000	0.010117	2	0.026501
10000	0.011665	2	0.024339

Quicksort	nlogn		
Entrada	tiempo real	complejidad	constantes
10	0.000022	33,2192809	#¡VALOR!
10	0.000017	33,2192809	#¡VALOR!
10	0.000016	33,2192809	#¡VALOR!
50	0.000071	282,192809	#¡VALOR!
50	0.000108	282,192809	#¡VALOR!
50	0.000091	282,192809	#¡VALOR!
100	0.000219	664,385619	#¡VALOR!
100	0.000155	664,385619	#¡VALOR!
100	0.000127	664,385619	#¡VALOR!
100	0.000129	664,385619	#¡VALOR!
500	0.000607	4482,89214	#¡VALOR!
500	0.000669	4482,89214	#¡VALOR!
500	0.000640	4482,89214	#¡VALOR!
1000	0.001417	9965,78428	#¡VALOR!
1000	0.001406	9965,78428	#¡VALOR!
1000	0.001259	9965,78428	#¡VALOR!
2000	0.002588	21931,5686	#¡VALOR!
2000	0.003933	21931,5686	#¡VALOR!
2000	0.002561	21931,5686	#¡VALOR!
5000	0.010015	61438,5619	#¡VALOR!
5000	0.008996	61438,5619	#¡VALOR!
5000	0.009197	61438,5619	#¡VALOR!
10000	0.011412	132877,124	#¡VALOR!
10000	0.010117	132877,124	#¡VALOR!
10000	0.011665	132877,124	#¡VALOR!
constante			#¡VALOR!

insertion	n^2		
Entrada	tiempo real	complejidad	constantes
10	0.000007	100	#¡VALOR!
10	0.000007	100	#¡VALOR!
10	0.000008	100	#¡VALOR!
50	0.000065	2500	#¡VALOR!
50	0.000097	2500	#¡VALOR!
50	0.000083	2500	#¡VALOR!
100	0.000388	10000	#¡VALOR!
100	0.000226	10000	#¡VALOR!
100	0.000289	10000	#¡VALOR!
100	0.000196	10000	#¡VALOR!
500	0.005374	250000	#¡VALOR!
500	0.005693	250000	#¡VALOR!
500	0.005410	250000	#¡VALOR!
1000	0.026583	1000000	#¡VALOR!
1000	0.023890	1000000	#¡VALOR!
1000	0.023316	1000000	#¡VALOR!
2000	0.094056	4000000	#¡VALOR!
2000	0.095553	4000000	#¡VALOR!
2000	0.150717	4000000	#¡VALOR!
5000	1,086256	25000000	4,35E-08
5000	1,077186	25000000	4,30874E-08
5000	0.746053	25000000	#¡VALOR!
10000	2,374476	100000000	2,37448E-08
10000	2,349675	100000000	2,34968E-08
10000	2,327565	100000000	2,32757E-08
constante			#¡VALOR!

merge	nlogn		
Entrada	tiempo real	complejidad	constantes
10	0.000023	33,21928095	#¡VALOR!
10	0.000019	33,21928095	#¡VALOR!
10	0.000017	33,21928095	#¡VALOR!
50	0.000099	282,1928095	#¡VALOR!
50	0.000110	282,1928095	#¡VALOR!
50	0.000104	282,1928095	#¡VALOR!
100	0.000262	664,385619	#¡VALOR!
100	0.000230	664,385619	#¡VALOR!
100	0.000162	664,385619	#¡VALOR!
100	0.000151	664,385619	#¡VALOR!
500	0.000911	4482,892142	#¡VALOR!
500	0.000929	4482,892142	#¡VALOR!
500	0.000841	4482,892142	#¡VALOR!
1000	0.001919	9965,784285	#¡VALOR!
1000	0.001846	9965,784285	#¡VALOR!
1000	0.001920	9965,784285	#¡VALOR!
2000	0.005977	21931,56857	#¡VALOR!
2000	0.004077	21931,56857	#¡VALOR!
2000	0.007360	21931,56857	#¡VALOR!
5000	0.021074	61438,5619	#¡VALOR!
5000	0.019386	61438,5619	#¡VALOR!
5000	0.011446	61438,5619	#¡VALOR!
10000	0.024166	132877,1238	#¡VALOR!
10000	0.026501	132877,1238	#¡VALOR!
10000	0.024339	132877,1238	#¡VALOR!
constante			#¡VALOR!

En este caso en que se basa la prueba, la ejecución es casi cero ya que los valores dependen de la máquina en la que se ejecuta.

Conclusiones

1. *Punto 1*

Podemos concluir que el algoritmo stooge-sort es un algoritmo de ordenamiento recursivo, pero debemos de tener en cuenta que este algoritmo es ineficiente en términos de ejecución, ya que su tiempo de ejecución es demasiado alto teniendo una de complejidad O(n^2.71), esto hace que entre más grande sea el número de elementos que un array tenga, mucho mayor será su tiempo de ejecución, lo que nos demuestra lo ineficiente que es para ordenar grandes cantidades de datos.

2. *Punto 2*

Para concluir con este algoritmo, el cual presenta una solución eficiente para encontrar la moda de un vector mediante la implementación de la metodología divide y vencerás. Donde encontramos que su complejidad teórica es O(n log n), indicando una eficiencia razonable La división del vector y la conquista de subproblemas se realizan de manera estructurada, y la combinación de soluciones se logra eficientemente mediante operaciones de conjunto. Aunque la complejidad práctica puede variar según la implementación y el entorno, el algoritmo ofrece una solución versátil y eficaz para encontrar la moda.

3. *Punto 3*

De acuerdo con los tiempos de ejecución, se puede ver que el programa más rápido es el quicksort ya que su tiempo de ejecución es 0.11 segundos y el más lento es el insertionsort ya que su tiempo de ejecución es de 2 segundos, esto nos demuestra que hay una gran diferencia en el tiempo de ejecución de estos programas.
El quicksort y el mergesort son programas con una complejidad más baja que el insertion y tienen un tiempo de ejecución más bajo haciéndolos más eficientes, mientras que el insertion tiene una complejidad n^2 por lo que se demora más y es menos eficiente