

respuestas

E01: Demuestre que $6n^3 \neq O(n^2)$.

En programación cuando se utiliza big O se quiere decir que la función no va a crecer más que la función en big O.

entonces: $6n^3 = O(n^2)$ es equivalente $6n^3 \leq n^2$

y en $n = 1$ $6 \cdot 1^3 \leq 1^2$

$6 \leq 1$ es falso. entonces : $6n^3 \neq O(n^2)$

E02: ¿Cómo sería un array de números (mínimo 10 elementos) para el mejor caso de la estrategia de ordenación Quicksort(n) ?

Según el criterio para elegir el pivote debería cambiar la lista:

Si se tomará el primer elemento la lista un ejemplo del mejor caso sería :

6	3	9	1	2	4	5	7	8	10	11
---	---	---	---	---	---	---	---	---	----	----

$$\begin{array}{ccccccc}
 & & & & 6 & & \\
 & & & & & & \\
 & & 3 - 1 - 2 - 4 - 5 & & 9 - 7 - 8 - 10 - 11 & & \\
 & & 3 & & 9 & & \\
 & 1 - 2 & & 4 - 5 & & 7 - 8 & & 10 - 11
 \end{array}$$

Si se tomara como pivote el último debería poner la misma lista al revés para el mejor caso.

Si se tomará el pivote en el centro debería ponerse la lista ordenada para el mejor caso.

E03:Cuál es el tiempo de ejecución de la estrategia Quicksort(A), Insertion-Sort(A) y Merge-Sort(A) cuando todos los elementos del array A tienen el mismo valor?

En QuickSort se tendría una complejidad $O(n^2)$ (en teoría pero con se me ocurre una implementación donde si podría hacer en una complejidad de $O(n)$ solo agregando un contador que se encargue de contar los nodos iguales al pivote)

En Merge-Sort se tendría una complejidad $O(n \cdot \log(n))$ (como en todos los casos)

En Insertion-Sort se tendría una complejidad $O(n)$ (es uno de los mejores caso para Insertion-Sort)

E06:

Radix sort es un algoritmo de ordenamiento que ordena enteros procesando sus dígitos de forma individual. Como los enteros pueden representar cadenas de caracteres (como,

nombres o fechas) y, especialmente, números en coma flotante, radix sort no está limitado solo a los enteros. Se puede ordenar de izquierda a derecha o de derecha a izquierda.

radix sort se basa en el utilización de llaves para el ordenamiento:

ejemplo (de derecha a izquierda) ;

25 57 48 37 12 92 86 33 02 76 63

primer paso: UNIDADES

paso dos: DECENAS

llaves

llaves

0:

0: 02

1:

1: 12

2: 12 02 92

2: 25

3: 63 33

3: 33 ; 37

4:

4: 48

5: 25

5: 57

6: 86 76

6: 63

7: 37 57

7: 73

8: 48

8: 86

9:

9: 92

lista UNIDADES

lista final:

12 09 92 63 33 25 86 76 37 57 48

02 12 25 33 37 48 57 63 73 86 92

La complejidad de esta forma de ordenamiento dependerá, no solo de la entrada(n) sino de cuántas llaves habrá y de cuántos dígitos tendrán las entradas(m).por lo que la complejidad sería $O(m \times n)$

E07:

a. $T(n) = 2T(n/2) + n^4$

$a=2$; $b = 2$; $f(n) = n^4$

caso 3: existe $\epsilon > 0$ tal que: $f(n) = \Omega(n^{\log_b(a) + \epsilon})$

$n^4 = \Omega(n^{\log_2(2) + \epsilon}) \quad \epsilon = 3$

$af(n/b) \leq cf(n)$ para alguna constante $c < 1$

$2(n/2)^4 \leq cn^4$

$(n^4/8) \leq cn^4 \quad c = 1/8$

$T(n) = \Theta(n^4)$

b. $T(n) = 2T(7n/10) + n$

$a=2; b=10/7; f(n)=n \quad \log_{10/7}(2) > 1$

caso 1: existe $\varepsilon > 0$ tal que: $f(n) = \Omega(n^{\log_b(a) - \varepsilon})$
 $n = \Omega(n^{\log_{10/7}(2) - \varepsilon})$

$\varepsilon = \lceil \log_{10/7}(2) \rceil - 1$

$T(n) = \Theta(n^{\log_{10/7}(2)})$

c. $T(n) = 16T(n/4) + n^2$

$a=16; b=4; f(n)=n^2 \quad \log_4(16) = 2$

caso 2: $f(n) = \Theta(n^{\log_b(a)})$

$n^2 = \Theta(n^{\log_4(16)})$

$f(n) = \Theta(n^2)$

$T(n) = \Theta[n^2 \log(n)]$

d. $T(n) = 7T(n/3) + n^2$

$a=7; b=3; c=2 \quad \log_3(7) < 2$

caso 1: $T(n) = \Theta(n^2)$

e. $T(n) = 7T(n/2) + n^2$

$a=7; b=2; c=2 \quad f(n)=n^2 \quad \log_2(7) > 2$

caso 3: $T(n) = \Theta(n^{\log_2(7)})$

$$af(n/b) \leq cf(n) \quad \text{para alguna constante} \quad c < 1$$

$$7(n/2)^2 \leq cn^2$$

$$7(n^2/4) \leq cn^2$$

No existe $c < 1$ que cumpla esta condición

f. $T(n) = 2T(n/4) + \sqrt{n}$

$$a=2; b=4; c=1/2 \quad \log_2(4) = 2$$

caso 2: $T(n) = \Theta(n^{1/2} \log(n))$