**1. El procedimiento llamado InsertionSort toma como parámetro un arreglo A[1. . . n] que contiene una secuencia de enteros que deben ordenarse. El algoritmo ordena los números de entrada en el mismo arreglo.**

**a) Análisis de complejidad temporal de InsertionSort**

Procedure InsertionSort(A)

For j ← 2 to n do n + 1

key ← A[j] 2(n – 1)

i ← j – 1 2(n – 1)

while i > 0 and A[i]>key do 4

A[i+1] ← A[i]

i ← i -1

A[i+1] ← key 3(n – 1)

// j ← j+1 2(n – 1)

Suma de los costos:

*n+1 + 9(n - 1) + 4 + = 10n - 8 + 4[ n\*(n+1)/2) – 1 ] + 6[ (]*

**T(n)=** , que esta en O().

**b) ¿Es mas rápido el ordenamiento de la burbuja o InsertionSort?**

Procedure Burbuja(A)

for i ← 1 to n do (n-1+1)+1+1 = n+2

for j ←1 to (n – i) do

if A[j] > A[j+1] 4

aux ← A[j] 2

A[j] ← A[j+1] 4

A[j+1] ←aux 3

//j←j+1 2

//i←i+1 2n

Realizando la suma de los costos tenemos que la ordenación por burbuja esta en O(). De acuerdo a los valores que tomen las constantes a, b y c en la función , puede que una ordenación sea ligeramente más rápida que la otra, pero esta diferencia es despreciable.

Por lo tanto la ordenación por burbuja tiene la misma complejidad temporal que la ordenación por inserción.

**2. Considere el problema de abajo el cual que se conoce como el problema de búsqueda lineal. Problema de búsqueda lineal.**

ENTRADA: una secuencia den números A=〈a1, a2, . . . , an〉y un valor ν.

SALIDA: un índice tal que ν=A[i] o el valor especial NIL si ν no aparece en A.

**(a) Escriba un pseudocódigo para la búsqueda lineal, que explora la secuencia buscando ν.**

procedure BusquedaLineal(A, v)

j ←1 1

i ← 0 1

while j ≤ n and i = 0 3(n+1)

if A[j] = v 2n

i ← j 1

j ← j + 1 2n

return i 1

**(b) Hacer el análisis de complejidad temporal de su algoritmo. ¿Cuánto es su costo asintótico?**

Su costo es T(n) = 7n + 7, el cual está en O(n)

**(c) ¿Se puede hacer un algoritmo con costo O(√n) para el problema de búsqueda lineal? Argumente su respuesta.**

Un algoritmo de búsqueda que tenga una cota menor a O(√n) es el algoritmo de búsqueda binaria.

Suponiendo que el arreglo este ordenado de forma creciente y el tamaño del arreglo es n, en la búsqueda binaria el “tamaño” de nuestra zona de búsqueda se reduce sucesivamente, siguiendo la forma n/.

En caso de no encontrarse el elemento, el tamaño de la zona de búsqueda llegaría a n/ = 1, entonces:

n = ↔ n = x. Por lo tanto, el algoritmo de búsqueda binaria esta en O(n).

Por limites: = 0 , entonces O(√n) → o(√n)

**3. Considere ordenar n números almacenados en un arreglo A de la siguiente forma. Primero se encuentra el elemento más pequeño de A y se intercambia con el elemento en A[1]. Luego se encuentra el segundo elemento mas pequeño de A y se intercambia con A[2]. Continúe de esta manera para los primeros n−1 elementos de A.**

**(a) Escriba un pseudocódigo para SelectionSort.**

procedure SelectionSort(A)

for i←1 to n-1 do {

min ← A[i]

indiceMin←i

intercambio←0

for j←i+1 to n do { ≈

if A[j] < min {

min←A[j]

indiceMin←j

intercambio←1

}

if intercambio = 1{

aux← A[i]

A[i] ← A[indiceMin]

A[indiceMin] ← aux

}

}

**(b) ¿Por qué necesita ejecutarse solo para los primeros n−1 elementos, en lugar de los n elementos?**

Debe ejecutarse de esta manera porque al principio se asume que el elemento en posición i es el menor de todos los elementos desde i+1 hasta n.

Si i llegara hasta n, la posición del elemento siguiente seria n+1, el cual sobrepasa el tamaño del arreglo.

**(c) Proporcione en notación asintótica el mejor y el peor de los tiempos de ejecución de SelectionSort. ¿Cuál es la entrada que hace que el algoritmo ejecute el número mínimo de operaciones básicas? ¿Cuál es la entrada que hace que el algoritmo ejecute el número máximo de operaciones básicas?**

I) A causa de los dos bucles en el algoritmo, SelectionSort está en el orden O().

1. La entrada que hace que el algoritmo ejecute el número mínimo de operaciones básicas es un arreglo ordenado ascendentemente.

Al estar ordenado ascendentemente:

* No se actualiza los valores de las variables que contienen los datos del menor elemento
* No se realiza el intercambio del “menor elemento” y A[i]

Ω ()

1. La entrada que hace que el algoritmo ejecute el número máximo de operaciones básicas es un arreglo ordenado descendentemente.

Al estar ordenado descendentemente:

* Siempre se actualiza los valores de las variables que contienen los datos del menor elemento
* Siempre se realiza el intercambio del “menor elemento” y A[i]

En este caso el arreglo se invierte, haciendo que el algoritmo ejecute el número máximo de operaciones básicas

O().

**4. Sea A un arreglo de n números naturales distintos. Si i < j y A[i]> A[j], entonces el par (i, j) se denomina una inversión de A.**

**(a) Listar las cinco inversiones del arreglo〈2,3,8,6,1〉.**

i) (8,6)

ii) (8,1)

iii) (6,1)

iv) (3,1)

v) (2,1)

**(b) ¿Cual es el arreglo con elementos del conjunto {1,2, . . . , n} que tiene mas inversiones?¿Cuantos tiene? Argumente su respuesta.**

Si tenemos un arreglo con n elementos, entonces el número total de pares que tendremos será:

.

Para una ordenación aleatoria, se espera que la mitad de esos pares sean inversiones: =

Por lo tanto tendrá aproximadamente inversiones.

**(c) ¿Cual es la relación entre el tiempo de ejecución de InsertionSort y el número de inversiones en el arreglo de entrada? Justifica tu respuesta.**

En el segundo bucle de InsertionSort ( while i > 0 and A[i]>key do ) se puede apreciar la relación entre el tiempo de ejecución del algoritmo y el numero de inversiones.

En el mejor caso, si alguna de las dos condiciones no se cumple, no se ingresaría a ese bloque y no se realizarían las inversiones. Entonces: T(n)= 10n – 8, que está en O(n).

En el peor caso (arreglo ordenado descendentemente) siempre se ingresaría al bloque y se realizarían las inversiones. Entonces T(n)= , que está en O().

**5. Justifica tu respuesta.**

**(a) ¿Es = O()?**

Como 0≤C≤∞ ,entonces g(n)=O(f(n)) → = O(.

**(b) ¿Es = O()?**

Como 0≤C≤∞ ,entonces g(n)=O(f(n)) → = O( por lo tanto ≠ O()

**6. Demuestre por inducción que el i-esimo numero de Fibonacci satisface la igualdad , donde =(1+√5)/ 2 es el numero de oro y es su conjugado.**

|  |  |  |  |
| --- | --- | --- | --- |
| i |  |  |  |
| 1 | 1 | 1 | 1 |
| 2 | 1 | 2 | 2 |
| 3 | 2 | 3 | 1.5 |
| 4 | 3 | 5 | 1.666666667 |
| 5 | 5 | 8 | 1.6 |
| 6 | 8 | 13 | 1.625 |
| 7 | 13 | 21 | 1.615384615 |

Hallando , a medida que aumentamos i, vemos como la razón va tendiendo a (1+√5)/ 2.

Esto quiere decir que:

↔ 1+ =

Si = , entonces 1+

, donde (1+√5)/ 2 y (1-√5)/ 2

Además tenemos que:

…

Si reemplazamos por y , y realizamos una resta:

Donde (1+√5)/ 2 - (1-√5)/ 2 = √5

Por lo tanto si y , despejando tenemos: **)/ √5**

**7. Crecimientos asintóticos relativos. Buscar en las referencias del curso las definiciones de las notaciones asintóticas (omega-grande), o (o-pequeña), ω(omega pequeña) y Θ (theta-grande) e indicar para cada par de expresiones (A,B) en la tabla de abajo si A es O, o, ωo Θ de B. Suponga que k ≥1, ꞓ>0 y C >1 son constantes. Su respuesta debe ser en forma de tabla con un ”SI” o un ”NO” escrito en cada casilla. Presente una breve argumentación de cada una de sus respuestas.**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| A | B | O | o | Ω | ω | Θ |
|  |  | NO | NO | SI | SI | NO |
|  |  | SI | SI | NO | NO | NO |
|  |  |  |  |  |  |  |
|  |  | NO | NO | SI | SI | NO |
|  |  | NO | NO | SI | SI | NO |
|  |  | SI | SI | NO | NO | NO |

a)

=

Como C=∞ ,entonces g(n)=ω(f(n)) → =ω()

0<C≤∞,entonces g(n)= Ω(f(n)) → =Ω()

b)

= =0 si 0<C<1

0≤C≤∞ ,entonces g(n)=O(f(n)) → = O(

C=0 , entonces g(n)=o(f(n))→ = o(

c)

= ==

=y

=

=(sin(n).ln(n))’

=cos(n).

y’=

d)

= == = ∞

Como C=∞ ,entonces g(n)=ω(f(n)) → =ω()

0<C≤∞,entonces g(n)= Ω(f(n)) → =Ω()

e)

= = =

ln(y)= ln()= ∞

= logc.ln(n)

y’=(

Como C=∞ ,entonces g(n)=ω(f(n)) → =ω()

0<C≤∞,entonces g(n)= Ω(f(n)) → =Ω()

f)

= == 0

0≤C≤∞ ,entonces g(n)=O(f(n)) → = O(

C=0 , entonces g(n)=o(f(n))→ = o(

Consultas

Demostración de la formula de Binet:

https://artofproblemsolving.com/wiki/index.php/Binet%27s\_Formula

Análisis de algoritmos:

Algorithms - S. Dasgupta, C. H.Papadimitriou and U. V. Vazirani

Diapositivas del Prof. Cristian Cappo