Algoritmia y Complejidad

Alejandro Madrazo 20170478

Tarea Número 1

Ejercicio Número 1

Escriban pseudocodigo para un algoritmo de busquedad lineal, que escanee la sequencia n y encuentre un valor v. Describan cual es el loop invariant de su algoritmo para demostrar que es correcto.
Su algoritmo debe de contener un:
Input: Secuencia de n numeros A = [a 1 , a 2 , a 3 ... , a n]
Output: Indice i tal que v = A [i] o nulo si no se encuentra el valor.

Respuesta

```
Array A = [a1,a2,a3,a4,...,an]
V = undefined
for i = length.A down to 1 {
            if A[i] == V {
                print "Exito, se encontró ".N." en A[".i."] "
                exit
        } else if i == length.A{
            Print: "No se encontró ninguna coincidencia."
            exit
        }
        else {
            continue {
        }
    }
    end
```

Loop Invariant:

El loop invariant en este algoritmo, es que el largo del String no cambia y es fínito. Por lo cual al iterar a través de el, si no encuentra ningúna coincidencia terminará.

Otro loop invarian es V el cual no cambia entre iteraciones ni en toda la vida del algoritmo. Por lo cual encontrará coincidencia si la hay y si no lo indicará.

Ejercicio Número 2

Dado el siguiente algoritmo de multiplicacion de matrices:
Input: matriz A (n x m) y matriz B (m x p)
Output: matriz C (n x p)
Procedimiento running time (azul)

```
For i from 1 to n: n veces
    For j from 1 to p: p veces
        Let sum = 0
        For k from 1 to m: m veces
            Set sum <- sum + A[i][k] * B[k][j]
        Set C[i][j] <- sum
return C
```

Demuestre cual el es running time de este algoritmo.

Running time: Worst Case Running time = O(n*p*m)
Sin embargo puede darse el caso en que ambas matrices sean del mismo tamaño, entonces podemos simplificar nuestro Worst Case Running time a: $O(n^3)$

Ejercicio Número 3

Analisis:

```
    Bubble sort algorithm
S is an array of integers

for i in 1: length(S) - 1 do n
    for j in (i+1) : length(S) do n
        if S[i] > S[j] constante * n
            swap S[i] and S[j] constante * n
        end if
    end for
end for
```

Cual es el worst-case running time de este algoritmo?
Explicar como se compara el best-case y worst-case running time de insertion sort contra bubble sort?

Analisis:
Insertion Sort algorithm
A is an array of integers.

```
    for j in 2: length(A) do n
        key = A[j] Constante* n -1
        i = j -1 Constante * n -1
        while i > 0 and A[i] > key do constante * n - 1 * n -2
            A[i+1] = A[i] constante * n - 1 * n -2
            i = i -1 constante * n - 1 * n -2
        A[i+1] = key constante * n-1
```

Running time:

Bubble Sort:
Podemos ver que el loop exterior toma n-1 veces, mientras que el loop interno toma n veces, dentro de este loop interior se ejecutan una comparación y un swap que ambos se pueden considerar constantes.
Ambos, worst y best case tomarían el mismo tiempo de ejecución debido a que el algoritmo compara siemrpe si S[i] > S[j]
Entonces, podemos concluir que Bubble Sort tiene una complejidad de n - 1 * n veces. Si lo simplificamos es: n^2
Running time: Worst Case: O(n^2) Best Case: O (n^2)

Insertion Sort:

En insertion sort podemos notar que el loop exterior toma n comparaciones ejecutando las instrucciones internas n - 1 en el loop interior. Luego el número de movimientos y de comparaciones de llave que realiza ejecuta n*(n-1)/2 veces. Lo cual nos resulta en promedio en un tiempo de O(n^2). Sin embargo existe la posibilidad que tome menos tiempo. Este sería el caso en el que se le entrega como input un arreglo ordenado de manera ascendiente, esto implicaría que el loop interior no se ejectutaría por completo. Dejandonos solamente comparaciones de llave. Esto nos daría un best case de O(n^2).

En conlcusión podemos decir que:
Running time: Worst Case: O(n^2) Best Case: O (n)

Comparación:

La grán diferencia entre estos dos algoritmos de ordenamiento es que uno realiza una comparación antes de reccorer el arreglo. Mientras que el otro (Bubble Sort) realiza la comparación dentro del loop que recorre el arreglo. Entonces en el caso de estár ya ordenado, igualmente tendrá que hacer todo el trabajo. Es por esto que apesar de ambos tener un Worst Case: O(n^2), en promedio parecen ser igual de eficientes. Pero, si consideramos casos en los cuales el arreglo viene casí ordenado, uno desempeña mejor que otro. En este caso Insertion Sort.