Laboratorio 1 | Análisis Asintótico

Geordie Quiroa

August 2, 2018

1 - Problema de Búsqueda

Escribir en pseudocódigo un algoritmo de búsqueda lineal, que escanee la sequencia n y encuentre un valor v. Describir cuál es el loop invariant de su algoritmo para demostrar que es correcto.

Condiciones

Input: Secuencia de n numeros A = [a1 , a2 , a3, ... , an], y el valor a buscar en dicho arreglo (v).
Output: indice i tal que v = A[i] o nulo en caso de que el valor ingresado no se encuentre en el arreglo.

Algorithm 1 Busqueda Lineal

```
1: procedure BUSQUEDA-LINEAL(A, v)
2:     n ← a.length
3:     if n//2 == 0 then **No tiene residuo
4:         j = n/2
5:         i = j
6:     else
7:         i = (n/2).floor
8:         j = (n/2).up
9:     for i downto 1 do
10:        if A[i] != v then
11:            i − −
12:        else if A[i] == v then
13:            Return(i)
14:            Break
15:    if n//2 == 0 then
16:        j ← j + 1 **Para que no evalue dos veces
17:    else
18:        j ← j
19:    for j to n do
20:        if A[j] != v then
21:            j + +
22:        else if A[j] == v then
23:            Return(j)
24:            Break
25:        else
26:            Return("Value − Not − Found")
```

Loop Invariant
Los valores que anteceden el índice i de los ciclos For, no son equivalentes al valor que se busca.

2 - Running Time
| Algoritmo de Multiplicación matricial

Condiciones

Input: matriz A (n x m) y matriz B (m x p)
Output: matriz C (n x p)

Algorithm 2 Multiplicación Matricial

```
1: procedure MULTIPLICAR-MATS(A(NxM), B(MxP))
2:     for i ← 1 to N do
3:         for j ← 1 to P do
4:             Let sum = 0
5:             for k ← 1 to M do
6:                 Set sum ← sum + A[i][k] * B[k][j]
7:             Set C[i][j] < −sum
8:     Return(C)
```

Comprobación Running Time

El running time para el algoritmo de multiplicación de matrices no depende sólo de una variable sino de 3 (N, P y M) y las últimas dos forman parte una anidación de ciclos FOR, por lo que cada instrucción dentro del ciclo se ejecutará las veces que indica la variable del ciclo anidado por las veces que se ejecute la variable de los ciclos padre. Esto da como resultado un efecto en cadena, es decir, el ciclo mas interno se ejecutará NxPxM veces. De esta forma, el total de veces que las instrucciones de ese ciclo se van a ejecutar es el producto de las variables de los ciclos padre y del ciclo actual. Consecuentemente, el ciclo intermedio se ejecutará NxP veces y el ciclo principal se ejecutará N veces. Tomando en cuenta este efecto en cadena, el Running Time es el producto de las veces que se ejecuta cada instrucción de los ciclos:

(N)+(NxP)+(NxPxM)

En conclusión el algoritmo es de orden 3, debido a que el grado mayor del Running Time resultante es cúbico.

$$\mathbf{RunningTime} = \theta(\mathbf{N^3}) \quad ; \quad \theta(\mathbf{MxNxP})$$

3 - Running Time
| Algoritmos: Insertion Sort y Bubble Sort

Condiciones

- Indicar el worst case del algoritmo Bubble Sort.
- Comparar worst case y best case de los algoritmos Bubble Sort e Insertion Sort.

Algorithm 3 Bubble Sort Algorithm

```
1: procedure BUBBLE-SORT(S)
2:     S is an array of integers
3:     for i ← 1 to (S.length − 1) do
4:         for j ← (i + 1) to S.length do
5:             if S[i] > S[j] then
6:                 Swap S[i]andS[j]
```

Algorithm 4 Insertion Sort Algorithm

```
1: procedure INSERTION-SORT(A)
2:     A is an array of integers
3:     for j ← 2 to (A.length − 1) do
4:         // Insert A[j] into the sorted Sequence A[1..j-1]
5:         i = j-1
6:         while i > 0 and A[i] > key do
7:             A[i + 1] = A[i]
8:             i = i − 1
9:         A[i + 1] = key
```

Análisis Running Time

Worst case running time bubble sort: $\theta(\mathbf{N^2})$

El **best case y worst case running time de bubble sort** es el mismo en ambos casos; ya que no existe ninguna instrucción o condicional que evite que el ciclo interno FOR del ciclo padre FOR se ejecute n-1 veces. Por esta razón las instrucciones en el For interno se ejecutarán **(n-1)*(n-1)** veces independientemente del valor i y j. Dando como resultado $\theta\mathbf{n^2}$. De la misma forma, el **worst case running time de Insertion Sort** es de orden cuadrático, debido a que tendría que recorrer el arreglo con el subíndice j (n-1 veces) e irlo intercambiando con el subíndice i (n-1 veces). Sin embargo, el **best case de Insertion Sort** es diferente, ya que el ciclo while tiene condiciones con las cuales puede obviar las ejecuciones que

contiene dependiendo de los valores de i y la desigualdad A[i] > key. Por lo tanto, el best case de éste es $\mathbf{\Omega(n)}$; escenario en el cual todos los valores se encuentran ordenados de manera no decreciente.