Universidad del Valle de Guatemala

Facultad de Ingeniería

Ciencias de la Computación y Tecnologías de la Información



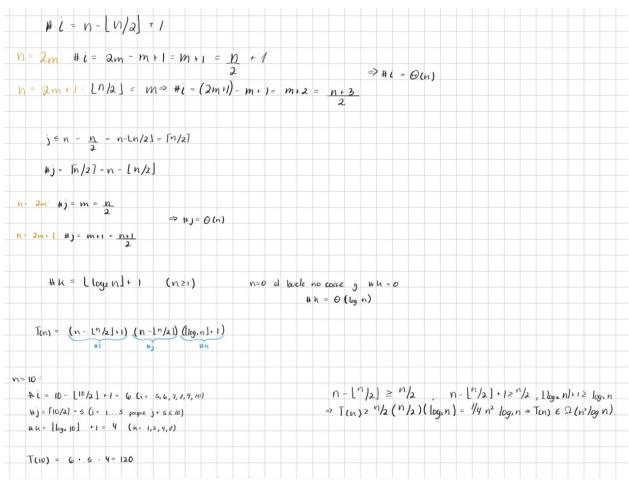
Laboratorio 8

Teoría de la computación

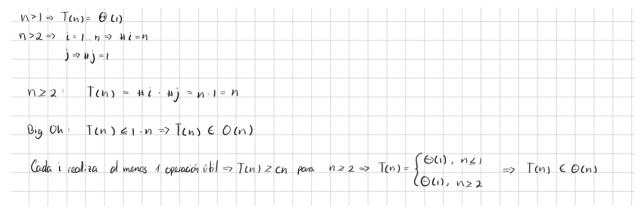
José Luis Gramajo Moraga, Carné 22907

Repositorio: https://github.com/Abysswalkr/TC-Lab08-Complejidad.git

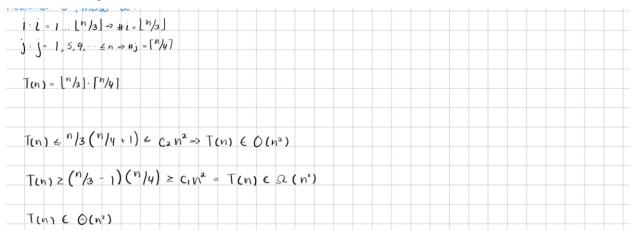
Problema 1



Problema 2



Problema 3



Problema 4

¿Cuál es el mejor?

- Para una búsqueda en arreglo ya ordenado:
 Mejor = Búsqueda Binaria (Θ(log n) peor/promedio) « Lineal (Θ(n)).
- Para una sola búsqueda en arreglo no ordenado (sin querer ordenar):
 Mejor = Búsqueda Lineal.
 (Ordenar para luego hacer Binaria cuesta Ω(nlog n), peor que mirar una sola vez Θ(n).)
- Para muchas búsquedas sobre el mismo arreglo no ordenado (m consultas):
 Ordenar una vez (Quick Sort, ≈ nlog n) y luego buscar con Binaria (mlog n).
 Comparación:

Lineal total $\approx m \, n \text{vsOrdenar+Binaria} \approx n \log n + m \log n$.

Ordenar+Binaria es mejor cuando

$$m \, n \, \> \; n \log \, n + m \log n \Rightarrow \; m \, \> \; \frac{n \log n}{n - \log n} \; \approx \; \log n.$$

Para ordenar (no para buscar):
 Quick Sort es mejor en promedio Θ(nlog n)(baja constante y buen uso de caché), pero su peor caso es Θ(n²); con pivote aleatorio/median-of-three se evita casi siempre ese peor caso.

) linear Se	sch: asseglo de tomañon, sin orden
Mejor caso:	
	C(n) = 1 => 0(1)
Peor caso:	
	$C(n) = n = \Theta(n)$
Caso prome	
y, y.m.e.	E[c] = 1+2++n = n+1
	n 2
	$E[c] = \rho \cdot \underbrace{n+1}_{2} + (1-p) \cdot \underbrace{n-n-p}_{2} (n-1) \Rightarrow \Theta(n)$
2) Binary	Search: arreglo ordenado de tamaño n
Mejor caso:	
	C(n) = 1 => O(1)
Peo: caso:	
	T(n) = T(ln/2)+1. T(1)=1 => C(n) = Llog2n] +1 => O(logn)
3) Quich so Mejor caso	st: participación lineal por nuel
	$T(n) = 2T(n/2) + (n \Rightarrow T(n) = \Theta(n \log n)$
Caso prome	
	$T(n) = \frac{1}{n} \sum_{k=0}^{n-1} (T(k) + T(n-1-k)) + cn$
Peor caso:	
	$T(n) = T(n-1) + T(0) + cn => C(n) = n(n-1) \Rightarrow O(n^2)$

Problema 5

```
Inciso a: Verdadero
     \exists c_1, c_2 > 0, n_0 : c_1 g(n) \leq f(n) \leq c_2 g(n)
                                                              (nzno)
     \exists d_1, d_2 > 0, n_1 : d_1h(n) \leq g(n) \leq d_2h(n)
                                                              (n \ge n_i)
n > N = max { no, n, }
               c,d,h(n) = f(n) = (2 d2h(n)
          ⇒ f(n) = O(h(n)). Como Oes simétrica h(n) = O(f(n))
Inciso D: Verdadero
     3C>0, no: ((n) ≤ Cg(n),
                                           ∃D > Onn : g(n) & Dh(n)
  => ((n) & CDh(n) (n > N = max (no, n, ))
     f(n) = O(h(n)) \rightarrow f = O(h) \Leftrightarrow h = \Omega(f)
                      h(n) ≥ 1/co f(n) => h(n) = Q (f(n))
Inciso C: Folso (cn) = O(n3)
    1. tuple (range (0, n)) - O(n)
    2. Doble bude: poves (i,j) con 0 ≤ i zj ≤ n-1
     "En cada par se crea el slice "atupla [i:j] de longitud n=j-i y se hashea para insertar en el set.
     · Coste por iteración: O(h) = O(j-i)
    3. Suma total:
          T(n) = \sum_{k=0}^{\infty} \sum_{j=0}^{\infty} (j-1) = \sum_{k=1}^{n-1} (n-k) k = N \sum_{k=1}^{n-1} k - \sum_{k=1}^{n-1} k^2 = n \frac{(n-1)n - (n-1)n(2n-1)}{2} = \frac{(n-1)n(n+1)}{6} = O(n^3)
    4. El termino O(n) no cambia el o(den: f(n) = \text{O}(n^3)
```

Conclusión:

- 5.a Verdadero
- 5.b Verdadero
- 5.c Falso; el tiempo es $\Theta(n^3)$ no $\Theta(n^2)$.