Universidad del Valle de Guatemala Christopher Sandoval 13660
Análisis y Diseño de Algoritmos - Sección 10 17/03/2020

Tarea 1

Análisis de algoritmos y notaciones asintóticas

1. **Describa un algoritmo con tiempo de ejecución $O(n \log_2 n)$ tal que, dados un conjunto S de n números enteros y un entero arbitrario x, determine si existen o no dos números en S cuya suma sea exactamente x. Puede suponer que el arreglo está ordenado.**

   Primero se itera sobre todas las parejas del arreglo de entrada. Esto nos tomará un tiempo de O(n). Luego sobre la suma de cada una de esas parejas se aplica un algoritmo de búsqueda binaria, que tiene un tiempo de ejecución O(log2n). El resultado de realizar estos dos algoritmos nos da como resultado un tiempo de O(nlog2n).

2. **La Regla de Horner dice que se puede evaluar un polinomio $P(x) = \Sigma^{n}_{k=0}\, a_k x^k$ de la siguiente manera:**

   $$a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n) \ldots))$$

   El siguiente trozo de pseudocódigo implementa esta regla para un conjunto de coeficientes a_i dado:

   ```
   1. y=0
   2. for i=n downto 0:
   3. y=a_i+x*y
   ```

   Calcule una cota ajustada para el tiempo de ejecución de este algoritmo.

   El tiempo de ejecución es c1+c2n+c3(n-1). Es acotada por O(n). Donde 0<=k1n<=2n<=k2n donde k1=1 y k2=3

3. **Escriba código naïve para la evaluación de un polinomio (suponga que no hay una instrucción primitiva para calcular x^y). Compare las tasas de crecimiento de este código y el que implementa la Regla de Horner.**

   ```
   res = 0
   for (i=0; i<cantidad_polinomios; i++):
       valor_polinomio = x
       for (j=0; j<grado_polinomio; j++):
           valor_polinomio *= x
       res += valor_polinomio
   ```

   Como vemos la implementación naive del algoritmo tiene una complejidad de O(n^2) por otro lado el algoritmo que implementa la Regla de Horner tiene una complejidad menor, de solo O(n)

4. **Para dos funciones $f(n)$ y $g(n)$ demuestre que $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.**

   Tenemos que $f(n) \leq f(n)+g(n)$ y $g(n) \leq f(n)+g(n)$
   Por lo que
   $\max(f(n),g(n)) \in O(f(n)+g(n))$

   Tenemos que $f(n)+g(n) \leq 2\max(f(n),g(n))$.
   Por lo que
   $\max(f(n),g(n)) \in \Omega(f(n)+g(n))$

   Por lo tanto
   $\max(f(n),g(n)) \in \Theta(f(n)+g(n))$

5. **Argumente por qué, para constantes reales cualesquiera a y $b > 0$, $(n + a)^b = \Theta(n^b)$. Hint: puede investigar o deducir la forma expandida $(n + a)^b$ para apoyar su respuesta.**
   Porque expandido es igual a $n^b + c_1n^{b-1}a + \ldots + c_kna^{b-1} + a^b$ y como a^b es una constante entonces la complejidad es determinada por n^b.

6. **¿Es $2^{n+1} = O(2^n)$?**
   Si porque al multiplicar 2^n por una constante mayor a 2 podemos acotar por arriba a 2^{n+1}.
   ¿Es $2^{2n} = O(2^n)$?
   No porque no hay ninguna constante por la cual podamos multiplicar a 2^n para que acote siempre por arriba a 2^{2n}.

7. **Demuestre las siguientes propiedades:**
   a. **$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))\ \hat{}\ f(n) = \Omega(g(n))$.**
      Como la definición de O es $f(n) \leq c * g(n)$ y la definición de Ω es $f(n) \geq c * g(n)$ entonces si f(x) es acotado por $O(g(n))$ y $\Omega(g(n))$ entonces podemos concluir que f(x) es acotado por $\Theta(g(n))$
   b. **$o(g(n)) \cap \omega(g(n)) = \varnothing$.**
      Ya que la definición de little o y little ω acotan a la función sin incluir el valor de la función entonces al hacer una intersección entre estos dos conjuntos va a ser el conjunto vacío.
   c. **$f(n) = O(g(n)) \Rightarrow \log2 f(n) = O(\log2 g(n))$, donde sepamos que $\log_2 g(n) \geq 1$ y $f(n) \geq 1$ para n suficientemente grande (i.e., para $n \geq n_0$ con algún n_0).**
      Ya que la definición de O es $f(n) \leq c * g(n)$ entonces podemos ver que si agregamos $\log_2$ de los dos lados $\log_2(f(n)) \leq \log_2(c * g(n))$ la desigualdad se sigue cumpliendo