Universidad del Valle de Guatemala
Análisis y Diseño de Algoritmos - Sección 10
María Fernanda Estrada 14198
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.**
Este algoritmo consta de dos partes. Primero, se tienen que tomar dos números y sumarlos; al hacerlo con todo el conjunto de S, se tiene un tiempo de ejecución *O(n)*. Segundo, al tener ya estos números seleccionados, se procede a realizar una búsqueda binaria de la suma de los números; este proceso tiene tiempo de ejecución $O(log_2n)$. En total, ir sumando y buscando esa suma en todo el conjunto S, toma un tiempo de ejecución $O(n\ log_2n)$.

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 paso uno solo se ejecuta 1 vez, el paso dos se ejecuta n veces y el paso tres se ejecuta n-1 veces. En total, su tiempo de ejecución es $c_1 + c_2n + c_3(n-1)$. Ahora, para calcular una cota ajustada, se deben encontrar los valores k_1 y k_2 en $0 \leq k_1n \leq c_1 + c_2n + c_3(n-1) \leq k_2n$. Si se toman $c_1 = c_2 = c_3 = 1$, se encuentra que para cumplir estas condiciones, $k_1 = 1$ y $k_2 = 3$.

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.**
```
f = 0
for (i=0; i<k; i++):
    for (m=0; m<k; m++):
          x_beforea *= x
    x_aftera = a_k*x_beforea
    f += x_aftera
```

En la evaluación naive, se debe evaluar uno por uno todos los términos. Este algoritmo tiene una tasa de crecimiento de n^2, mientras que la Regla de Horner tiene tasa de crecimiento n, haciéndolo mucho más eficiente y rápido.

4. **Para dos funciones $f(n)$ y $g(n)$ demuestre que max($f(n)$, $g(n)$) = $\Theta(f(n) + g(n))$.**
   Se sabe que $f(n) \leq f(n) + g(n)$ y que $g(n) \leq f(n) + g(n)$. Entonces, se tiene que
   $$\max(f(n), g(n)) \in O(f(n) + g(n))$$
   Ahora, notamos que $f(n) + g(n) \leq 2\max(f(n), g(n))$. Entonces, se tiene que
   $$\max(f(n), g(n)) \in \Omega(f(n) + g(n))$$
   Como se acota por abajo y por arriba, se concluye que
   $$\max(f(n), g(n)) = \Theta(f(n) + g(n))$$

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.**
   Al expandir el polinomio, queda de la forma $n^b + Kn^{b-1}a + \ldots + Kna^{b-1} + a^b$. El polinomio más grande y el que acota la función es n^b, por lo que el tiempo de ejecución es $\Theta(n^b)$.

6. **¿Es $2^{n+1} = O(2^n)$? ¿Es $2^{2n} = O(2^n)$?**
   La primera afirmación es cierta. Para comprobar, se quiere saber si hay una constante c que al multiplicarse por 2^n el resultado es igual o mayor a 2^{n+1}. Si se separa 2^{n+1} en $2*2^n$, entonces se observa que si la constante 2 (o mayor) se cumple esta condición.
   La segunda afirmación es falsa. No hay una constante que se pueda multiplicar a 2^n para que sea mayor o igual a 2^{2n}.

7. **Demuestre las siguientes propiedades:**
   a. **$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n))$ ^ $f(n) = \Omega(g(n))$.**
      Por la definición de Θ, ésta función debe estar acotada por arriba (O) y por debajo (Ω). Y de regreso la afirmación es válida también; si una función es acotada por arriba y por abajo, es de complejidad Θ.
   b. **$o(g(n)) \cap \omega(g(n)) = \varnothing$.**
      Ambas notaciones acotan por arriba y por abajo, pero no son incluyentes. Es decir, que no comparten elementos con la función que acotan. Entonces al hacer la intersección de ambas, queda el conjunto vacío.
   c. **$f(n) = O(g(n)) \Rightarrow \log_2 f(n) = O(\log_2 g(n))$, 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).**
      Se cumple esta definición ya que si se multiplica la función y el acotamiento por otra función (en este caso $\log_2 x$), se sigue cumpliendo que el acotamiento es mayor a la función. Es decir, $\log_2 f(n) \leq \log_2 g(n)$.