Universidad del Valle de Guatemala
Departamento de Matemática
Licenciatura en Matemática Aplicada

Estudiante: Rudik Roberto Rompich
Correo: rom19857@uvg.edu.gt
Carné: 19857

Análisis y diseño de algoritmos - Catedrático: Tomás Galvéz
12 de marzo de 2023

Tarea

Problema 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.*

> *Hint: considere usar, como parte de su algoritmo, al algoritmo de búsqueda binaria, cuyo tiempo de ejecución es $O(\log_2 n)$.*

Solución. Sea $h(n) = n \log_2 n$. A encontrar: Un algoritmo con $O(h(n)) = \{f(n) : \exists c, n_0 > 0 | \forall n \geq n_0, 0 \leq f(n) \leq ch(n)\}$. Como pista, nos dicen que podemos integrar el algoritmo de búsqueda binaria.

El algoritmo de búsqueda binaria es un algoritmo que encuentra la posición de un número T dentro de un arreglo ordenado A con n elementos, que está ordenado de forma ascendente. Se define como:

```
binary_search(A, n, T):
    L = 0
    R = n - 1
    while L <= R:
        m = floor((L + R) / 2)
        if A[m] < T:
            L = m + 1
        else if A[m] > T:
            R = m - 1
        else:
            return m
    return unsuccessful
```

Este algoritmo tiene un tiempo de ejecución de $O(\log_2 n)$.

Entonces, esto nos indica que para obtener el tiempo de ejecución $h(n)$ debemos aplicarle un ciclo *for* al algoritmo de búsqueda binaria que es $O(\log_2 n)$. Sin embargo, primero debemos encontrar un algoritmo que ponga en orden un arreglo S con n elementos que debe ir insertado en el algoritmo del binary-search. Nótese que en clase ya habíamos estudiado el algoritmo de Merge-Sort el cual tiene un tiempo de ejecución $O(h(n))$.

El algoritmo de merge-sort es un algoritmo de ordenamiento, en donde A es un array desornado, en donde p es el primer elemento del array (se inicializa con $p = 1$) y r es la longitud del array:

```
merge_sort(A,p,r):
    if p<r:
        q = floor((p+r)/2)
        merge_sort(A,p,q)
        merge_sort(A,q+1,r)
        merge(A,p,q,r)
```

Este algoritmo tiene un tiempo de ejecución de $O(\log_2 n)$.

En resumidas cuentas, si juntamos en un algoritmo el merge-sort con un ciclo *for* del binary search, entonces obtendremos un algoritmo con tiempo de ejecución

$$O(h(n)) + O(h(n)) = 2O(h(n)) = 2O(n \log_2 n)$$

Pero el dos es una constante, entonces el algoritmo es de complejidad

$$O(n \log_2 n)$$

Entonces, el algoritmo propuesto, que dado un array S de n números enteros y un entero arbitrario x, que verifica que si existen o no dos números en S cuya suma sea exactamente x. Es decir, se busca verificar que

$$x = S[i] + S[j], \quad i, j \in S$$

Pero esto, esencialmente podría reescribirse como, al fijar un $S[j]$, debemos encontrar

$$S[j] = x - S[i]$$

Entonces $S[j]$ es nuestra candidato para aplicarle el binary-search, ya que si lo encontramos, quiere decir que hay dos números en S que al sumarlos dan x. Entonces, definimos nuestro algoritmo de la siguiente manera:

```
algoritmo_verificacion(S,x):
    p = 1
    n = length(S)
    S = merge_sort(S,p,n)
    for i=1 to n:
        S_j = x-S[i]
        verificador_j = binary_search(S,r,S_j)
        if verificajor_j != "unsuccessful":
            return "Verificacion exitosa"
    return "Verificacion fallida"
```

El cual por la argumentación anterior, es $O(n \log_2 n)$. □

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

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

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.

Solución. Analizamos por linea:

1. $\Theta(1)$
2. $\Theta(n)$
3. $\Theta(1)$

Usamos Θ ya que por definición Big-theta hace referencia a la cota ajustada. Entonces, la regla de Horner tiene una cota ajustada de $\Theta(n)$ ya que es el más grande del algoritmo. □

Problema 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.*

> *Nota: una solución naïve se refiere a la implementación de una solución de la forma más simple posible, haciendo uso de funciones u operaciones fundamentales y conocidas. Por ejemplo, la forma naïve de encontrar un elemento en un diccionario es ver la primera palabra y determinar si es la que buscamos. Si no es, pasamos a la siguiente y repetimos el proceso.*

Solución. Analizamos primero los casos, para darnos una idea, considerando

$$P(x) = \sum_{k=0}^{n} a_k x^k$$

1. $n = 0$,

$$\begin{aligned} P(x) &= \sum_{k=0}^{0} a_k x^k \\ &= 0 \end{aligned}$$

2. $n = 1$,

$$\begin{aligned} P(x) &= \sum_{k=0}^{1} a_k x^k \\ &= a_0 x^0 + a_1 x^1 = a_0 + a_1 x^1 \end{aligned}$$

3. $n = 2$,

$$\begin{aligned} P(x) &= \sum_{k=0}^{2} a_k x^k \\ &= a_0 + a_1 x^1 + a_2 x^2 \\ &= a_0 + x(a_1 + a_2 x) \end{aligned}$$

4. $n = 3$,

$$\begin{aligned} P(x) &= \sum_{k=0}^{3} a_k x^k \\ &= a_0 + a_1 x^1 + a_2 x^2 + a_3 x^3 \\ &= a_0 + x(a_1 + a_2 x + a_3 x^2) \\ &= a_0 + x(a_1 + x(a_2 + a_3 x)) \end{aligned}$$

Entonces, tenemos el pseudocódigo:

```
DECLARE ARRAY a[n]
DECLARE INTEGER N
N = LENGTH(a)

DECLARE INTEGER y
y = 0

FOR i = 0 TO N DO
    DECLARE INTEGER y_i
    y_i = 1

    FOR j = 1 TO i DO
        y_i = y_i * x
    END FOR

    y = y + a[i] * y_i
END FOR
```

□

Problema 4. *Para dos funciones $f(n)$ y $g(n)$ demuestre que*

$$\max(f(n), g(n)) = \Theta(f(n) + g(n)).$$

Demostración. Por la definición de Θ,

$$\Theta(f(n)+g(n)) = \{h(n) : \exists c_1, c_2, n_0 > 0 | \forall n \geq n_0, 0 \leq c_1 (f(n) + g(n)) \leq h(n) \leq c_2 (f(n) + g(n))\},$$

Es decir, lo que debemos probar es que

$$h(n) = \text{máx}(f(n), g(n)),$$

con sus respectivas constantes. Sin pérdida de generalidad, asúmase que $f(x)$ y $g(x)$ son asintóticamente positivas, es decir $f(x) \geq 0, n > n_1$ y $g(x) \geq 0, n > n_2$. Sea $n_0 := \text{máx}(n_1, n_2)$. Entonces, debemos encontrar las dos cotas, la superior e inferior:

- Inferior. Por definición de máx, tenemos:

$$\begin{aligned} f(x) &\leq \text{máx}(f(n), g(n)) \\ g(x) &\leq \text{máx}(f(n), g(n)) \end{aligned}$$

  Sumamos ambas expresiones:

$$f(x) + g(x) \leq \text{máx}(f(n), g(n)) + \text{máx}(f(n), g(n)) = 2\,\text{máx}(f(n), g(n))$$

  Entonces:

$$\frac{1}{2}\left(f(x) + g(x)\right) \leq \text{máx}(f(n), g(n))$$

- Superior, en este caso, es trivial:

$$(1)(f(x) + g(x)) \geq \text{máx}(f(n), g(n))$$

Entonces, $c_1 = \frac{1}{2}$, $c_2 = 1, n_0 = \text{máx}(n_1, n_2)$, tal que:

$$\frac{1}{2}\left(f(x) + g(x)\right) \leq \text{máx}(f(n), g(n)) \leq f(x) + g(x)$$

Por lo tanto,

$$\text{máx}(f(n), g(n)) = \Theta(f(n) + g(n)).$$

■

Problema 5. *Argumente por qué, para constantes reales cualesquiera a y $b > 0$, $(n+a)^b = \Theta\left(n^b\right)$.*

> *Hint: puede investigar o deducir la forma expandida $(n + a)^b$ para apoyar su respuesta.*

Demostración. Sea $\Theta\left(n^b\right)$, definido como:

$$\Theta(n^b) = \left\{h(n) : \exists c_1, c_2, n_0 > 0 | \forall n \geq n_0, 0 \leq c_1\left(n^b\right) \leq h(n) \leq c_2\left(n^b\right)\right\}$$

Entonces debemos encontrar que $h(n) = (n + a)^b$ con sus respectivas constantes. Por hipótesis, $a, b \in \mathbb{R}$ y $b > 0$. Entonces, analizamos la expresión $(n + a)^b$, en donde debemos encontrar su cota inferior y superior:

- Superior. Nótese que $n \geq n_0 > 0$, entonces por desigualdad triangular,

$$\begin{aligned} n + a &\leq |n + a| \\ &\leq \underbrace{|n|}_{n>0} + |a| \\ &\leq n + |a| \end{aligned}$$

  Tenemos que, a partir de cierto punto $|a| \leq n$

$$\leq n + n = 2n$$

  Entonces, como $b > 0$, elevamos a la b ambos lados de la desigualdad:

$$(n+a)^b \leq (2n)^b = 2^b n^b$$

- Inferior. Nótese que $n \geq n_0 > 0$, entonces por desigualdad triangular,

$$\begin{aligned} n + a &\geq |n - a| \\ &\geq \underbrace{|n|}_{n>0} - |a| \\ &\geq n - |a| \end{aligned}$$

  Tenemos que, a partir de cierto punto $|a| \leq n \implies 2|a| \leq 2n \implies 2|a| \leq n \leq 2n \implies 2|a| \leq n \implies |a| \leq n/2 \implies -|a| \geq -n/2$

$$\geq n - \frac{n}{2} = \frac{n}{2}$$

  Entonces, como $b > 0$, elevamos a la b ambos lados de la desigualdad:

$$(n+a)^b \geq \left(\frac{n}{2}\right)^b = \left(\frac{1}{2}\right)^b n^b$$

Por lo tanto,

$$n_0 = 2|a|, \quad c_1 = \left(\frac{1}{2}\right)^b, \quad c_2 = 2^b$$

Tal que

$$\left(\frac{1}{2}\right)^b \left(n^b\right) \leq (n+a)^b \leq 2^b \left(n^b\right)$$

Por lo tanto,

$$(n+a)^b = \Theta\left(n^b\right)$$

$\blacksquare$

Problema 6. *¿Es* $2^{n+1} = O\left(2^n\right)$ *? ¿Es* $2^{2n} = O\left(2^n\right)$ *?*

Solución. Por definición de $O(2^n)$,

$$O(2^n) = \{f(n) : \exists n_0, c > 0 | \forall n \geq n_0, 0 \leq f(n) \leq c2^n\}$$

Entonces, nótese que $2^{n+1} = 2^n 2$ y $2^{2n} = ((2)^n)^2$. Es decir, $c = 2$. Por lo tanto,

$$2^{n+1} = O\left(2^n\right)$$

y

$$2^{2n} \neq O\left(2^n\right)$$

ya que no se puede obtener la constante c. □

Problema 7. *Demuestre las siguientes propiedades:*

1. $f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$.

   Demostración. Por doble implicación:

   - ($\Longrightarrow$) Sea

   $$\begin{aligned} f(n) &= \Theta(g(n)) \\ &= \{f(n) : \exists c_1, c_2, n_0 > 0 | \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\} \\ &= \{f(n) : \exists c_1, n_0 > 0 | \forall n \geq n_0, 0 \leq c_1 g(n) \leq f(n)\} \wedge \\ &\quad \wedge \{f(n) : \exists c_2, n_0 > 0 | \forall n \geq n_0, 0 \leq f(n) \leq c_2 g(n)\} \\ &= [f(n) = \Omega(g(n))] \wedge [f(n) = O(g(n))] \\ &= [f(n) = O(g(n))] \wedge [f(n) = \Omega(g(n))] \end{aligned}$$

   - ($\Longleftarrow$) Sea

   $$\begin{aligned} [f(n) = O(g(n))] \wedge [f(n) = \Omega(g(n))] &= \{f(n) : \exists c_2, n_1 > 0 | \forall n \geq n_1, 0 \leq f(n) \leq c_2 g(n)\} \wedge \\ &\quad \wedge \{f(n) : \exists c_1, n_2 > 0 | \forall n \geq n_2, 0 \leq c_1 g(n) \leq f(n)\} \end{aligned}$$

   Sea $n_0 = \text{máx}(n_1, n_2)$,

   $$\begin{aligned} &= \{f(n) : \exists c_1, c_2, n_0 > 0 | \forall n \geq n_0, \\ &\quad 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\} \\ &= [f(n) = \Theta(g(n))] \end{aligned}$$

   Por lo tanto,

   $$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \wedge f(n) = \Omega(g(n))$$

   ■

2. $o(g(n)) \cap \omega(g(n)) = \varnothing$.

Demostración. Sea

$$
\begin{aligned}
f(n) &= o(g(n)) \cap \omega(g(n)) \\
&= \{f(n) : \exists c_2, n_1 > 0 | \forall n \geq n_1, 0 \leq f(n) < c_2 g(n)\} \cap \\
&\quad \cap \{f(n) : \exists c_1, n_2 > 0 | \forall n \geq n_2, 0 \leq c_1 g(n) < f(n)\}
\end{aligned}
$$

Sea $n_0 = \text{máx}(n_1, n_2)$,

$$= \{f(n) : \exists c_1, c_2, n_0 > 0 | \forall n \geq n_0, 0 \leq c_1 g(n) < f(n) < c_2 g(n)\}$$

Es decir que $o(g(n)) \cap \omega(g(n))$ nunca se llegan a intersectar a partir de un punto. Por lo tanto su interseccion es el $\varnothing$. ■

3. $f(n) = O(g(n)) \Rightarrow \log_2 f(n) = O(\log_2 g(n))$, *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 *).*

Solución. Sea

$$
\begin{aligned}
f(n) &= O(g(n)) \\
&= \{f(n) : \exists c, n_0 > 0 | \forall n \geq n_0, 0 \leq f(n) \leq c g(n)\}
\end{aligned}
$$

Como sabemos que $\log_2 g(n) \geq 1$ y $f(n) \geq 1$ para $n \geq n_0$, es decir

$$\{\log_2 f(n) : \exists c, n_0 > 0 | \forall n \geq n_0, 0 \leq \log_2 f(n) \leq c \log_2 g(n)\}$$

Por lo tanto,

$$\log_2 f(n) = O(\log_2 g(n)).$$

□