75.04 Algoritmos y Programación II

Práctica 4: complejidad algorítmica y recurrencias

Notas preliminares

- Esta práctica se propone sentar los fundamentos matemáticos básicos para el análisis de algoritmos.
- Los ejercicios marcados con el símbolo ♣ constituyen un subconjunto mínimo de ejercitación. No obstante, recomendamos fuertemente realizar todos los ejercicios.

---

1. Notación asintótica

Ejercicio 1

Supongamos tener dos algoritmos para el mismo problema: el algoritmo A insume tiempo n, mientras que el algoritmo B toma tiempo $3 \log n + 5$. ¿Cuándo se debería elegir A y cuándo B?

Ejercicio 2

Demostrar las siguientes propiedades:

(a) $f(n) \in \mathrm{O}(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$.

(b) $f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in \mathrm{O}(g(n))$ y $f(n) \in \Omega(g(n))$.

(c) $f(n) \in \mathrm{O}(h(n))$ y $g(n) \in \mathrm{O}(h(n)) \Rightarrow (f+g)(n) \in \mathrm{O}(h(n))$.

(d) $f(n) \in \mathrm{O}(h(n))$ y $g(n) \in \mathrm{O}(l(n)) \Rightarrow (f \cdot g)(n) \in \mathrm{O}(h(n)l(n))$.

(e) (reflexividad) $f(n) \in \mathrm{O}(f(n))$.

(f) (transitividad) $f(n) \in \mathrm{O}(g(n))$ y $g(n) \in \mathrm{O}(h(n)) \Rightarrow f(n) \in \mathrm{O}(h(n))$.

Ejercicio 3 ♣

Verificar que:

(a) $1000000 \in \mathrm{O}(1)$

(b) $10n^2 \in \mathrm{O}(n^3)$

(c) $2^n \in \mathrm{O}(n^n)$

(d) $n! \in \mathrm{O}(n^n)$

(e) $\sum_{k=0}^{n} k \in \mathrm{O}(n^2)$

(f) $\sum_{k=0}^{n} k^2 \in \mathrm{O}(n^3)$

(g) $\sum_{k=0}^{n} k^3 \in \mathrm{O}(n^4)$

(h) Si $p(n) = \sum_{k=0}^{m} a_k\, n^k$ ($a_k \in \mathbb{R}$), entonces $p(n) \in \mathrm{O}(n^m)$

Ejercicio 4 ♣

Analizar si cada una de las siguientes afirmaciones es verdadera o falsa, argumentando apropiadamente las respuestas dadas:

(a) $2^{n+1} \in \mathrm{O}(2^n)$

(b) $2^{2n} \in \mathrm{O}(2^n)$

(c) $\mathrm{O}(n^2) \cap \Omega(n) = \Theta(n^2) \cup \Theta(n \log n) \cup \Theta(n)$

(d) $\Omega(n^{\frac{3}{2}}) \cap \mathrm{O}(n) = \Theta(n^{\frac{3}{2}}) \cap \Theta(n)$

(e) $\mathrm{O}(1) \subseteq \mathrm{O}(1/n)$

(f) Si $f(n) \in \mathrm{O}(g(n))$, entonces $\log f(n) \in \mathrm{O}(\log g(n))$

(g) Si $f(n) \in \Theta(g(n))$, entonces $2^f(n) \in \Theta(2^g(n))$

Ejercicio 5

Probar que

$$\sum_{k=0}^{n} a^k \in \mathrm{O}(1)$$

siendo $0 \leq a < 1$ y $n \geq 0$.

Ejercicio 6

Dadas dos clases de complejidad $\mathrm{O}(f)$ y $\mathrm{O}(g)$, decimos que $\mathrm{O}(f) \leq \mathrm{O}(g)$ si y sólo si para toda función $h \in \mathrm{O}(f)$ sucede que $h \in \mathrm{O}(g)$.

(a) ¿Qué significa, intuitivamente, $\mathrm{O}(f) \leq \mathrm{O}(g)$? ¿Qué se puede concluir cuando, simultáneamente, tenemos $\mathrm{O}(f) \leq \mathrm{O}(g)$ y $\mathrm{O}(g) \leq \mathrm{O}(f)$?

(b) ¿Cómo ordena $\leq$ las siguientes clases de complejidad?

• $O(1)$	• $O(x+1)$	• $O(x^2)$
• $O(\sqrt{x})$	• $O(1/x)$	• $O(x^x)$
• $O(\sqrt{2})$	• $O(\log x)$	• $O(\log^2 x)$
• $O(\log x^2)$	• $O(\log \log x)$	• $O(x!)$
• $O(\log x!)$	• $O(2^x)$	• $O(x \log x)$

Ejercicio 7

Para cada par de funciones $f(n)$ y $g(n)$, indicar si $f(n) = \mathrm{O}(g(n))$ y si $g(n) = \mathrm{O}(f(n))$.

(a) $f = 10n, g = n^2 - 10n$

(b) $f = n^3, g = n^2 \log n$

(c) $f = n \log n, g = n + \log n$

(d) $f = \log n, g = \sqrt[k]{n}$

(e) $f = \ln n, g = \log n$

(f) $f = \log(n+1), g = \log n$

(g) $f = \log\log n, g = \log n$

(h) $f = 2^n, g = 10^n$

(i) $f = n^m, g = m^n$

(j) $f = \cos(n \cdot \pi/2), g = \sin(n \cdot \pi/2)$

(k) $f = n^2, g = (n \cos n)^2$

Ejercicio 8 ♣

Indicar -respondiendo sí/no- para cada par de expresiones (A, B) de la tabla, si A es O, Ω, o Θ de B. Suponer que $k \geq 1$, $\epsilon > 0$, y $c > 1$ son constantes.

A	B	O	Ω	Θ
$\lg^k n$	n^ϵ			
n^k	c^n			
$\sqrt{n}$	$n^{\sin n}$			
2^n	$2^{n/2}$			
$n^{\lg c}$	$c^{\lg n}$			
$\lg n$	$\lg n^n$			

Ejercicio 9

Dar una expansión asintótica de

$$f(n) = n\left(\sqrt[n]{a} - 1\right)$$

en términos de $O(n^{-3})$, siendo $a > 0$.

Ejercicio 10

Es posible extender las notaciones asintóticas para el caso de dos parámetros n y m creciendo independientemente, a distintas velocidades. Para una dada función $g(n, m)$, entendemos por $O(g(n, m))$ al conjunto de funciones

$$O(g(n,m)) = \{f(n,m) : \begin{array}{l}\text{existen constantes positivas } c,\ n_0 \text{ y } m_0 \\ \text{tales que } 0 \leq f(n,m) \leq c\, g(n,m) \\ \text{para todo } n \geq n_0 \text{ y } m \geq m_0\}.\end{array}$$

Usando esto, dar las definiciones correspondientes para las clases $\Omega(g(n, m))$ y $\Theta(g(n, m))$.

Ejercicio 11 ♣

Explicar el error en el siguiente resultado:

$$O(f(n)) - O(f(n)) = 0$$

Además, ¿cómo debería ser realmente el miembro derecho de la igualdad anterior?

2. Análisis de algoritmos iterativos sencillos

Ejercicio 12 ♣

Determinar una expresión Θ para el tiempo de corrida de peor caso de cada uno de los siguientes fragmentos de código.

(a)
```
f(n, 10, 0);
g(n, m, k);
h(n, m, 1000000);
```

(b)
```
for (int i = 0; i < n; ++i)
        f(n, m, k);
```

(c)
```
for (int i = 0; i < e(n, 10, 100); ++i)
        f(n, 10, 0);
```

(d)
```
for (int i = 0; i < e(n, m, k); ++i)
        f(n, 10, 0);
```

(e)
```
for (int i = 0; i < n; ++i)
for (int j = i; j < n; ++j)
        f(n, m, k);
```

Suponer que n, m y k son de tipo entero, y que las funciones e, f, g y h tienen las siguientes características:

- e(n, m, k) es O(1) y devuelve valores entre 1 y (n + m + k);
- f(n, m, k) es O$(n + m)$;
- g(n, m, k) es O$(m + k)$;
- h(n, m, k) es O$(n + k)$.

Ejercicio 13 ♣

Para cada uno de las siguientes funciones C++, averiguar qué es lo que calculan. Expresar la respuesta como función de n. Expresar el tiempo de corrida de peor caso en notación Θ.

(a)
```
int f(int n)
{
        int sum = 0;
        for (int i = 1; i <= n; ++i)
                sum += i;
        return sum;
}
```

(b)
```
int g(int n)
{
        int sum = 0;
        for (int i = 1; i < n; ++i)
                sum += i + f(n);
        return sum;
}
```

(c)
```
int h(int n)
{
        return f(n) + g(n);
}
```

Ejercicio 14 ♣

Para cada uno de los siguientes fragmentos de programa, encontrar la complejidad temporal expresada en notación Θ.

(a)
```
for (i = n; i >= 1; i /= 2)
        cout << "Awesome\n";
```

(b)
```
for (i = 1; i < n; i *= 2)
        for (j = 1; j <= i; ++j)
                cout << "Homework\n";
```

(c)
```
for (i = 1; i * i <= n; ++i)
        for (j = i; j >= 1; --j)
                cout << "Assignment\n";
```

3. Recurrencias

Ejercicio 15

Resolver las siguientes recurrencias:

(a) $R(n) = R(n-1) + 1;\ R(0) = 1$

(b) $R(n) = R(n-a) + 1;\ R(i) = 1, i \leq a$

(c) $R(n) = 2\,R(n-1) + 1;\ R(0) = 1$

(d) $R(n) = 2\,R(n-1) + n;\ R(0) = 1$

(e) $R(n) = 2\,R(\frac{n}{2}) + 1;\ R(1) = 1$

Ejercicio 16 ♣

Sin utilizar el Teorema Maestro, demostrar que la solución de

$$T(n) = T(n/2) + 1$$

es una $\mathrm{O}(\log n)$.

Ejercicio 17 ♣

Verificar que la solución de

$$T(n) = 2T(\sqrt{n}) + \log n$$

es $T(n) = \mathrm{O}(\log n \log\log n)$.

Hint: usar cambios de variable.

Ejercicio 18 ♣

Usando un árbol de recursión como ayuda, encontrar la solución de

$$T(n) = 3T(n/2) + n$$

Verificar la solución encontrada usando el método de sustitución.

Ejercicio 19

Supongamos estar ante una situación en la que tenemos que elegir uno de entre tres posibles algoritmos para resolver un problema de tamaño n:

(a) El primer algoritmo resuelve nuestro problema dividiéndolo en cinco subproblemas, cada uno de la mitad de tamaño, para luego resolverlos en forma recursiva y combinarlos en tiempo lineal.

(b) La segunda posibilidad consiste en encontrar la solución al problema de tamano n resolviendo recursivamente dos problemas de tamaño $n - 1$, y luego combinando las soluciones parciales en tiempo constante.

(c) El tercer algoritmo permite resolver el problema dividiéndolo en nueve subproblemas de tamaño $n/3$, para luego resolverlos recursivamente, y combinándolos en tiempo cuadrático a fin de obtener la respuesta buscada.

Para cada uno de los ítems anteriores, derivar una expresión asintótica que permita caracterizar la complejidad temporal de cada algoritmo. ¿Cuál de éstos es conveniente elegir?

Ejercicio 20 ♣

Determinar si cada una de las siguientes afirmaciones es verdadera o falsa, argumentando detalladamente las respuestas dadas:

(a) La solución de la recurrencia $T(n) = 2T(n/2) + \log n$ es una $\Theta(n \log n)$.

(b) Existen valores de α ($0 < \alpha < 1$) para los cuales la solución de la recurrencia

$$T(n) = 2T(\alpha n) + n$$

es $\Theta(n)$.

(c) El Teorema Maestro permite encontrar la solución asintótica de cualquier recurrencia que represente la complejidad temporal de un algoritmo que use la técnica de dividir y conquistar.

(d) Sea $A[1 \ldots n]$ un arreglo. Si en Quicksort la búsqueda del pivote toma tiempo $P(n)$, y dicho pivote separa los elementos de A en una proporción p ($0 \leq p \leq 1$), entonces la recurrencia para representar la complejidad temporal de dicho algoritmo viene dada por

$$T(n) = T(pn) + T((1 - p)n) + P(n) + 1$$