Práctica 5

Complejidad Algorítmica

Introducción a la Computación

1^{er} cuatrimestre 2019

Ejercicio 1. Sean $T_1(n) = 10\,n^2$, $T_2(n) = 1000 \log\log n$, $T_3(n) = 3^n/100$, $T_4(n) = 50\,n \log n$, y $T_5(n) = n^3$ los tiempos de ejecución de cinco programas que respetan la misma especificación. Graficar n vs. $T_i(n)$ para $i = 1 \ldots 5$. ¿Cuál programa es preferible si se sabe de antemano que siempre $n \leq 10$? ¿Y si $n \leq 20$? ¿Y si no se sabe nada acerca del tamaño de la entrada?

Ejercicio 2. Usamos la notación $O(\cdot)$ para expresar la cota superior del tiempo de ejecución de un programa. Sea $T(n) = n^2 - 4n - 2$ el tiempo de ejecución de un programa. Demostrar que $T(n) \in O(n^2)$ y también que $T(n) \in O(n^3)$.

Ejercicio 3. Demostrar que para todo $k \geq 0$ y toda función de complejidad temporal $T(n)$, si $T(n) \in O(n^k)$, entonces $T(n) \in O(n^{k+1})$.

Ejercicio 4. Dadas dos clases de complejidad $O(f)$ y $O(g)$, decimos que $O(f) \leq O(g)$ si y sólo si para toda función de complejidad $T \in O(f)$ sucede que $T \in O(g)$.

1. ¿Qué significa, intuitivamente, $O(f) \leq O(g)$? ¿Qué se puede concluir cuando, simultáneamente, tenemos $O(f) \leq O(g)$ y $O(g) \leq O(f)$?

2. ¿Cómo ordena $\leq$ las siguientes clases de complejidad?

- $O(1)$
- $O(\sqrt{n})$
- $O(\sqrt{2})$
- $O(\log n^2)$
- $O(\log n!)$
- $O(n+1)$
- $O(1/n)$
- $O(\log n)$
- $O(\log\log n)$
- $O(2^n)$
- $O(n^2)$
- $O(n^n)$
- $O(\log^2 n)$
- $O(n!)$
- $O(n \log n)$

Ejercicio 5. Calcular el tiempo de ejecución (en cantidad de operaciones) y el orden de complejidad de los algoritmos realizados para los ejercicios 1, 2 y 3 de la práctica 2.

Ejercicio 6. Adaptar los algoritmos de búsqueda vistos en clase para:

(a) Escribir un algoritmo que, dados una lista A y un entero x, retorne todos los elementos de A menores o iguales que x. Calcular la complejidad temporal del algoritmo.

(b) Escribir un algoritmo que realice lo mismo que el del punto (a), pero sabiendo que la lista está ordenada. Calcular la complejidad temporal del algoritmo.

(c) Escribir un algoritmo que dados una lista ordenada A y dos enteros x, y, retorne todos los elementos de A mayores o iguales que x y menores o iguales que y. Calcular la complejidad temporal del algoritmo.

Ejercicio 7. Demostrar que los siguientes programas tienen el orden de complejidad temporal de peor caso indicado, suponiendo que todas las operaciones sobre listas son de $O(1)$:

a) SUMA, que calcula la suma de una lista de enteros. $T_{\text{Suma}}(|A|) \in O(|A|)$

```
SUMA: A ∈ ℤ[] → ℤ
    RV ← 0
    i ← 0
    while (i < |A|) {
        RV ← RV + A[i]
        i ← i + 1
    }
```

b) INSERTIONSORT, que ordena la lista pasada como parámetro. $T_{\text{InsertionSort}}(|A|) \in O(|A|^2)$

```
INSERTIONSORT: A ∈ ℤ[] → ∅. Modifica A.
    i ← 1
    while (i < |A|) {
        valor ← A[i]
        j ← i − 1
        while (j ≥ 0 ∧ A[j] > valor) {
            A[j + 1] ← A[j]
            j ← j − 1
        }
        A[j + 1] ← valor
        i ← i + 1
    }
```

c) SUMAREC, que calcula la suma de una lista de enteros, ahora de manera recursiva. $T_{\text{SumaRec}}(|A|) \in O(|A|)$

```
SUMAREC: A ∈ ℤ[] → ℤ
    RV ← SumaRecAux(A, 0)

SUMARECAUX: A ∈ ℤ[] × i ∈ ℤ → ℤ
    RV ← 0
    if (i < |A|) {
      RV ← SumaRecAux(A, i + 1) + A[i]
    }
```

d) FIBREC, que calcula el n-ésimo término de la sucesión de Fibonacci (1, 1, 2, 3, 5, 8, 13, 21, 34, ...) de manera recursiva. $T_{\text{FibRec}}(n) \in O(2^n)$

```
FIBREC: n ∈ ℤ → ℤ
    if (n ≤ 2) {
      RV ← 1
    } else {
      RV ← FibRec(n − 1) + FibRec(n − 2)
    }
```

e) FIB, que calcula el n-ésimo término de la sucesión de Fibonacci, ahora de manera no recursiva. $T_{\text{Fib}}(n) \in O(n)$

FIB: $n \in \mathbb{Z} \to \mathbb{Z}$
    $fibPrevio \leftarrow 0$
    $fibActual \leftarrow 1$
    if ($n \leq 2$) {
        $RV \leftarrow 1$
    } else {
        while ($n > 1$) {
            $fibNuevo \leftarrow fibPrevio + fibActual$
            $fibPrevio \leftarrow fibActual$
            $fibActual \leftarrow fibNuevo$
            $n \leftarrow n - 1$
        }
        $RV \leftarrow fibActual$
    }

¿Cuál de estas implementaciones de Fibonacci es más eficiente para valores de n arbitrariamente grandes?

Ejercicio 8. Demostrar que el orden de complejidad temporal de peor caso de las funciones SUMA y SUMAREC de los ejercicios 5.a y 5.d, respectivamente, es cuadrático, si se supone que las operaciones de asignación y consulta sobre listas son $O(n)$, donde n es la cantidad de elementos de la lista.

Ejercicio 9. Escribir un algoritmo que resuelva cada uno de los siguientes problemas con el orden de complejidad temporal indicado. Demostrar que el algoritmo hallado tiene el orden indicado.

1. Calcular la media de una lista de enteros. $O(|A|)$
2. Calcular la mediana de una lista de enteros. $O(|A|^2)$
3. Determinar, dado un n natural, si n es (o no) primo. $O(\sqrt{n})$
4. Dado un n natural, obtener su representación binaria (como una secuencia de unos y ceros). $O(\log n)$
5. Dado un arreglo de unos y ceros representando un número en base binaria, obtener el número correspondiente en base decimal. $O(|A|)$

¿Considera que en alguno de los casos anteriores la complejidad del algoritmo propuesto es *óptima* (en términos de orden de complejidad, ignorando constantes)?