Pontificia Universidad Católica de Chile
Departamento de Ciencia de la Computación
IIC1253 - Matemáticas Discretas

Tarea 5

04 de noviembre de 2024

2º semestre 2024 - Profesores P. Bahamondes - D. Bustamante - M. Romero

---

Requisitos

- La tarea es individual. Los casos de copia serán sancionados con la reprobación del curso con nota 1,1.
- **Entrega**: Hasta las 23:59 del 13 de noviembre a través del buzón habilitado en el sitio del curso (Canvas).
  - Esta tarea debe ser hecha completamente en LaTeX. Tareas hechas a mano o en otro procesador de texto **no serán corregidas**.
  - Debe usar el template LaTeX publicado en la página del curso.
  - Cada solución de cada problema debe comenzar en una nueva hoja. ***Hint:*** Utilice `\newpage`
  - Los archivos que debe entregar son el archivo `PDF` correspondiente a su solución con nombre `numalumno.pdf`, junto con un `zip` con nombre `numalumno.zip`, conteniendo el archivo `numalumno.tex` que compila su tarea. Si su código hace referencia a otros archivos, debe incluirlos también.
- El no cumplimiento de alguna de las reglas se penalizará con un descuento de 0.5 en la nota final (acumulables).
- No se aceptarán tareas atrasadas (salvo que utilice su cupón `#problemaexcepcional`).
- Si tiene alguna duda, el foro de Github (issues) es el lugar oficial para realizarla.

Pregunta 1

(a) (1.0 pts) Sean $f, g : \mathbb{N} \to \mathbb{R}^+$ funciones arbitrarias. Demuestre que $f+g \in O(\text{máx}\{f, g\})$.

(b) (2.0 pts) Considere la siguiente recurrencia:

$$T(n) = \begin{cases} 1 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + 1 & n \geq 2 \end{cases}$$

Demuestre usando inducción que $T(n) \in O(n)$.

(c) (3.0 pts) Considere la recurrencia vista en clases para MergeSort:

$$T(n) = \begin{cases} 1 & n = 1 \\ T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n & n \geq 2 \end{cases}$$

Demuestre usando inducción que $T(n) \in O(n \log n)$.

Hint: Para la parte (b) y (c), las siguientes propiedades pueden ser útiles:

$$n = \left\lfloor \frac{n}{2} \right\rfloor + \left\lceil \frac{n}{2} \right\rceil \qquad \left\lfloor \frac{n}{2} \right\rfloor \leq \frac{n}{2} \qquad \left\lceil \frac{n}{2} \right\rceil \leq \frac{n+1}{2}$$

Para la parte (b), se recomienda probar algo más fuerte[1] que $T(n) \leq c \cdot n$. Por ejemplo, algo de la forma $T(n) \leq g(n)$ donde $g(n) < c \cdot n$ y g es una función que usted debe escoger. Lo mismo para la parte (c).

[1] Puede parecer contraintuitivo que probar por inducción algo más fuerte nos ayude. Esto es una estrategia bastante común. La ventaja es que al probar algo más fuerte por inducción, la hipótesis inductiva también se hace más fuerte, y luego la demostración de la tesis inductiva puede resultar más simple.

Pregunta 2

(a) (2.0 pts) Considere la siguiente recurrencia:

$$T(n) = \begin{cases} 1 & n = 1 \\ T(n-1) + n & n \geq 2 \end{cases}$$

Encuentre una notación Θ para $T(n)$.

(b) (2.0 pts) Considere el siguiente algoritmo para ordenar una lista:

input : Arreglo $A[0, \ldots, n-1]$, largo n
output: Arreglo ordenado

`QuickSort`(A, n)**:**

```
1   if n ≤ 1 then
2       return A
3   else
4       q ← A[0]
5       A_1, A_2 ← Particionar(A[1, ..., n − 1], q)
6       if A_1 ≠ ∅ then
7           B_1 ← QuickSort(A_1, largo(A_1))
8       if A_2 ≠ ∅ then
9           B_2 ← QuickSort(A_2, largo(A_2))
10      return [B_1, q, B_2]
```

En el algoritmo, la función *largo* retorna el largo de una lista, y la función *Particionar* se define como sigue:

input : Arreglo A, natural q
output: Arreglos A_1, A_2

`Particionar`(A, q)**:**

```
1   A_1 ← ∅
2   A_2 ← ∅
3   for i ∈ {0, ..., largo(A) − 1} do
4       if A[i] ≤ q then
5           A_1 ← [A_1, A[i]]
6       else
7           A_2 ← [A_2, A[i]]
8   return A_1, A_2
```

Encuentre una notación Θ para la complejidad de peor caso $T(n)$ de QuickSort.

(c) Encuentre una notación Θ para la complejidad de mejor caso $T(n)$ de QuickSort.

Hint: Al analizar QuickSort puede enfocarse en la comparación de la línea 1 de QuickSort, y en la comparación de la línea 4 de Particionar. Puede asumir que el peor caso para QuickSort es cuando la partición que entrega *Particionar* es lo más "desbalanceada" posible, y el mejor caso es cuando es lo más "balanceada" posible.