Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignaturas: Diseño de Algoritmos - Algoritmos II
Primer Cuatrimestre de 2025

# Guía Práctica No. 2: Divide & Conquer / Decrease & Conquer

Esta guía práctica abarca las técnicas de diseño Divide & Conquer y Decrease & Conquer. Refiere por lo tanto a los capítulos 4 y 5 de *Introduction to the Design and Analysis of Algorithms* (Levitin 2003), que son lectura recomendada para el desarrollo de la práctica.

Parte de esta práctica cuenta con esquemas de programación y tests de asistencia a su resolución. Éstos pueden accederse a partir del classroom de github de la materia.

Esta práctica no tiene entrega formal. Su resolución es opcional.

1. Dado un arreglo `A[0..n-1]` de números reales distintos, llamaremos *inversión* a un par de valores `(A[i],A[j])` almacenados en el arreglo tales que `A[i]` $>$ `A[j]`, con `i` $<$ `j`. Diseñe, utilizando Divide & Conquer, un algoritmo que calcule el número de inversiones en un arreglo dado. Calcule el tiempo de ejecución en el peor caso de su algoritmo, e impleméntelo en Java. Por ejemplo, para el arreglo $[3, 1, 5, 2]$ la cantidad de inversiones que se realizan es 3 y para el arreglo $[4, 1, 3, 2]$ la cantidad de inversiones que se realizan es 4.

2. A cuáles de estas ecuaciones de recurrencia es posible aplicar el Teorema Maestro?

   - $T(n) = 4T(n/2) + n$
   - $T(n) = 4T(n/2) + n^2$
   - $T(n) = 4T(n/2) + n^3$
   - $T(n) = 4T(n/2) + 2^n$
   - $T(n) = 4T(n-2) + n$
   - $T(n) = 4T(n-1) + n^2$
   - $T(n) = 4T(n-4) + 1$

3. En los casos en que sea posible, aplicar el Teorema Maestro, para decidir cuál es la tasa de crecimiento de las siguientes ecuaciones de recurrencia:

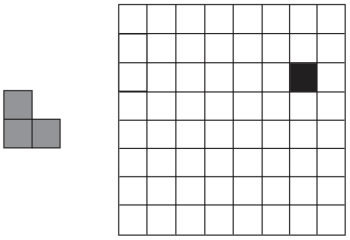
   - $T(1) = 1; T(n) = 4T(\frac{n}{2}) + n$
   - $T(1) = 1; T(n) = 4T(\frac{n}{2}) + n^2$
   - $T(1) = 1; T(n) = T(\frac{n}{2}) + 1$
   - $T(1) = 1; T(n) = 2T(n-1) + 2n$
   - $T(1) = 1; T(n) = T(\frac{n}{2}) + 2^n$
   - $T(1) = 1; T(n) = 2T(\frac{n}{2}) + \log n$

4. Diseñe usando Divide & Conquer e implemente en Java un algoritmo para resolver el problema de encontrar la subsecuencia de suma mínima, de una secuencia dada. Realice el análisis de tiempo de ejecución para el peor caso de su algoritmo. En caso de tener complejidad superior a $O(n \times \log n)$, optimice su algoritmo para conseguir tal eficiencia.

5. Diseñe usando Divide & Conquer e implemente en Haskell un programa que, dadas dos secuencias de caracteres, construya la subsecuencia común a ambas de longitud máxima. Se entiende por *subsecuencia* una cadena de caracteres que se deriva de la original mediante la eliminación de caracteres pero sin cambiar el orden de los caracteres en la secuencia original.

6. Implemente en Java o Haskell el algoritmo para resolver el problema de, dado un conjunto de puntos del plano, encontrar el par de puntos (distintos) cuya distancia es la menor entre todos los puntos del plano, en $O(n \log n)$, discutido en *Introduction to the Design and Analysis of Algorithms* (Levitin).

7. Un trominó (más precisamente, un trominó recto) es una pieza en forma de L formada por tres casillas de $1 \times 1$. El problema consiste en cubrir cualquier tablero de ajedrez de $2^n \times 2^n$ con una casilla faltante con trominós. Los trominós pueden orientarse de forma arbitraria, pero deben cubrir todas las casillas del tablero excepto la casilla faltante, de forma exacta y sin solapamientos. Diseñe usando Divide & Conquer e implemente un algoritmo en haskell para resolve reste problema.



8. Diseñe usando Divide & Conquer e implemente en Java un algoritmo para resolver el problema de dibujar la silueta de los edificios de una ciudad (skyline). Dada la ubicación y forma de $n$ edificios rectangulares en 2-dimensiones, computar la silueta de los edificios eliminando las líneas superpuestas.

   Para el skyline que se presenta en la figura 1

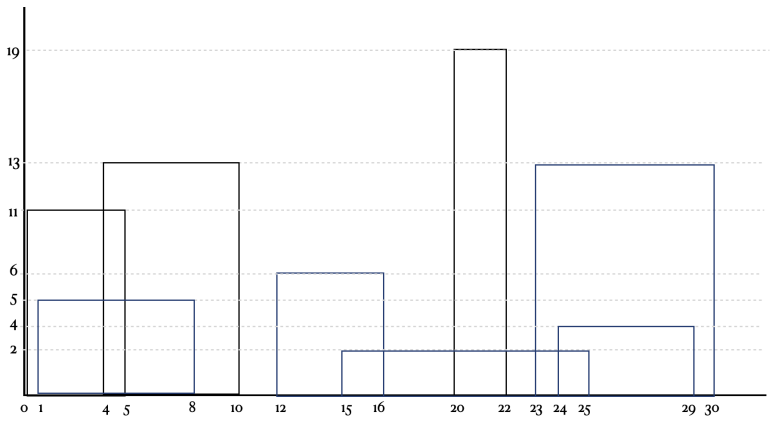



Figure 1: Ejemplo: Skyline

La silueta queda representada por la figura 2 que se muestra debajo. Notar que la silueta del skyline se define únicamente por la lista de puntos claves marcados en la figura. Como referencia, el test *test_example()* implementa este escenario.

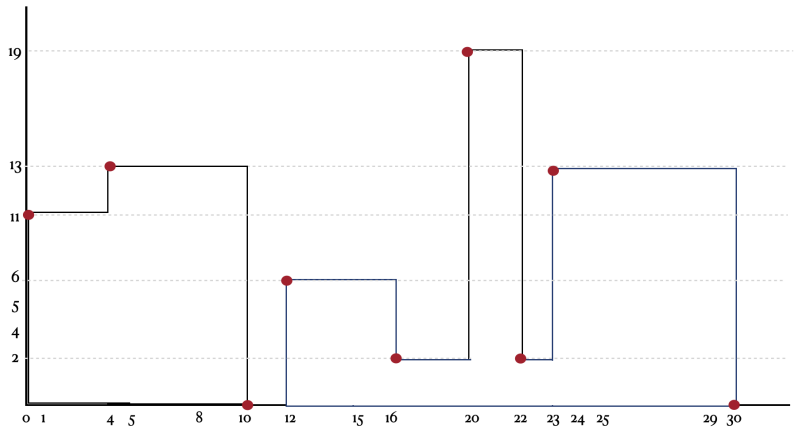



Figure 2: Ejemplo: Skyline (Silueta)

9. Utilizando la técnica Decrease & Conquer, diseñe un algoritmo para encontrar los elementos mayor y menor de una secuencia de $n$ enteros positivos. Implemente su algoritmo en Haskell.

10. Implemente en Java soluciones Decrease & Conquer con decremento por una constante y por un factor constante para el problema de multiplicar dos números enteros. Calcule además el número de sumas efectuadas por su algoritmo, en función del tamaño de la entrada.

    En relación a una de sus soluciones, considera que es más económico computar $n + n$ o $2 \times n$? Justifique su respuesta.

11. Utilizando la técnica Decrease & Conquer, diseñe un algoritmo para encontrar el índice del mayor elemento de una secuencia de $n$ enteros positivos. Piense en una alternativa a este algoritmo diseñado utilizando Fuerza Bruta, y compare implementaciones para estos dos algoritmos en Haskell.

    Realice además el análisis correspondiente para calcular cuántas comparaciones de elementos son realizadas por ambos algoritmos en el peor caso.

12. Utilizando la técnica Decrease & Conquer, diseñe un algoritmo para encontrar la distancia Hamming entre dos cadenas de igual longitud.

13. Diseñe e implemente en Haskell un programa que resuelva el problema de, dada un par de secuencias $p$ y $t$, determine si p es subsecuencia (de elementos contiguos) de t.