Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignaturas: Diseño de Algoritmos - Algoritmos II
Primer Cuatrimestre de 2023

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

Esta guía práctica abarca las técnicas de diseño Decrease & Conquer y Divide & 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 Diseno-de-Algoritmos-Algoritmos-II-2023.

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.

2. Aplicando el Teorema Maestro, decida cuál es la tasa de crecimiento de las siguientes ecuaciones de recurrencia:

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

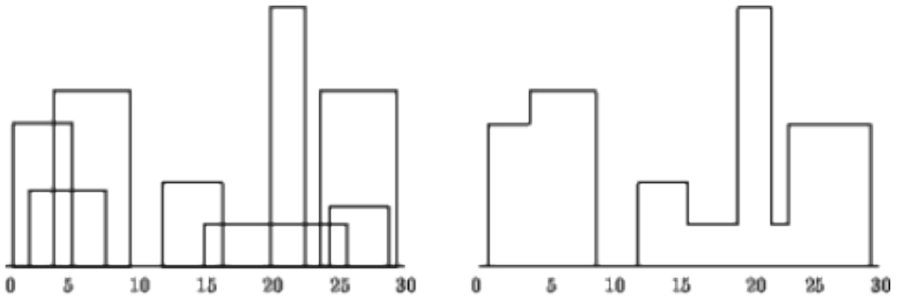
3. A cuáles de estas ecuaciones de recurrencia es posible aplicar el Teorema Maestro?

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

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. Archivo: SumaMin.java

5. Diseñe usando Divide & Conquer e implemente en Java y 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. 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.



7. 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). Explique además las razones por las cuales no es necesario comparar *todos* los pares de puntos en las inmediaciones de la línea media, sino que para cada punto a uno de los lados de esta línea, sólo hace falta considerar a lo sumo 6 puntos del otro lado. pag 140

8. 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. Archivo: Max&Min.hs

9. 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. maxIndex -> src -> MaxIndex.hs

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

11. 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. Archivo: MultEnteros.hs

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

12. Diseñe usando Decrease & Conquer e implemente Haskell un programa que resuelva el problema de, dada un par de secuencias p y t, determinar si p es subsecuencia (de elementos contiguos) de t. Archivo: subSeq.hs

13. Diseñe usando Decrease & Conquer e implemente en Java o Haskell un programa que, dada una lista de asignaturas de un plan de estudios y un conjunto de correlatividades de cursado (cada una de las cuales indica un pre-requisito para el cursado de una materia), determine si el plan es *consistente*, es decir, si el mismo no contiene correlatividades cíclicas. Su algoritmo no sólo debe emitir el *veredicto* (consistente o inconsistente), sino que en caso de consistencia debe dar un ejemplo, es decir una forma de cursar asignaturas en cuatrimestres consecutivos de manera tal de respetar las correlatividades.

14. Encontrar el máximo entre un conjunto de nros usando Divide & Conquer -> Archivo: MaxElem.hs