Metodología y tecnología de la programación

Ingeniería Informática (UCM)

Hoja de ejercicios 3 — Curso 2007/2008

Ejercicios de divide y vencerás

Ejercicio 1 Desarrollar un algoritmo para multiplicar n números complejos usando solo $3(n-1)$ multiplicaciones de números reales.

Ejercicio 2 Para cada número real $\alpha > 1$ existe un algoritmo A_α que puede multiplicar dos números enteros de n cifras en tiempo $O(n^\alpha)$. Sea entonces el siguiente algoritmo:

fun supermul(**e** u, v : *ent*) **dev** m
{ suponemos que u y v tienen el mismo tamaño }
$n :=$ tamaño de u y v
$\alpha := 1 + (\log\log n)/\log n$
$m := A_\alpha(u, v)$
ffun

A primera vista, este algoritmo parece multiplicar dos números enteros de n cifras en tiempo $O(n \log n)$ puesto que $n^\alpha = n \log n$ cuando $\alpha = 1 + (\log\log n)/\log n$. Encontrar al menos dos errores fundamentales en este análisis de supermul.

Ejercicio 3 Dado un vector ordenado $T[1..n]$ de n enteros distintos, escribir un algoritmo que en tiempo $O(\log n)$ encuentre un entero i tal que $1 \le i \le n$ y $T[i] = i$, siempre que tal i exista.

Ejercicio 4 Dado un vector $T[1..n]$ de n elementos (que tal vez no se puedan ordenar), se dice que un elemento x es *mayoritario en* T cuando el número de veces que x aparece en T es estrictamente mayor que $n/2$. Escribir un algoritmo que en tiempo $O(n \log n)$ decida si un vector $T[1..n]$ contiene un elemento mayoritario y devuelva tal elemento cuando exista.

Ejercicio 5

1. Suponiendo que los elementos del vector $T[1..n]$ se puedan ordenar, escribir un algoritmo que en tiempo *lineal* decida si $T[1..n]$ contiene un elemento mayoritario y devueva tal elemento cuando exista.
2. Hacer lo mismo que en el apartado anterior pero sin suponer que los elementos se puedan ordenar.

Ejercicio 6 Si en el algoritmo *divide y vencerás* para multiplicar dos números binarios de tamaño n se utiliza la expresión $(A+B)(C+D)$ para reducir el número de llamadas recursivas de 4 a 3, existe el problema de que el tamaño de los números que se multiplican puede ser $n/2+1$ en vez de $n/2$. Explicar cómo reducir una multiplicación de tamaño $n/2+1$ a una de tamaño $n/2$ a base de hacer más sumas y desplazamientos de coste lineal.

Ejercicio 7

1. Dado un valor x fijo, escribir un algoritmo para calcular x^n con un coste $O(\log n)$ en términos del número de multiplicaciones.
2. Sea F la matriz $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. Calcular el producto del vector $(i \;\; j)$ y la matriz F. ¿Qué ocurre cuando i y j son números consecutivos de la sucesión de Fibonacci?
3. Utilizando las ideas de los dos apartados anteriores, desarrollar un algoritmo para calcular el n-ésimo número de Fibonacci $f(n)$ con un coste $O(\log n)$ en términos del número de operaciones aritméticas elementales.

Ejercicio 8 Diseñar un algoritmo para resolver el siguiente problema e implementarlo como procedimiento. Dado un árbol binario ordenado cuyos nodos contienen información de tipo entero, y dados dos enteros *k1* y *k2*, crear una lista ordenada que contenga los valores almacenados en el árbol que sean mayores o iguales que *k1* y menores o iguales que *k2*.

Ejercicio 9 Se quiere organizar un torneo en el que intervengan n competidores. Cada uno de ellos debe jugar exactamente una vez contra cada posible oponente; más aún, cada participante debe jugar exactamente un partido cada día, con la posible excepción de un único día. Escribir un algoritmo que, para cualquier entero $n > 1$, construya un calendario que permita terminar el torneo en $n - 1$ días si n es par, o en n días si n es impar.

Ejercicio 10 Sean $X[1..n]$ e $Y[1..n]$ dos vectores de enteros *ordenados* de forma no decreciente, con $n \geq 1$. Escribir un algoritmo de coste $O(\log n)$ para hallar la *mediana* del vector formado por los $2n$ elementos juntos, donde la mediana de un vector $V[i..j]$ se define como el elemento que ocupa la posición $(i + j)$ div 2 después de ordenar el vector $V[i..j]$.

Ejercicio 11 Dado un vector $T[1..n]$ de elementos que se pueden ordenar (números enteros, por ejemplo) y que pueden estar repetidos, se desean hallar los m elementos más pequeños, donde m es mucho más pequeño que n. ¿Qué es mejor,

1. ordenar $T[1..n]$ y después coger los m primeros elementos $T[1..m]$,
2. hacer m llamadas a *selección2*$(T[1..n], k)$ con $k = 1, 2, \ldots, m$, o
3. utilizar algún otro método?