Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2019

Práctico 1 - Parte 2

ejercicios para el lunes 18/3

1. (a) Ordená los arreglos del ejercicio 4 del práctico anterior utilizando el algoritmo de ordenación rápida.

   (b) En el caso del inciso a), dar la secuencia de llamadas al procedimiento quick_sort_rec con los valores correspondientes de sus argumentos.

2. Escribí una variante del procedimiento partition que en vez de tomar el primer elemento del segmento $a[izq, der]$ como pivot, elige el valor intermedio entre el primero, el último y el que se encuentra en medio del segmento. Es decir, si el primer valor es 4, el que se encuentra en el medio es 20 y el último es 10, el algoritmos deberá elegir como pivot al último.

3. Escribí un algoritmo que dado un arreglo a : **array**$[1..n]$ **of int** y un número natural $k \leq n$ devuelve el elemento de a que quedaría en la celda $a[k]$ si a estuviera ordenado. Está permitido realizar intercambios en a, pero no ordenarlo totalmente. La idea es explotar el hecho de que el procedimiento partition del quick_sort deja al pivot en su lugar correcto.

4. El procedimiento partition que se dio en clase separa un fragmento de arreglo principalmente en dos segmentos: menores o iguales al pivot por un lado y mayores o iguales al pivot por el otro. Modificá ese algoritmo para que separe en tres segmentos: los menores al pivot, los iguales al pivot y los mayores al pivot. En vez de devolver solamente la variable pivot, deberá devolver pivot_izq y pivot_der que informan al algoritmo quick_sort_rec las posiciones inicial y final del segmento de repeticiones del pivot. Modificá el algoritmo quick_sort_rec para adecuarlo al nuevo procedimiento partition.

ejercicios para el miércoles 20/3

5. (a) Ordená los arreglos del ejercicio 4 del práctico anterior utilizando el algoritmo de ordenación por intercalación.

   (b) En el caso del inciso a) del ejercicio 4, dar la secuencia de llamadas al procedimiento merge_sort_rec con los valores correspondientes de sus argumentos.

6. (a) Escribí el procedimiento "intercalar_cada" que recibe un arreglo a : **array**$[1..2^n]$ **of int** y un número natural i : **nat**; e intercala el segmento $a[1, 2^i]$ con $a[2^i + 1, 2 * 2^i]$, el segmento $a[2 * 2^i + 1, 3 * 2^i]$ con $a[3 * 2^i + 1, 4 * 2^i]$, etc. Cada uno de dichos segmentos se asumen ordenados. Por ejemplo, si el arreglo contiene los valores 3, 7, 1, 6, 1, 5, 3, 4 y se lo invoca con con $i = 1$ el algoritmo deberá devolver el arreglo 1, 3, 6, 7, 1, 3, 4, 5. Si se lo vuelve a invocar con este nuevo arreglo y con $i = 2$, devolverá 1, 1, 3, 3, 4, 5, 6, 7 que ya está completamente ordenado. El algoritmo asume que cada uno de estos segmentos está ordenado, y puede utilizar el procedimiento de intercalación dado en clase.

   (b) Utilizar el algoritmo "intercalar_cada" para escribir una versión iterativa del algoritmo de ordenación por intercalación. La idea es que en vez de utilizar recursión, invoca al algoritmo del inciso anterior sucesivamente con $i = 0, 1, 2, 3$, etc.

   (c) (Difícil) ¿Cómo modificarías el algoritmo para ordenar arreglos de cualquier longitud?