Algoritmos y Estructuras de Datos II - 1° cuatrimestre 2016

Práctico 1 - Parte 2

Objetivo: familiarizarse con algunos de los colores de la patria grande (ejercicios 1, 2 y 3), ejercitar los algoritmos de ordenación divide y vencerás (ordenación por intercalación y ordenación rápida) para comprender su funcionamiento (4 y 6) y algunas de sus variantes, como la versión iterativa (no recursiva) de la ordenación por intercalación (5) y ordenación rápida con variantes con el pivote (8 y 9). También se pretende comprender en general la técnica divide y vencerás que estos algoritmos utilizan (7, 10, 11 y 12) y a calcular el orden de complejidad de estos algoritmos recursivos.

Dificultad estimada: *Primer nivel de dificultad: 4, 6 y 10. Segundo: 1, 2, 3, 11 y 12. Tercero: 7 y 8. Cuarto: 9. Quinto: 5.*

1. Dé un procedimiento que, dado un arreglo [1..n] de enteros, reordene sus elementos de manera que todos los números impares ocupen posiciones contiguas. Notemos que el **qué** del procedimiento pedido se puede expresar en el formalismo de Algoritmos I como:

   Pre: $\{a = A\}$

   Pos: $\{perm.a.A \wedge todosimparesjuntos.a\}$

   donde perm y todosimparesjuntos se expresan como:

   $perm.a.b \equiv \langle \forall i : i \in Z : ocurs.a.i = ocurs.b.i \rangle$

   $todosimparesjuntos.xs \equiv \langle \exists as, bs, cs : xs = as +\!\!+ bs +\!\!+ cs : sonpares.(as +\!\!+ cs) \wedge sonimpares.bs \rangle$

2. *Problema de la bandera argentina* Dé un procedimiento que, dado un arreglo [1..n] con valores en $\{BLANCO, CELESTE\}$, lo reordene en tres segmentos de todos celestes, todos blancos y todos celestes intentando que los largos de los dos segmentos celestes sean lo más parecido posible.

   Pre: $\{a = A\}$

   Pos: $\{perm.a.A \wedge bandera.a\}$

   donde bandera se expresa como:

   $bandera.xs \equiv \langle \exists as, bs, cs : xs = as +\!\!+ bs +\!\!+ cs : soncelestes.(as +\!\!+ cs) \wedge sonblancos.bs \wedge |\#as - \#cs| \leq 1 \rangle$

3. (a) Dé una función que, dado un arreglo [1..n] de T y un elemento t de T, devuelva el número de veces que ocurre t en el arreglo dado.

   (b) *Problema de la bandera boliviana* Dé un procedimiento que, dado un arreglo [1..n] con valores en $\{ROJO, AMARILLO, VERDE\}$, lo reordene en tres segmentos de todos rojos, todos amarillos y todos verdes.

   (c) * ¿Cómo modificarías el algoritmo para ordenar arreglos cuyos elementos son naturales entre 1 y 10? ¿Cuál es el orden del algoritmo? Comparalo con el más eficiente algoritmo de ordenación que conozcas.

   (d) * Y si los naturales del arreglo fueran entre 1 y k, ¿qué orden tendría el algoritmo?

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

   (b) En el caso del inciso a) del ejercicio 5, dar la secuencia de llamadas al procedimiento merge_sort_rec con los valores correspondientes de sus argumentos.

5. (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) * ¿Cómo modificarías el algoritmo para ordenar arreglos de cualquier longitud?

6. (a) Ordená los arreglos del ejercicio 5 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.

7. 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 pivot del quick_sort deja al pivote en su lugar correcto.

8. Escribí una variante del procedimiento pivot que en vez de tomar el primer elemento del segmento $a[izq, der]$ como pivote, elige el valor intermedio entre el primero, el último y el que se encuentra en medio del segmento.

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

10. El siguiente algoritmo calcula el mínimo elemento de un arreglo a : **array**$[1..n]$ **of nat** mediante la técnica de programación *divide y vencerás*. Analizá la eficiencia de $minimo(1, n)$.

```
fun minimo(a : array[1..n] of nat, i, k : nat)  ret m : nat
    if i = k then m := a[i]
    else
        j := (i + k) div 2
        m := min(minimo(a, i, j), minimo(a,j+1,k))
    fi
end  fun
```

11. Dado un arreglo a : **array**$[1..N]$ **of nat** se define una *cima* de a como un valor k en el intervalo $1, \ldots, N$ tal que $a[1..k]$ está ordenado crecientemente y $a[k..N]$ está ordenado decrecientemente.

    (a) Escribí un algoritmo que determine si un arreglo dado tiene cima.

    (b) Escribí un algoritmo que encuentre la cima de un arreglo dado (asumiendo que efectivamente tiene una cima) utilizando una búsqueda secuencial, desde el comienzo del arreglo hacia el final.

    (c) Escribí un algoritmo que resuelva el mismo problema del inciso anterior utilizando la idea de *búsqueda binaria*.

    (d) Calculá y compará el orden de complejidad de ambos algoritmos.

12. Una secuencia de valores $x_1, ..., x_n$ se dice que tiene *orden cíclico* si existe un i con $1 \leq i \leq n$ tal que $x_i < x_{i+1} < \ldots < x_n < x_1 < \ldots < x_{i-1}$. Por ejemplo, la secuencia $5, 6, 7, 8, 9, 1, 2, 3, 4$ tiene orden cíclico (tomando $i = 6$).

    (a) Escribí un algoritmo que determine si un arreglo almacena una secuencia de valores que tiene orden cíclico o no.

    (b) Escribí un algoritmo que dado un arreglo a : **array**$[1..n]$ **of nat** que almacena una secuencia de valores que tiene orden cíclico, realice una búsqueda secuencial en el mismo para encontrar la posición del menor elemento de la secuencia (es decir, la posición i).

    (c) Escribí un algoritmo que resuelva el problema del inciso anterior utilizando la idea de *búsqueda binaria*.

    (d) Calculá y compará el orden de complejidad de ambos algoritmos.