Algoritmos y Estructuras de Datos II - 23 de abril de 2014

Primer Parcial

Alumno: .....

Siempre se debe explicar la solución, una respuesta correcta no es suficiente sino viene acompañada de una justificación que demuestre que la misma ha sido comprendida. Las explicaciones deben ser completas. En el ejercicio de implementación, debe utilizarse pseudo-código. La utilización de código c influirá negativamente.

1.

```
proc A(in/out a: array[1..n,1..n] of nat, in b,c: nat)
    var f: nat
    f:= a[b,c]
    a[b,c]:= a[c,b]
    a[c,b]:= f
end proc

proc C(in/out a: array[1..n,1..n] of nat)
    for i:= 1 to n do
        for j:= i to n do
            A(a,i,j)
        od
    od
end proc

proc E (in/out a: array[1..n] of nat)
    var b,c: nat
    for i:= n-1 downto 1 do
        b:= i
        do b < n ∧ a[b] > a[b+1] ⟶ c:= a[b+1]
                                   a[b+1]:= a[b]
                                   a[b]:= c
                                   b:= b+1
        od
    od
end proc

proc B(in/out a: array[1..n,1..n] of nat)
    for i:= 1 to n do
        for j:= i+1 to n do
            A(a,i,j)
        od
    od
end proc

proc D(in/out a: array[1..n,1..n] of nat)
    for i:= 1 to n do
        for j:= 1 to n do
            A(a,i,j)
        od
    od
end proc

fun F (a: array[1..n] of nat) ret b: array[1..3] of nat
    b[1]:= a[1]
    b[2]:= 1
    b[3]:= 1
    for i:= 2 to n do
        b[1]:= b[1] + a[i]
        if a[i] < a[b[2]] then b[2]:= i
        else if a[i] > a[b[3]] then b[3]:= i fi
        fi
    od
end fun
```

Para cada uno de los algoritmos precedentes, explicá con tus palabras con la mayor precisión posible:

a) Qué hace el algoritmo (por ejemplo: "calcula el factorial de su argumento").

b) Cómo lo hace (por ejemplo: "multiplica los números menores o iguales a su argumento, de menor a mayor").

c) Cuál es el orden del algoritmo (por ejemplo: "es lineal, o sea, de orden n, porque consiste de un único ciclo que se repite n veces y cuyo cuerpo es constante").

d) Explicá si existen mejor y peor caso diferentes y cuáles son (por ejemplo: "todos los casos realizan exactamente el mismo número de operaciones porque ..." o si no "el peor caso ocurre cuando el arreglo está ordenado porque ... y el algoritmo se vuelve cuadrático, ..., en cambio el mejor caso ocurre cuando ...").

2. *a*) Dado el siguiente algoritmo, planteá la recurrencia que indica la cantidad de asignaciones realizadas a la variable m en función de la entrada n:

```
fun f (n: nat) ret m: nat
    if n ≤ 2 then
        m := n
    else
        m := 3*f(n/2) + n
    fi
end
```

b) Resolvé la siguiente recurrencia: $t(n) = \begin{cases} 1 & \text{si } n \leq 2 \\ 4t(n/2) + n^2 & \text{si } n > 2 \end{cases}$

c) La recurrencia que obtuviste en el item a) ¿es la misma que resolviste en el item b)?. En caso contrario, resolvéla también.

3. Ordená las siguientes funciones según el orden creciente de sus $\mathcal{O}$, estableciendo claramente en cuáles casos vale el $=$ y en cuáles el $\subset$. Justificá utilizando la jerarquía y las propiedades demostradas en la teoría.

   a) $\log_2(3^{n!})$ *b*) $\log_2(n * (3^{n!})^2)$ *c*) $n * \log_2(3^{(n-1)!})$ *d*) $(\log_2(3^n))^n$

4. Casi todos los lenguajes tienen a los números naturales o a los enteros como tipo concreto. De todas formas, los números naturales también pueden surgir "naturalmente" al analizar un problema. Después de todo, los números naturales no son otra cosa que contadores con otras operaciones más (como la suma, la resta, la multiplicación, etc.), y ya hemos visto que los contadores pueden surgir "naturalmente".

   En este ejercicio, se pide que completes la especificación del TAD natural que se parece al TAD contador, pero tiene otras operaciones más. Para facilitar la comprensión, en lugar de llamar inicial e incrementar a sus constructores, los llamaremos cero y sucesor.

   TAD natural
   constructores
   cero : natural
   sucesor : natural → natural
   operaciones
   es_cero : . . .
   predecesor : . . . { pre: argumento ≠ cero }
   suma : . . .
   multiplicación : . . .
   exponenciación : . . . { pre: sus dos argumentos no pueden ser cero simultáneamente}
   ecuaciones
   . . .
   . . .
   . . .

5. A continuación se especifica el TAD colable:

   TAD colable
   constructores
   vacía : colable
   encolar : colable × elem → colable
   operaciones
   es_vacía : colable → booleano
   primero : colable → elem { pre: argumento ≠ vacía }
   último : colable → elem { pre: argumento ≠ vacía }
   decolar : colable → colable { pre: argumento ≠ vacía }
   colar : elem × colable → colable
   ecuaciones
   es_vacía(vacía) = verdadero
   es_vacía(encolar(q,e)) = falso
   primero(encolar(vacía,e)) = e
   primero(encolar(encolar(q,e'),e)) = primero(encolar(q,e'))
   último(encolar(q,e)) = e
   decolar(encolar(vacía,e)) = vacía
   decolar(encolar(encolar(q,e'),e)) = encolar(decolar(encolar(q,e')),e)
   colar(e,vacía) = encolar(vacía,e)
   colar(e,encolar(q,e')) = encolar(colar(e,q),e')

   a) Explicá en qué se diferencia del TAD cola.

   b) Implementá el TAD colable utilizando listas enlazadas circulares con puntero al último nodo. Utilizá el tipo

   ```
   type node = tuple
                 value: elem
                 next: pointer to node
               end
   type queue = pointer to node
   ```

   para implementar las 7 operaciones.