Algoritmos y Estructuras de Datos II - 24 de junio de 2013

Primer Parcial - Recuperatorio

Alumno: .....

Recuerde utilizar una hoja aparte para la solución de cada ejercicio. La cátedra le proveerá las hojas que solicite. Recuerde también explicar y justificar con claridad cada una de sus respuestas.

1. El siguiente algoritmo es el de ordenación rápida tal como se lo ha visto en clase.

```
proc quick_sort (in/out a: array[1..n] of T)
    quick_sort_rec(a,1,n)
end proc

proc quick_sort_rec (in/out a: array[1..n] of T, in izq,der: nat)
    var piv: nat
    if der > izq → pivot(a,izq,der,piv)
                   quick_sort_rec(a,izq,piv-1)
                   quick_sort_rec(a,piv+1,der)
    fi
end proc

proc pivot (in/out a: array[1..n] of elem, in izq, der: nat, out piv: nat)
    var i,j: nat
    piv:= izq
    i:= izq+1
    j:= der
    do i ≤ j → if a[i] ≤ a[piv] → i:= i+1
               a[j] > a[piv] → j:= j-1
               a[i] > a[piv] ∧ a[j] ≤ a[piv] → swap(a,i,j)
                                               i:= i+1
                                               j:= j-1
             fi
    od
    swap(a,piv,j)
    piv:= j
end proc
```

Se puede observar que el procedimiento pivot, finaliza dejando el pivote en la posición piv, por eso las dos llamadas recursivas de quick_sort_rec son hasta piv-1 (donde se encuentran elementos menores o iguales al pivote) y desde piv+1 (donde se encuentran elementos mayores al pivote) respectivamente.

Surge la idea de omitir dejar el pivote en la posición piv eliminando la antepenúltima línea del procedimiento pivot (la que dice swap(a,piv,j)) pero no la línea siguiente (piv:= j no se elimina). Dado que el pivote no ha quedado en la posición piv, corresponde reemplazar piv-1 por piv en la primera llamada recursiva en la definición del procedimiento quick_sort_rec de la siguiente manera:

```
proc quick_sort_rec (in/out a: array[1..n] of T, in izq,der: nat)
    var piv: nat
    if der > izq → pivot(a,izq,der,piv)
                   quick_sort_rec(a,izq,piv)
                   quick_sort_rec(a,piv+1,der)
    fi
end proc
```

Evaluar la conveniencia de este cambio. ¿Qué se gana y qué se pierde? Examine con atención y explique con claridad. Ayuda: no es un cambio insignificante.

2. Ordenar las siguientes funciones según el orden creciente de sus $\mathcal{O}$.

   a) $\log_2(n^n)$ *b)* n^n *c)* n^{2n} *d)* $(2n)^n$ *e)* $n^2 \log_2 n$

3. Dar la especificación completa del TAD pila.

4. *a*) Dado el siguiente algoritmo, plantear 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 = 0 then
        m := n
    else
        m := f(n-1) + n
    fi
end
```

b) Resolver la siguiente recurrencia:

$$t(n) = \begin{cases} n & \text{si } n = 0 \\ t(n-1) + n & \text{si } n > 0 \end{cases}$$

5. Implementar el TAD pila con la siguiente representación:

```
type node = tuple
                elems: array[1..N] of elem
                size: nat
                next: pointer to node
            end
type stack = pointer to node
```

La idea que se persigue es utilizar la representación basada en un arreglo, pero evitando que la pila pueda llenarse. En efecto, con esta representación, en caso de llenarse el arreglo, cuando se agregue un nuevo elemento (merced a la operación push) deberá agregarse un nuevo nodo y el nuevo elemento se alojará en la primera posición del arreglo que se encuentra en ese nuevo nodo.

De esta manera, una pila puede estar representada por una lista de nodos donde en cada nodo habrá entre 1 y N elementos de la pila. Observe que no tiene sentido que en un nodo haya 0 elementos ya que ese nodo, en ese caso, podría liberarse con la operación free. La pila vacía, por ejemplo, se representa por el puntero null.