Apellido y Nombre: email:

nota

1	2	3	4	5

Algoritmos y Estructuras de Datos II **Parcial 3** **23/6/2008**

1. El siguiente algoritmo es el de ordenación rápida (quicksort) dado en clase.

```
proc quickSort (in/out a: array[1..n] of elem, in izq: int, in der: int)
    var piv: int
    if der > izq → pivot(a,izq,med,piv)
                                  {todos los elementos en a[izq,piv] son ≤ que a[piv]}
                                {∧ todos los elementos en a(piv,der] son > que a[piv]}
                   quickSort(a,izq,piv-1)
                   quickSort(a,piv+1,der)
    fi
end
```

donde el procedimiento pivot es el que sigue

```
proc pivot (in/out a: array[1..n] of elem, in izq: int, in der: int, out piv: int)
    var i,j: int
    piv:= izq
    i:= izq+1
    j:= der
    do i ≤ j →   {piv < i ≤ j+1 ∧ todos los elementos en a[izq,i) son ≤ que a[piv]}
                                  {∧ todos los elementos en a(j,der] son > que a[piv]}
            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          {i = j+1, por eso todos los elementos en a[izq,j] son ≤ que a[piv]}
                                  {∧ todos los elementos en a[i,der] son > que a[piv]}
    swap(a,piv,j)                    {dejando el pivot en una posición más central}
    piv:= j                                {señalando la nueva posición del pivot}
end
```

Como se puede observar, el procedimiento elige como pivot el elemento que se encuentra en la primera celda del "segmento de arreglo" a[izq..der] (es decir, en la posición izq).

a) escribir una variante del algoritmo que selecciona como pivot el elemento que se encuentra en la última celda (es decir, en la posición der). Para resolver este ejercicio NO se permite colocar el pivot en la posición izq (por ejemplo, mediante swap(a,izq,der)) y luego reusar el procedimiento dado. Sí está permitido, si prefiere, escribir el procedimiento de otra manera (siempre y cuando use como pivot el elemento mencionado).

b) escribir otra variante del algoritmo que elige como pivot el valor intemedio entre a[izq], a[der] y a[(izq+der) ÷ 2]. En este caso sí se permite mover este elemento a una celda que le resulte conveniente. ¿Qué interés puede tener esta variante?

2. Dado un grafo dirigido con costos no negativos en las aristas, le interesa ir desde su casa (nodo x) hasta la casa de un amigo (nodo z) pero pasando a buscar a su novio/novia (nodo y). ¿Qué algoritmos vistos en clase podría utilizar para calcular el (costo del) camino de costo mínimo que realice este trayecto? Explique de qué manera los utilizaría para resolver este problema. Discuta las ventajas de una u otra posibilidad.

3. Se cuenta con una mochila de 173 kilogramos de capacidad y se quiere obtener el máximo valor posible sin exceder dicha capacidad. Se cuenta con 17 objetos -todos ellos fraccionables-, cuyos pesos y valores se dan en la siguiente tabla.

objetos	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	total
valor	23	15	10	19	5	83	55	45	39	13	71	67	64	43	37	77	87	753
peso	7	5	3	6	2	27	17	14	12	4	22	21	20	14	11	25	28	238

¿Cuál es el máximo valor alcanzable? ¿Con cuáles objetos? Explicar indicando el algoritmo utilizado. Justificar también la elección del algoritmo.

4. Utilizando inicialización virtual se ha llegado al siguiente estado de los arreglos:

mem	13	21	3	44	35	76	72	33	19	1	1	17	31	0	1
ord	9	3	15	1	5	8	5	4	9	13	1	7	6	43	37
sec	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Asumiendo que el contador cont está en 5, indique qué asignaciones se hicieron (a través del procedimiento asignar) y en qué orden (tenga presente que no todos los valores de la tabla tienen sentido, algunos pueden corresponder a "basura" que se encontraba en la memoria). ¿Puede deducir el valor de algunas celdas del arreglo sec? ¿Cuáles? ¿Qué valores deberían tener?

A modo de ayuda se incluyen las primitivas que realizan inicialización virtual.

```
type memoria = tuple
                 mem: array[1..15] of elem
                 ord: array[1..15] of int
                 sec: array[1..15] of int
                 cont: int
               end
proc inicializar(out m: memoria)
    m.cont:= 0
end
fun calculado(m: memoria, i: int) ret b: bool
   b:= (1 ≤ m.ord[i] ∧ m.ord[i] ≤ m.cont ∧ i=m.sec[m.ord[i]])
end
proc asignar(in/out m: memoria, i: int, e: elem) .        {pre: ¬calculado(m,i)}
    m.mem[i]:= e
    m.cont:= m.cont+1
    m.ord[i]:= m.cont
    m.sec[m.cont]:= i
end
```

5. El siguiente algoritmo calcula el menor número de monedas necesarias para pagar el monto M con denominaciones $d_1, \ldots, d_n$ utilizando programación dinámica.

```
fun moneda(d:array[1..n] of nat, M: nat) ret r: nat
   var m: array[0..n,0..M)] of nat
   for i:= 0 to n do m[i,0]:= 0 od
   for j:= 1 to M do m[0,j]:= ∞ od
   for i:= 1 to n do
      for j:= 1 to M do
         if d[i] >j then m[i,j]:= m[i-1,j] else m[i,j]:= min(m[i-1,j],1+m[i,j-d[i]]) fi
      od
   od
   r:= m[n,M]
end
```

Modificar el algoritmo de modo que calcule cuáles son las monedas que hacen falta para pagar de manera óptima dicho monto. Para ello, en vez de devolver r, el algoritmo devolverá un arreglo k: array[1..n] of nat, y en vez de realizar la asignación r:= m[n,M] deberá calcular el arreglo k.