6. 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.

```
proc quick_sort_rec(in/out a: array[1..n] of T, in lft,rgt: nat)
    var piv_izq,piv_der: nat

    if rgt > lft then
        partition(a,lft,rgt,piv_izq,piv_der)
        quick_sort_rec(a,lft,piv_izq-1)
        quick_sort_rec(a,piv_der+1,rgt)
    fi
end proc

proc partition(in/out a: array[1..n] of T, in lft,rgt: nat, out piv_izq,piv_der: nat)
    var i,j,x,k,t,pivot: nat

    pivot := lft
    piv_izq := 0
    piv_der := 0
    i := lft+1
    j := rgt

    while i ≤ j do
        if a[i] < a[pivot] → i := i+1
        □ a[j] > a[pivot] → j := j-1
        □ a[i] > a[pivot] ∧ a[j] < a[pivot] → swap(a,i,j)
                                              i := i+1
                                              j := j-1
        □ a[i] = a[pivot] → swap(a,i,lft+1+piv_izq)
                            piv_izq := piv_izq+1
                            i := i+1
        □ a[j] = a[pivot] → swap(a,j,rgt-pivot_der)
                            piv_der := piv_der+1
                            j := j-1
        fi
    od

    swap(a,j,pivot)
    x := 1
    k := lft+1
    t := rgt

    while k ≤ lft+piv_izq do
        swap(a,k,j-x)
        x := x+1
        k := k+1
    od
    while t > rgt-piv_der do
        swap(a,t,i)
        t := t-1
    od
    piv_izq := j-piv_izq
    piv_der := j+piv_der
end proc
```