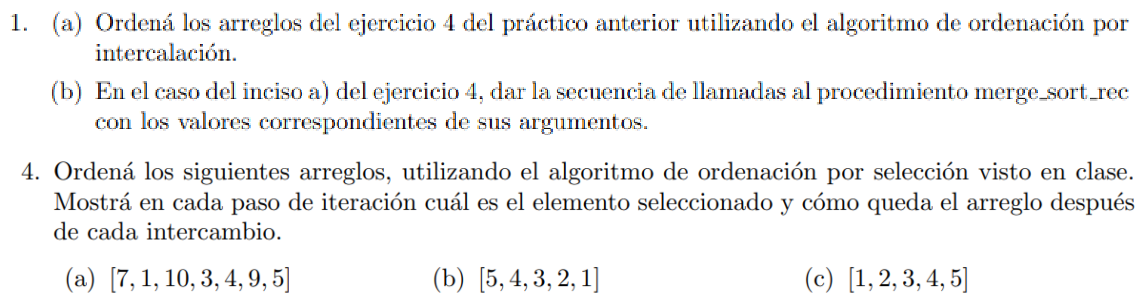
AYED II

Lautaro Gastón Peralta

Practico 1.2



1. (a) Ordená los arreglos del ejercicio 4 del práctico anterior utilizando el algoritmo de ordenación por intercalación.
   (b) En el caso del inciso a) del ejercicio 4, dar la secuencia de llamadas al procedimiento merge_sort_rec con los valores correspondientes de sus argumentos.

4. Ordená los siguientes arreglos, utilizando el algoritmo de ordenación por selección visto en clase. Mostrá en cada paso de iteración cuál es el elemento seleccionado y cómo queda el arreglo después de cada intercambio.

   (a) [7, 1, 10, 3, 4, 9, 5] (b) [5, 4, 3, 2, 1] (c) [1, 2, 3, 4, 5]

Merge:

```
proc merge (in/out a : array[1..N] of T, in lft,mid,rgt : nat)
    var tmp : array [1..n] of T
    var j,k : nat
    for  i:=lft to mid do tmp[i] := a[i] od
    j:=lft
    k:=mid+1
    for i:=lft to rgt do
        if j<=mid ^ (k>rgt v tmp[j]<=a[k]) then
            a[i]:=tmp[j]
            j:=j+1
        else
            a[i]:=a[k]
            k:=k+1
        fi
    od
end proc
```

Merge_sort_rec:

```
proc merge_sort_rec (in/out a : array[1..N] of T,in lft,rgt : nat)
    var mid : nat
    if rgt > lft ->  mid:= (rgt + lgt) / 2
                     merge_sort_rec(a,lft,mid)
                     merge_sort_rec(a,mid+1,rgt)
                     merge(a,lft,mid,rgt)
    fi
end proc
```

Merge_sort:

```
proc merge_sort(in/out a : array[1..N] of T)
    merge_sort_rec(a,1,n)
end proc
```

4 (a) y (b)

(a)

```
[7,1,10,3,4,9,5]
        merge_sort(a[1..7])
            merge_sort_rec(a,1,7)        [7,1,10,3,4,9,5]
            lft = 1 ; rgt = 7 ; if rgt(7) > lft (1) ; mid = 4
                merge_sort_rec(a,1,4) [7,1,10,3]
                merge_sort_rec(a,5,7) [4,9,5]
                merge (a,1,4,7) se espera 1.1.1.1 y 1.1.1.2
            merge_sort_rec(a,1,4)        [7,1,10,3]
            lft = 1 ; rgt = 4 ; if rgt(4) > lft(1) ; mid = 2
                merge_sort_rec(a,1,2) [7,1]
                merge_sort_rec(a,3,4) [10,3]
                merge (a,1,2,4) se espera 1.1.2.1 y 1.1.3.1
                    merge_sort_rec(a,1,2) [7,1]
                    lft = 1 ; rgt = 2 ; if rgt(2) > lft(1) ; mid = 1
                        merge_sort_rec(a,1,1) [7]
                        merge_sort_rec(a,2,2) [1]
                        merge (a,1,1,2) espera a 1.1.2.2 y 1.1.2.3
                            merge_sort_rec(a,1,1) [7]
                            {se asume ordenado}
                            merge_sort_rec(a,2,2) [1]
                            {se asume ordenado}
                *merge (a,1,1,2)
                a = [7,1] ; tmp = [7,_] ; j = 1 ; k = 2 ; for i:=lft(1) to rgt(2) do
                    i:=1
                    if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {no se cumple}
                    else -> a[1] = a[2]      k = k+1 -> 3

                a = [1,1] ; tmp = [7,_] ; j = 1 ; k = 3 ; for i:=lft(2) to rgt(2) do
                    i:=2
                    if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
                        -> a[2] = tmp [1] -> j=j+1 -> 2
                        a=[1,7] ; tmp = [7,_]
                SE TERMINA CICLO Y MERGE(a,1,1,2) -> a = [1,7]

                    merge_sort_rec(a,3,4) [10,3]
                    lft = 3 ; rgt = 4 ; if rgt(4) > lft(3) ; mid = 3
                        merge_sort_rec(a,3,3) [10]
                        merge_sort_rec(a,4,4) [3]
                        merge (a,3,3,4) espera a 1.1.2.4 y 1.1.2.5
                            merge_sort_rec(a,3,3) [10]
                            {se asume ordenado}
                            merge_sort_rec(a,4,4) [3]
                            {se asume ordenado}
                *merge (a,3,3,4)
                a = [10,3] ; tmp = [10,_] ; j = 3 ; k = 4 ; for i:=lft(3) to rgt(4) do
                    i:=3
                    if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {no se cumple}
                    else -> a[3] = a[4]      k = k+1 -> 5
```

```
    a = [3,3] ; tmp = [10,_] ; j = 3 ; k = 5 ; for i:=lft(4) to rgt(4) do
        i:=4
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
            -> a[4] = tmp [3] -> j=j+1 -> 2
            a=[3,10] ; tmp = [10,_]
    SE TERMINA CICLO Y MERGE(a,3,3,4) -> a = [3,10]

    *merge (a,1,2,4) a = [1,7,3,10]
    mid = 2 ; tmp = [1,7,_,_] ; j = 1 ; k = 3 ; for i:=lft(1) to rgt(4) do
        i:=1
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
        ->      a[1] = tmp[1]  j=j+1->2  -> a = [1,7,3,10]

    mid = 2 ; tmp = [1,7,_,_] ; j = 2 ; k = 3 ; for i:=lft(2) to rgt(4) do
        i:=2
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
        else ->         a[2] = a[3]      k=k+1->4-> a = [1,3,3,10]

    mid = 2 ; tmp = [1,7,_,_] ; j = 2 ; k = 4 ; for i:=lft(3) to rgt(4) do
        i:=3
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
        ->      a[3] = tmp[2]  j=j+1->3  -> a = [1,3,7,10]

    mid = 2 ; tmp = [1,7,_,_] ; j = 3 ; k = 4 ; for i:=lft(4) to rgt(4) do
        i:=4
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
        ->      a = [1,3,7,10]
merge_sort_rec(a,5,7)        [4,9,5]
lft = 5 ; rgt = 7 ; if rgt(7) > lft(5) ; mid = 6
    merge_sort_rec(a,5,6) [4,9]
    merge_sort_rec(a,7,7) [5]
    merge (a,5,6,7) se espera 1.1.2.2 y 1.1.2.3
        merge_sort_rec(a,5,6) [4,9]
        lft = 5 ; rgt = 6 ; if rgt(6) > lft(5) ; mid = 5
            merge_sort_rec(a,5,5) [4]
            merge_sort_rec(a,6,6) [9]
            merge (a,5,5,6) espera a 1.1.3.7 y 1.1.3.8
                merge_sort_rec(a,5,5) [4]
                {se asume ordenado}
                merge_sort_rec(a,6,6) [9]
                {se asume ordenado}
    *merge (a,5,5,6)
    a = [4,9] ; tmp = [4,_] ; j = 5 ; k = 6 ; for i:=lft(5) to rgt(6) do
        i:=5
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
        -> a[5] = tmp[6]         j = j+1 -> 6
```

```
                a = [4,9] ; tmp = [4,_] ; j = 6 ; k = 6 ; for i:=lft(6) to rgt(6) do
                    i:=6
                    if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
                        -> a[6] = a[6] -> k=k+1 -> 7
                        a=[4,9] ; tmp = [4,_]
                SE TERMINA CICLO Y MERGE(a,5,5,6) -> a = [4,9]
                merge_sort_rec(a,7,7) [5]
                {SE ASUME ORDENADO}
                *1.1.2.7 merge (a,5,6,7)
                a = [4,9,5] ; tmp = [4,9,_] ; j = 5 ; k = 7 ; for i:=lft(5) to rgt(7) do
                    i:=5
                    if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
                    -> a[5] = tmp[5]          j = j+1 -> 6
                a = [4,9,5] ; tmp = [4,9,_] ; j = 6 ; k = 7 ; for i:=lft(6) to rgt(7) do
                    i:=6
                    if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
                    -> a[6] = a[7]   k = k+1 -> 8
                a = [4,5,5] ; tmp = [4,9,_] ; j = 6 ; k = 8 ; for i:=lft(7) to rgt(7) do
                    i:=7
                    if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
                    -> a[7] = tmp[6]          j = j+1 -> 7
                SE TERMINA CICLO Y MERGE(a,5,6,7) -> a = [4,5,9]
        TENEMOS DOS PARTES ORDENADAS, QUEDA EL ULTIMO MERGE:
            [1,3,7,10] y [4,5,9]
*merge (a,1,4,7)
a = [1,3,7,10,4,5,9] ; tmp = [1,3,7,10,_,_,_] ; mid =4 ; j = 1 ; k = 5 ; for i:=lft(1) to rgt(7) do
        i:=1
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
        -> a[1] = tmp[1]          j = j+1 -> 2
a = [1,3,7,10,4,5,9] ; tmp = [1,3,7,10,_,_,_] ; mid =4 ; j = 2 ; k = 5 ; for i:=lft(2) to rgt(7) do
        i:=2
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
        -> a[2] = tmp[2]          j = j+1 -> 3
a = [1,3,7,10,4,5,9] ; tmp = [1,3,7,10,_,_,_] ; mid =4 ; j = 3 ; k = 5 ; for i:=lft(3) to rgt(7) do
        i:=3
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
        -> a[3] = a[5]            k = k+1 -> 6
a = [1,3,4,10,4,5,9] ; tmp = [1,3,7,10,_,_,_] ; mid =4 ; j = 3 ; k = 6 ; for i:=lft(4) to rgt(7) do
        i:=4
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
        -> a[4] = a[6]            k = k+1 -> 7
a = [1,3,4,5,4,5,9] ; tmp = [1,3,7,10,_,_,_] ; mid =4 ; j = 3 ; k = 7 ; for i:=lft(5) to rgt(7) do
        i:=5
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
        -> a[5] = tmp[3]          j = j+1 -> 4
a = [1,3,4,5,7,5,9] ; tmp = [1,3,7,10,_,_,_] ; mid =4 ; j = 4 ; k = 7 ; for i:=lft(6) to rgt(7) do
        i:=6
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
        -> a[6] = a[7]              k = k+1 -> 8
a = [1,3,4,5,7,9,9] ; tmp = [1,3,7,10,_,_,_] ; mid =4 ; j = 4 ; k = 8 ; for i:=lft(7) to rgt(7) do
        i:=7
        if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
        -> a[7] = tmp[4]            j = j+1 -> 5
        SE TERMINA CICLO Y MERGE(a,1,4,7) -> a = [1,3,4,5,7,9,10]
```

(b)
[5,4,3,2,1]

```
        merge_sort(a[1..5])
                merge_sort_rec(a,1,5)        [5,4,3,2,1]
                lft = 1 ; rgt = 5 ; if rgt(5) > lft (1) ; mid = 3
                        merge_sort_rec(a,1,3) [5,4,3]
                        merge_sort_rec(a,4,5) [2,1]
                        merge (a,1,3,5)
                merge_sort_rec(a,1,3)        [5,4,3]
                lft = 1 ; rgt = 3 ; if rgt(3) > lft (1) ; mid = 2
                        merge_sort_rec(a,1,2) [5,4]
                        merge_sort_rec(a,3,3) [3] {SE ASUME ORDENADO}
                        merge (a,1,2,3)
                merge_sort_rec(a,1,2)        [5,4]
                lft = 1 ; rgt = 2 ; if rgt(2) > lft (1) ; mid = 1
                        merge_sort_rec(a,1,1) [5] {SE ASUME ORDENADO}
                        merge_sort_rec(a,2,2) [4] {SE ASUME ORDENADO}
                        merge (a,1,1,2)
                merge(a,1,1,2)          [5,4]
                a = [5,4] ; tmp = [5,_] ; j = 1 ; k = 2 ; for i:=lft(1) to rgt(2) do
                i:=1
                if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
                -> a[1] = a[2]              k = k+1 -> 3
                a = [4,4] ; tmp = [5,_] ; j = 1 ; k = 3 ; for i:=lft(2) to rgt(2) do
                i:=2
                if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
                -> a[2] = tmp[1]            j = j+1 -> 2
                SE TERMINA CICLO Y MERGE(a,1,1,2) -> a = [4,5]
                merge (a,1,2,3)         [4,5,3]
                a = [4,5,3] ; tmp = [4,5,_] ; j = 1 ; k = 3 ; for i:=lft(1) to rgt(3) do
                i:=1
                if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
                -> a[1] = a[3]              k = k+1 -> 4
                a = [3,5,3] ; tmp = [4,5,_] ; j = 1 ; k = 4 ; for i:=lft(2) to rgt(3) do
                i:=2
                if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
                -> a[2] = tmp[1]            j = j+1 -> 2
                a = [3,4,3] ; tmp = [4,5,_] ; j = 2 ; k = 4 ; for i:=lft(3) to rgt(3) do
                i:=3
                if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
                -> a[3] = tmp[2]            j = j+1 -> 3
```

```
            SE TERMINA CICLO Y MERGE(a,1,2,3) -> a = [3,4,5]
            merge_sort_rec(a,4,5)        [2,1]
            lft = 4 ; rgt = 5 ; if rgt(5) > lft (4) ; mid = 4
                merge_sort_rec(a,4,4) [2] {SE ASUME ORDENADO}
                merge_sort_rec(a,5,5) [1] {SE ASUME ORDENADO}
                merge (a,4,4,5)
            merge(a,4,4,5)         [2,1]
            a = [2,1] ; tmp = [2,_] ; j = 4 ; k = 5 ; for i:=lft(4) to rgt(5) do
            i:=1
            if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
            -> a[1] = a[2]            k = k+1 -> 6
            a = [1,1] ; tmp = [2,_] ; j = 4 ; k = 6 ; for i:=lft(5) to rgt(5) do
            i:=2
            if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
            -> a[2] = tmp[1]          j = j+1 -> 5
            SE TERMINA CICLO Y MERGE(a,4,4,5) -> a = [1,2]
        TENEMOS DOS PARTES ORDENADAS, QUEDA EL ULTIMO MERGE:
            [3,4,5] y [1,2]
            merge(a,1,3,5)         [3,4,5,1,2]
            a = [3,4,5,1,2] ; tmp = [3,4,5,_] ; j = 1 ; k = 4 ; for i:=lft(1) to rgt(5) do
            i:=1
            if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
            -> a[1] = a[4]            k = k+1 -> 5
            a = [1,4,5,1,2] ; tmp = [3,4,5,_] ; j = 1 ; k = 5 ; for i:=lft(2) to rgt(5) do
            i:=2
            if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
            -> a[2] = a[5]            k = k+1 -> 6
            a = [1,2,5,1,2] ; tmp = [3,4,5,_] ; j = 1 ; k = 6 ; for i:=lft(3) to rgt(5) do
            i:=3
            if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
            -> a[3] = tmp[1]          j = j+1 -> 2
            a = [1,2,3,1,2] ; tmp = [3,4,5,_] ; j = 2 ; k = 6 ; for i:=lft(4) to rgt(5) do
            i:=4
            if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
            -> a[4] = tmp[2]          j = j+1 -> 3
            a = [1,2,3,4,2] ; tmp = [3,4,5,_] ; j = 3 ; k = 6 ; for i:=lft(5) to rgt(5) do
            i:=5
            if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
            -> a[5] = tmp[3]          j = j+1 -> 4
        SE TERMINA CICLO Y MERGE(a,1,3,5) -> a = [1,2,3,4,5]
```

(c)
[1,2,3,4,5]

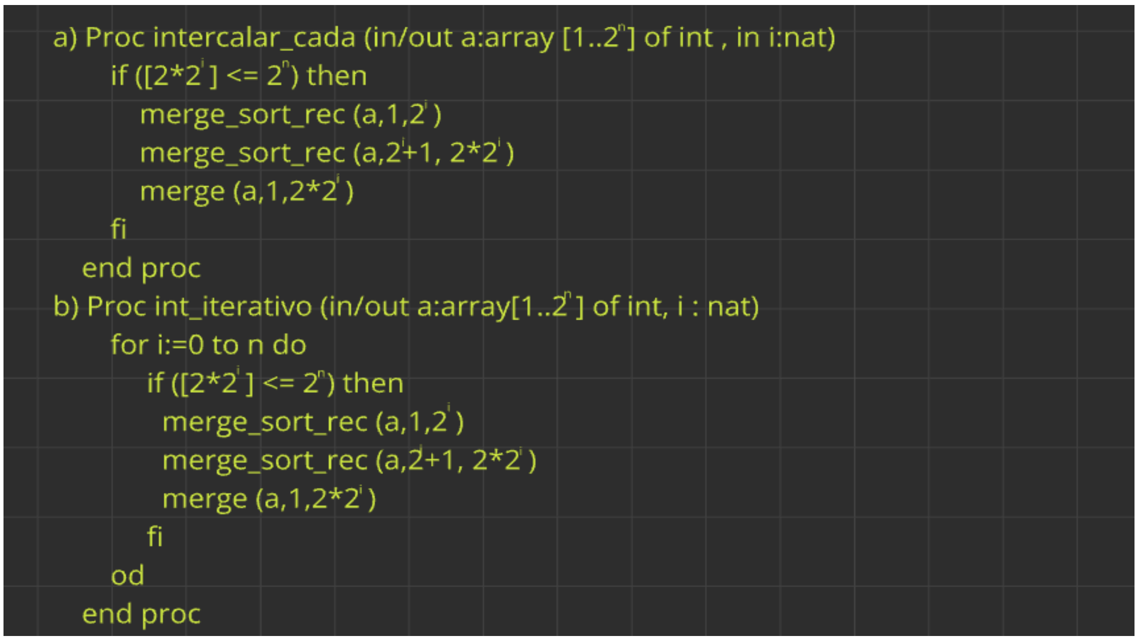
```
        merge_sort(a[1..5])
            merge_sort_rec(a,1,5)        [1,2,3,4,5]
            lft = 1 ; rgt = 5 ; if rgt(5) > lft (1) ; mid = 3
                merge_sort_rec(a,1,3) [1,2,3]
                merge_sort_rec(a,4,5) [4,5]
                merge (a,1,3,5)
```

```
merge_sort_rec(a,1,3)        [1,2,3]
lft = 1 ; rgt = 3 ; if rgt(3) > lft (1) ; mid = 2
        merge_sort_rec(a,1,2) [1,2]
        merge_sort_rec(a,3,3) [3] {SE ASUME ORDENADO}
        merge (a,1,2,3)
merge_sort_rec(a,1,2)        [1,2]
lft = 1 ; rgt = 2 ; if rgt(2) > lft (1) ; mid = 1
        merge_sort_rec(a,1,1) [1] {SE ASUME ORDENADO}
        merge_sort_rec(a,2,2) [2] {SE ASUME ORDENADO}
        merge (a,1,1,2)
merge(a,1,1,2)        [1,2]
a = [1,2] ; tmp = [1,_] ; j = 1 ; k = 2 ; for i:=lft(1) to rgt(2) do
i:=1
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
-> a[1] = tmp[1]        j = j+1 -> 2
a = [1,2] ; tmp = [1,_] ; j = 2 ; k = 2 ; for i:=lft(2) to rgt(2) do
i:=2
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
-> a[2] = a[2]          k = k+1 -> 3
SE TERMINA CICLO Y MERGE(a,1,1,2) -> a = [1,2]
merge (a,1,2,3)        [1,2,3]
a = [1,2,3] ; tmp = [1,2,_] ; j = 1 ; k = 3 ; for i:=lft(1) to rgt(3) do
i:=1
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
-> a[1] = tmp[1]        j = j+1 -> 2
a = [1,2,3] ; tmp = [1,2,_] ; j = 2 ; k = 3 ; for i:=lft(2) to rgt(3) do
i:=2
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
-> a[2] = tmp[2]        j = j+1 -> 3
a = [1,2,3] ; tmp = [1,2,_] ; j = 3 ; k = 3 ; for i:=lft(3) to rgt(3) do
i:=3
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
-> a[3] = a[3]          k = k+1 -> 4
SE TERMINA CICLO Y MERGE(a,1,2,3) -> a = [1,2,3]
merge_sort_rec(a,4,5)        [4,5]
lft = 4 ; rgt = 5 ; if rgt(5) > lft (4) ; mid = 4
        merge_sort_rec(a,4,4) [4] {SE ASUME ORDENADO}
        merge_sort_rec(a,5,5) [5] {SE ASUME ORDENADO}
        merge (a,4,4,5)
merge(a,4,4,5)        [4,5]
a = [4,5] ; tmp = [4,_] ; j = 4 ; k = 5 ; for i:=lft(4) to rgt(5) do
i:=1
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {CUMPLE}
-> a[1] = tmp[2]        j = j+1 -> 5
a = [4,5] ; tmp = [4,_] ; j = 5 ; k = 5 ; for i:=lft(5) to rgt(5) do
i:=2
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
-> a[2] = a[5]          k = k+1 -> 6
```

SE TERMINA CICLO Y MERGE(a,4,4,5) -> a = [4,5]
TENEMOS DOS PARTES ORDENADAS, QUEDA EL ULTIMO MERGE:
[1,2,3] y [4,5]
merge(a,1,3,5)   [1,2,3,4,5]
a = [1,2,3,4,5] **;** tmp = [1,2,3,_] ; j = 1 ; k = 4 ; **for i:=lft(1) to rgt(5) do**
i:=1
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
-> a[1] = tmp[1]   j = j+1 -> 2
a = [1,2,3,4,5] **;** tmp = [1,2,3,_] ; j = 2 ; k = 4 ; **for i:=lft(2) to rgt(5) do**
i:=2
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
-> a[2] = tmp[2]   j = j+1 -> 3
a = [1,2,3,4,5] **;** tmp = [1,2,3,_] ; j = 3 ; k = 4 ; **for i:=lft(3) to rgt(5) do**
i:=3
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {SE CUMPLE}
-> a[3] = tmp[3]   j = j+1 -> 4
a = [1,2,3,4,5] **;** tmp = [1,2,3,_] ; j = 4 ; k = 4 ; **for i:=lft(4) to rgt(5) do**
i:=4
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
-> a[4] = a[4]   k = k+1 -> 5
a = [1,2,3,4,5] **;** tmp = [3,4,5,_] ; j = 4 ; k = 5 ; **for i:=lft(5) to rgt(5) do**
i:=5
if j <= mid ^ (k>rgt v tmp[j]<=a[k]) {NO SE CUMPLE}
-> a[5] = a[5]   k = k+1 -> 6
SE TERMINA CICLO Y MERGE(a,1,3,5) -> a = [1,2,3,4,5]

2. (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.



```
a) Proc intercalar_cada (in/out a:array [1..2^n] of int , in i:nat)
     if ([2*2^i ] <= 2^n) then
        merge_sort_rec (a,1,2^i )
        merge_sort_rec (a,2^i+1, 2*2^i )
        merge (a,1,2*2^i )
     fi
  end proc
b) Proc int_iterativo (in/out a:array[1..2^n ] of int, i : nat)
     for i:=0 to n do
        if ([2*2^i ] <= 2^n) then
          merge_sort_rec (a,1,2^i )
          merge_sort_rec (a,2^i+1, 2*2^i )
          merge (a,1,2*2^i )
        fi
     od
  end proc
```

3. (a) Ordená los arreglos del ejercicio 4 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.

Partition:

```
proc partition (in/out a : array[1..N] of T, in lft, rgt : nat, out ppiv : nat)
        var i,j : nat
        ppiv := lft
        i:=lft+1
        j:=rgt
        do i<=j ->      if a[i] <= a[ppiv] -> i:=i+1
                        if a[j] >= a[ppiv] -> j:=j-1
                        if a[i] > a[ppiv] ^ a[j] < a[ppiv] ->      swap(a,i,j)
                                                                   i:=i+1
                                                                   j:=j-1
                        fi
        od
        swap(a,ppiv,j)
        ppiv:=j
end proc
```

Quick_sort_rec:

```
proc quick_sort_rec (in/out a : array [1..N] of T, in lft, rgt : nat)
        var ppiv : nat
        if rgt>lft ->   partition(a,lft,rgt,ppiv)
                        quick_sort_rec(a,lft,ppiv-1)
                        quick_sort_rec(a,ppiv+1,rgt)
        fi
end proc
```

Quick_sort:

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

4 (a) y (b)

(a)

[7,1,10,3,4,9,5]

```
        quick_sort(a[1..7])
                quick_sort_rec(a,1,7)
                        lft=1 ; rgt = 7 ; if rgt(7) > lft(1)
                        partition(a,1,7,ppiv) -> ppiv = 5
                        quick_sort_rec(a,1,ppiv-1) -> quick_sort_rec(a,1,4)
                        quick_sort_rec(a,ppiv+1,7)  -> quick_sort_rec(a,6,7)
                *partition(a,1,7,ppiv)
                lft = 1 ; rgt = 7 ; ppiv = 1 ; i = 2 ; j = 7 do 2<=7
                        if a[2] <= a[1] -> si -> i = i+1 -> i=3
                        if a[7] >= a[1] -> no
                        if a[2] > a[1] ^ a[7] < a[1] -> no
                lft = 1 ; rgt = 7 ; ppiv = 1 ; i = 3 ; j = 7 do 3<=7
                        if a[3] <= a[1] -> no
                        if a[7] >= a[1] -> no
                        if a[3] > a[1] ^ a[7] < a[1] -> si      swap(a,3,7)
                                                                i=i+1 -> i = 4
                                                                j=j-1 -> j = 6
                                                                [7,1,5,3,4,9,10]
                lft = 1 ; rgt = 7 ; ppiv = 1 ; i = 4 ; j = 6 do 4<=6
                        if a[4] <= a[1] -> si -> i=i+1 -> i = 5
                        if a[6] >= a[1] -> si -> j=j-1 -> j = 5
                        if a[4] > a[1] ^ a[6] < a[1] -> no
                lft = 1 ; rgt = 7 ; ppiv = 1 ; i = 5 ; j = 5 do 5<=5
                        if a[5] <= a[1] -> si -> i=i+1 -> i=6
                        if a[5] >= a[1] -> no
                        if a[5] > a[1] ^ a[5] < a[1] -> no
                lft = 1 ; rgt = 7 ; ppiv = 1 ; i = 6 ; j = 5 do 6<=5 {TERMINA CICLO}
                [7,1,5,3,4,9,10]
                swap (a,1,5)
                ppiv := j -> ppiv = 5
                [4,1,5,3,7,9,10]
```

```
 quick_sort_rec(a,1,4)          [4,1,5,3]
          lft=1 ; rgt = 4 ; if rgt(4) > lft(1)
          partition(a,1,4,ppiv) -> ppiv = 3
          quick_sort_rec(a,1,ppiv-1) -> quick_sort_rec(a,1,2)
          quick_sort_rec(a,ppiv+1,4)  -> quick_sort_rec(a,4,4)
*partition(a,1,4,ppiv)
lft = 1 ; rgt = 4 ; ppiv = 1 ; i = 2 ; j = 4 do 2<=4
          if a[2] <= a[1] -> si -> i = i+1 -> i=3
          if a[4] >= a[1] -> no
          if a[2] > a[1] ^ a[4] < a[1] -> no
lft = 1 ; rgt = 4 ; ppiv = 1 ; i = 3 ; j = 4 do 3<=4
          if a[3] <= a[1] -> no
          if a[4] >= a[1] -> no
          if a[3] > a[1] ^ a[7] < a[1] -> si          swap(a,3,4)
                                                      i=i+1 -> i = 4
                                                      j=j-1 -> j = 3
                                                      [4,1,3,5]
lft = 1 ; rgt = 4 ; ppiv = 1 ; i = 4 ; j = 3 do 4<=3 {TERMINA CICLO}
[4,1,3,5]
swap (a,1,3)
ppiv := j -> ppiv = 3
[3,1,4,5]
quick_sort_rec(a,1,2)          [3,1]
          lft=1 ; rgt = 2 ; if rgt(2) > lft(1)
          partition(a,1,2,ppiv) -> ppiv = 2
          quick_sort_rec(a,1,ppiv-1) -> quick_sort_rec(a,1,1)
          quick_sort_rec(a,ppiv+1,4)  -> quick_sort_rec(a,2,2)
*partition(a,1,2,ppiv)
lft = 1 ; rgt = 2 ; ppiv = 1 ; i = 2 ; j = 2 do 2<=2
          if a[2] <= a[1] -> si -> i = i+1 -> i=3
          if a[2] >= a[1] -> no
          if a[2] > a[1] ^ a[2] < a[1] -> no
lft = 1 ; rgt = 2 ; ppiv = 1 ; i = 3 ; j = 2 do 3<=2 {TERMINA CICLO}
[3,1]
swap (a,1,2)
ppiv := j -> ppiv = 2
[1,3]
quick_sort_rec(a,1,1) -> se asume ordenado
quick_sort_rec(a,2,2) -> se asume ordenado

quick_sort_rec(a,4,4) -> se asume ordenado
se termina la llamada ->  quick_sort_rec(a,1,4) -> res => [1,3,4,5]

quick_sort_rec(a,6,7)          [9,10]
          lft=6 ; rgt = 7 ; if rgt(7) > lft(6)
          partition(a,6,7,ppiv) -> ppiv = 6
          quick_sort_rec(a,1,ppiv-1) -> quick_sort_rec(a,6,5)
          quick_sort_rec(a,ppiv+1,4)  -> quick_sort_rec(a,7,7)
```

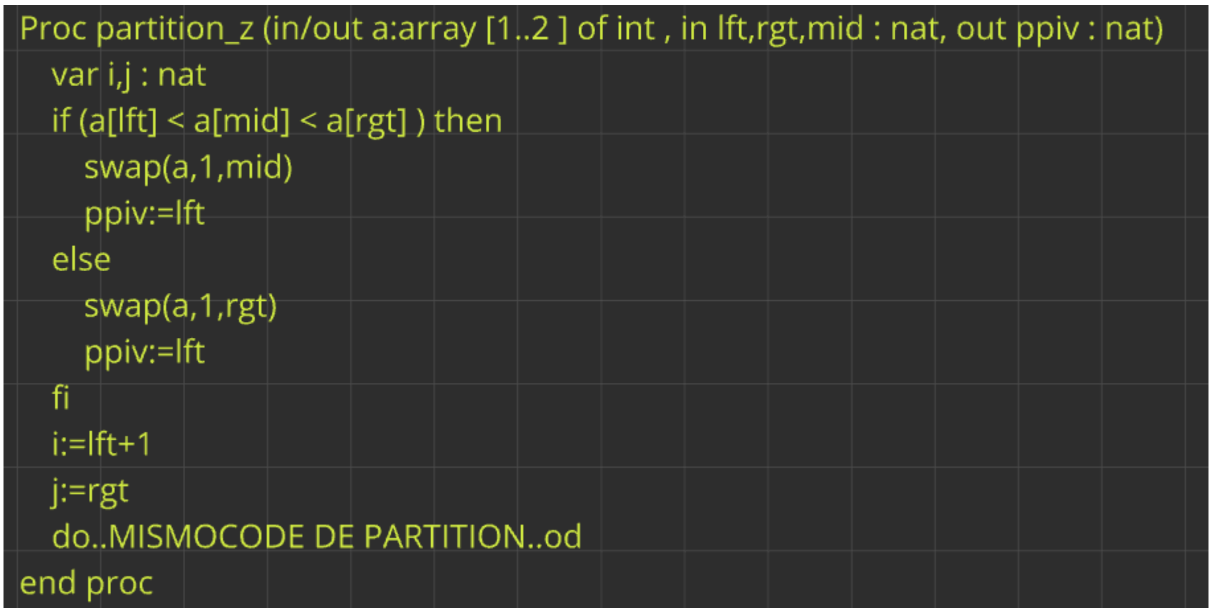
```
        *partition(a,6,7,ppiv)
        lft = 6 ; rgt = 7 ; ppiv = 6 ; i = 7 ; j = 7 do 7<=7
            if a[7] <= a[6] -> no
            if a[7] >= a[6] -> si -> j=j-1 -> j=6
            if a[7] > a[6] ^ a[7] < a[6] -> no
        lft = 1 ; rgt = 2 ; ppiv = 6 ; i = 7 ; j = 6 do 7<=6 {TERMINA CICLO}
        [9,10]
        swap (a,6,6)
        ppiv := j -> ppiv = 6
        [9,10]
        quick_sort_rec(a,6,5) -> se asume ordenado -> cond: rgt > lft.
        quick_sort_rec(a,7,7) -> se asume ordenado
        se termina la llamada ->  quick_sort_rec(a,1,4) -> res => [9,10]
        RESULTADO DE TODO => [1,3,4,5,7,9,10]
(b)
[5,4,3,2,1]
    quick_sort(a[1..5])
        quick_sort_rec(a,1,5)
            lft=1 ; rgt = 5 ; if rgt(5) > lft(1)
            partition(a,1,5,ppiv) -> ppiv = 5
            quick_sort_rec(a,1,ppiv-1) -> quick_sort_rec(a,1,4)
            quick_sort_rec(a,ppiv+1,5)  -> quick_sort_rec(a,6,5)
        *partition(a,1,5,ppiv)
        lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 2 ; j = 5 do 2<=5
            if a[2] <= a[1] -> si -> i = i+1 -> i=3
            if a[5] >= a[1] -> no
            if a[2] > a[1] ^ a[7] < a[1] -> no
        lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 3 ; j = 5 do 3<=5
            if a[3] <= a[1] -> si -> i = i+1 -> i=4
            if a[5] >= a[1] -> no
            if a[3] > a[1] ^ a[5] < a[1] -> no
        lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 4 ; j = 5 do 4<=5
            if a[4] <= a[1] -> si -> i=i+1 -> i = 5
            if a[5] >= a[1] -> no
            if a[4] > a[1] ^ a[5] < a[1] -> no
        lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 5 ; j = 5 do 5<=5
            if a[5] <= a[1] -> si -> i=i+1 -> i=6
            if a[5] >= a[1] -> no
            if a[5] > a[1] ^ a[5] < a[1] -> no
        lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 6 ; j = 5 do 6<=5 {TERMINA CICLO}
        [5,4,3,2,1]
        swap (a,1,5)
        ppiv := j -> ppiv = 5
        [1,4,3,2,5]
         quick_sort_rec(a,1,4)          [1,4,3,2]
            lft=1 ; rgt = 4 ; if rgt(4) > lft(1)
            partition(a,1,4,ppiv) -> ppiv = 1
            quick_sort_rec(a,1,ppiv-1) -> quick_sort_rec(a,1,0)
    quick_sort_rec(a,ppiv+1,4)  -> quick_sort_rec(a,2,4)
*partition(a,1,4,ppiv)
lft = 1 ; rgt = 4 ; ppiv = 1 ; i = 2 ; j = 4 do 2<=4
    if a[2] <= a[1] -> no
    if a[4] >= a[1] -> si -> j = j-1 -> j = 3
    if a[2] > a[1] ^ a[4] < a[1] -> no
lft = 1 ; rgt = 4 ; ppiv = 1 ; i = 2 ; j = 3 do 2<=3
    if a[2] <= a[1] -> no
    if a[3] >= a[1] -> si -> j = j-1 -> j = 2
    if a[2] > a[1] ^ a[3] < a[1] -> no
lft = 1 ; rgt = 4 ; ppiv = 1 ; i = 2 ; j = 2 do 2<=2
    if a[2] <= a[1] -> no
    if a[2] >= a[1] -> si -> j = j-1 -> j = 1
    if a[2] > a[1] ^ a[2] < a[1] -> no
lft = 1 ; rgt = 4 ; ppiv = 1 ; i = 2 ; j = 1 do 2<=1 {TERMINA CICLO}
[1,4,3,2]
swap (a,1,1)
ppiv := j -> ppiv = 1
[1,4,3,2]
quick_sort_rec(a,1,0) [1]  -> se asume ordenado -> cond: rgt > lft
quick_sort_rec(a,2,4) [4,3,2]
    lft=2 ; rgt = 4 ; if rgt(4) > lft(2)
    partition(a,2,4,ppiv) -> ppiv = 2
    quick_sort_rec(a,2,ppiv-1) -> quick_sort_rec(a,2,1)
    quick_sort_rec(a,ppiv+1,4)  -> quick_sort_rec(a,3,3)
*partition(a,2,4,ppiv)
lft = 2 ; rgt = 4 ; ppiv = 2 ; i = 3 ; j = 4 do 3<=4
    if a[3] <= a[2] -> si -> i = i + 1 -> i = 4
    if a[4] >= a[2] -> no
    if a[3] > a[2] ^ a[4] < a[2] -> no
lft = 2 ; rgt = 4 ; ppiv = 2 ; i = 4 ; j = 4 do 4<=4
    if a[4] <= a[2] -> si -> i = i + 1 -> i = 5
    if a[4] >= a[2] -> no
    if a[4] > a[2] ^ a[4] < a[2] -> no
lft = 2 ; rgt = 4 ; ppiv = 2 ; i = 5 ; j = 4 do 5<=4 {TERMINA CICLO}
[4,3,2]
swap (a,2,4)
ppiv := j -> ppiv = 4
[2,3,4]
quick_sort_rec(a,2,3) [2,3]
    lft=2 ; rgt =3 ; if rgt(3) > lft(2)
    partition(a,2,3,ppiv) -> ppiv = 4
    quick_sort_rec(a,2,ppiv-1) -> quick_sort_rec(a,2,3)
    quick_sort_rec(a,ppiv+1,4)  -> quick_sort_rec(a,5,4)
*partition(a,2,3,ppiv)
lft = 2 ; rgt = 3 ; ppiv = 2 ; i = 3 ; j = 3 do 3<=3
    if a[3] <= a[2] -> no
    if a[3] >= a[2] -> si -> j = j-1 -> j = 2
```

```
                if a[3] > a[2] ^ a[3] < a[2] -> no
            lft = 2 ; rgt = 3 ; ppiv = 2 ; i = 3 ; j = 2 do 3<=2 {TERMINA CICLO}
            [2,3]
            swap (a,2,2)
            ppiv := j -> ppiv = 2
            [2,3]
            quick_sort_rec(a,2,1) [2] -> se asume ord -> cond rgt > lft
            quick_sort_rec(a,3,3) [3] -> se asume ord -> cond rgt > lft
            se termina la llamada -> quick_sort_rec(a,1,4) -> [1,2,3,4]
            quick_sort_rec(a,6,5) -> se asume ord -> cond rgt > lft
            RESULTADO DE TODO => [1,2,3,4,5]
(c)
[1,2,3,4,5]
    quick_sort(a[1..5])
            quick_sort_rec(a,1,5)
                lft=1 ; rgt = 5 ; if rgt(5) > lft(1)
                partition(a,1,5,ppiv) -> ppiv = 1
                quick_sort_rec(a,1,ppiv-1) -> quick_sort_rec(a,1,0)
                quick_sort_rec(a,ppiv+1,5)  -> quick_sort_rec(a,2,5)
            *partition(a,1,5,ppiv)
            lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 2 ; j = 5 do 2<=5
                if a[2] <= a[1] -> no
                if a[5] >= a[1] -> si -> j = j-1 -> j = 4
                if a[2] > a[1] ^ a[7] < a[1] -> no
            lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 2 ; j = 4 do 2<=4
                if a[2] <= a[1] -> no
                if a[4] >= a[1] -> si -> j = j-1 -> j = 3
                if a[2] > a[1] ^ a[4] < a[1] -> no
            lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 2 ; j = 3 do 5<=3
                if a[2] <= a[1] -> no
                if a[3] >= a[1] -> si -> j = j-1 -> j = 2
                if a[2] > a[1] ^ a[3] < a[1] -> no
            lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 2 ; j = 2 do 2<=2
                if a[2] <= a[1] -> no
                if a[2] >= a[1] -> si -> j = j-1 -> j = 1
                if a[2] > a[1] ^ a[2] < a[1] -> no
            lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 2 ; j = 1 do 2<=1 {TERMINA CICLO}
            [1,2,3,4,5]
            swap (a,1,1)
            ppiv := j -> ppiv = 1
            [1,2,3,4,5]
            quick_sort_rec(a,ppiv+1,5)  [2,3,4,5]
                lft=2 ; rgt = 5 ; if rgt(5) > lft(2)
                partition(a,2,5,ppiv) -> ppiv = 2
                quick_sort_rec(a,2,ppiv-1) -> quick_sort_rec(a,2,1)
                quick_sort_rec(a,ppiv+1,5)  -> quick_sort_rec(a,3,5)
            *partition(a,2,5,ppiv)
            lft = 2 ; rgt = 5 ; ppiv = 2 ; i = 3 ; j = 5 do 3<=5
```

```
        if a[3] <= a[2] -> no
        if a[5] >= a[2] -> si -> j = j-1 -> j = 4
        if a[3] > a[2] ^ a[5] < a[2] -> no
lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 3 ; j = 4 do 3<=4
        if a[3] <= a[2] -> no
        if a[4] >= a[2] -> si -> j = j-1 -> j = 3
        if a[3] > a[1] ^ a[4] < a[2] -> no
lft = 1 ; rgt = 5 ; ppiv = 1 ; i = 3; j = 3 do 3<=3
        if a[3] <= a[2] -> no
        if a[3] >= a[2] -> si -> j = j-1 -> j = 2
        if a[3] > a[2] ^ a[3] < a[2] -> no
lft = 1 ; rgt = 4 ; ppiv = 1 ; i = 2 ; j = 1 do 3<=2 {TERMINA CICLO}
[2,3,4,5]
swap (a,2,2)
ppiv := j -> ppiv = 2
[2,3,4,5]
quick_sort_rec(a,2,1) [2]  -> se asume ordenado -> cond: rgt > lft
quick_sort_rec(a,3,5) [3,4,5]
        lft=3 ; rgt = 5 ; if rgt(5) > lft(3)
        partition(a,3,5,ppiv) -> ppiv = 3
        quick_sort_rec(a,2,ppiv-1) -> quick_sort_rec(a,2,2)
        quick_sort_rec(a,ppiv+1,4)  -> quick_sort_rec(a,4,5)
*partition(a,3,5,ppiv)
lft = 3 ; rgt = 5 ; ppiv = 3 ; i = 4 ; j = 5 do 4<=5
        if a[4] <= a[3] -> no
        if a[5] >= a[3] -> si -> j = j-1 -> j = 4
        if a[4] > a[3] ^ a[5] < a[3] -> no
lft = 3 ; rgt = 5 ; ppiv = 3 ; i = 4 ; j = 4 do 4<=4
        if a[4] <= a[3] -> no
        if a[4] >= a[3] -> si -> j = j-1 -> j = 3
        if a[4] > a[3] ^ a[4] < a[3] -> no
lft = 2 ; rgt = 4 ; ppiv = 3 ; i = 5 ; j = 4 do 4<=3 {TERMINA CICLO}
[3,4,5]
swap (a,3,3)
ppiv := j -> ppiv = 3
[3,4,5]
quick_sort_rec(a,4,5) [4,5]
        lft=4 ; rgt =5 ; if rgt(5) > lft(4)
        partition(a,4,5,ppiv) -> ppiv = 4
        quick_sort_rec(a,4,ppiv-1) -> quick_sort_rec(a,4,3)
        quick_sort_rec(a,ppiv+1,4)  -> quick_sort_rec(a,5,5)
*partition(a,4,5,ppiv)
lft = 4 ; rgt = 5 ; ppiv = 4 ; i = 5 ; j = 5 do 5<=5
        if a[5] <= a[4] -> no
        if a[5] >= a[4] -> si -> j = j-1 -> j = 4
        if a[5] > a[4] ^ a[5] < a[4] -> no
lft = 4 ; rgt = 5 ; ppiv = 4 ; i = 5 ; j = 4 do 5<=4 {TERMINA CICLO}
[4,5]
```

swap (a,4,4)
ppiv := j -> ppiv = 4
[4,5]
quick_sort_rec(a,4,3) [4] -> se asume ord -> cond rgt > lft
quick_sort_rec(a,5,5) [5] -> se asume ord -> cond rgt > lft
se termina la llamada -> quick_sort_rec(a,ppiv+1,5) -> [2,3,4,5]
quick_sort_rec(a,1,0) [1] -> se asume ordenado
RESULTADO DE TODO => [1,2,3,4,5]

4. Escribí una variante del procedimiento partition que en vez de tomar el primer elemento del segmento $a[izq, der]$ como pivot, elige el valor intermedio entre el primero, el último y el que se encuentra en medio del segmento. Es decir, si el primer valor es 4, el que se encuentra en el medio es 20 y el último es 10, el algoritmos deberá elegir como pivot al último.



```
Proc partition_z (in/out a:array [1..2 ] of int , in lft,rgt,mid : nat, out ppiv : nat)
   var i,j : nat
   if (a[lft] < a[mid] < a[rgt] ) then
      swap(a,1,mid)
      ppiv:=lft
   else
      swap(a,1,rgt)
      ppiv:=lft
   fi
   i:=lft+1
   j:=rgt
   do..MISMOCODE DE PARTITION..od
end proc
```

-> algunas modificaciones a tener en cuenta:
-el argumento "mid" no deberia de tenerlo como entrada, deberia definirse.

```
proc partition_z (in/out a : array [1..2 ] of int , in lft,rgt : nat, out ppiv : nat)
        var i,j, mid : nat
        mid:= (rgt+lft)/2
        if (a[lft] < a[mid] < a[rgt] ) then
                swap(a,1,mid)
                ppiv:=lft
        else
                swap(a,1,rgt)
                ppiv:=lft
        fi
        i:=lft+1
        j:=rgt
        do..MISMOCODE DE PARTITION..od
end proc
```

5. 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 partition del quick_sort deja al pivot en su lugar correcto.

```
fun encuentra_k (in/out a : array [1...N] of int , in lft,rgt, k : nat) ret m : int
  var ppiv,i,j : nat
  i:=lft
  j:=rgt
  ppiv := partition(a,i,j,ppiv)
  do ppiv!=k   ->
    if  ppiv<=k -> encuentra_k(a,ppiv+1,rgt,k)
        ppiv>=k -> encuentra_k(a,lft,ppiv-1,k)
    fi
    m:=a[ppiv]
  od
end proc
```