Presentación Proyecto 1

Algoritmos y Estructuras de Datos II

Leonardo Rodríguez

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[4, 1, 0, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 4, 6, 5, 2]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 6, 5, 4]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 6, 5, 4]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 6, 5, 4]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 6, 5, 4]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 6, 5, 4]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 4, 5, 6]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 4, 5, 6]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 4, 5, 6]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 4, 5, 6]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 4, 5, 6]$$

Selection-sort

```
proc selection_sort (in/out a: array[1..n] of T)
        var minp: nat
        for i:= 1 to n-1 do
            minp:= min_pos_from(a,i)
            swap(a,i,minp)
        od
end proc

fun min_pos_from (a: array[1..n] of T, i: nat) ret minp: nat
      minp:= i
      for j := i+1 to n do if a[j] < a[minp] then minp:= j fi
      od
end fun
```

$$[0, 1, 2, 4, 5, 6]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[4, 1, 0, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[4, 1, 0, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[4, 1, 0, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[1, 4, 0, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[1, 4, 0, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[1, 4, 0, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[1, 0, 4, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 6, 5, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 5, 6, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 5, 6, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 5, 6, 2]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 5, 2, 6]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 4, 2, 5, 6]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 2, 4, 5, 6]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 2, 4, 5, 6]$$

Insertion-sort

```
proc insertion_sort (in/out a: array[1..n] of T)
        for i:= 2 to n do
            insert(a,i)
        od
end proc

proc insert (in/out a: array[1..n] of T, in i: nat)
        var j: nat
        j:= i
        do j > 1 ∧ a[j] < a[j − 1] → swap(a,j-1,j)
                                     j:= j-1
        od
end proc
```

$$[0, 1, 2, 4, 5, 6]$$

Quick-sort

```
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
```

$$[4, 1, 0, 6, 5, 2]$$

Quick-sort

```
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
```

$$[4, \overline{1}, 0, 6, 5, 2]$$

Quick-sort

```
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
```

$$[4, 1, \overline{0}, 6, 5, 2]$$

Quick-sort

```
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
```

$$[4, 1, 0, \overline{6}, 5, 2]$$

Quick-sort

```
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
```

$$[4, 1, 0, \overline{2}, 5, 6]$$

Quick-sort

```
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
```

$$[4, 1, 0, 2, \overline{5}, 6]$$

Quick-sort

```
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
```

$$[4, 1, 0, \underline{2}, \overline{5}, 6]$$

Quick-sort

```
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
```

$$[2, 1, 0, 4, \overline{5}, 6]$$

Quick-sort

```
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
```

$$[2, 1, 0], 4, [5, 6]$$

Quick-sort

```
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
```

$$[2, 1, 0], 4, [5, 6]$$

Quick-sort

```
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
```

$$[2, \overline{1}, \underline{0}], 4, [5, 6]$$

Quick-sort

```
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
```

$$[2, 1, \overline{0}], 4, [5, 6]$$

Quick-sort

```
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
```

$$[2, 1, 0], \overline{4}, [5, 6]$$

Quick-sort

```
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
```

$$[0, 1, \underline{2}], \overline{4}, [5, 6]$$

Quick-sort

```
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
```

$$[0, 1], 2, 4, [5, 6]$$

Quick-sort

```
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
```

$$[0, 1], 2, 4, [5, 6]$$

Quick-sort

```
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
```

$$[0, \overline{1}], 2, 4, [5, 6]$$

Quick-sort

```
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
```

$$[\underline{0}, \overline{1}], 2, 4, [5, 6]$$

Quick-sort

```
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
```

$$0, [1], 2, 4, [5, 6]$$

Quick-sort

```
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
```

$$0, 1, 2, 4, [5, 6]$$

Quick-sort

```
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
```

$$0, 1, 2, 4, [5, 6]$$

Quick-sort

```
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
```

$$0, 1, 2, 4, [5, \overline{6}]$$

Quick-sort

```
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
```

$$0, 1, 2, 4, [5, 6]$$

Quick-sort

```
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
```

$$0, 1, 2, 4, 5, [6]$$

Quick-sort

```
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
```

$$0, 1, 2, 4, 5, 6$$

Quick-sort

```
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
```

$$0, 1, 2, 4, 5, 6$$

Quick-sort

- El *pivote* se elige siempre a la izquierda.
- A veces eso puede ser perjudicial:

$$[4, \overline{3}, 2, 1, \underline{0}]$$

- Sólo puedo mover el rojo, hasta que intercambio.
- Nunca divide el arreglo a la mitad, y por ello pierdo eficiencia.
- Una buena idea es elegir un elemento al azar e intercambiarlo por la primera posición.

Proyecto 1

- Implementar los algoritmos en `sort.c`

```
unsigned int selection_sort(int a[], unsigned int length);
unsigned int insertion_sort(int a[], unsigned int length);
unsigned int quick_sort(int a[], unsigned int length);
unsigned int rand_quick_sort(int a[], unsigned int length);
```

- Toma un arreglo y su longitud.
- Ordena el arreglo con el algoritmo correspondiente.
- Devuelve la cantidad de comparaciones.
- Probar los algoritmos con la interfaz que les dimos hecha.

Pasos a seguir:

1. Repasar C de Algoritmos I.
2. Leer cuidadosamente el enunciado del proyecto.
3. Implementar los algoritmos siguiendo el pseudo-código, ignorando por ahora la cantidad de comparaciones (devolver siempre 0).
4. Una vez que comprueben que funciona, agregar la lógica correspondiente al conteo de comparaciones.
5. Tienen disponible hecho un programa llamado *compare* que ejecuta todos los algoritmos sobre el mismo arreglo y muestra la cantidad de comparaciones realizadas.