



Práctico

Vector

```
procedure agregarAlFinal (var v:vector; var diml; num:integer);
```

```
begin
```

if $(diml + 1) \leq dimf$ then begin → verificamos que haya lugar

diml := diml + 1 → aumentamos la dimensión.

v[diml] := num → guardamos el valor.
end;

```
end;
```

```
procedure insertarEnVector (var v:vector, var diml:integer, pos: integer, num: integer);
```

```
var i:integer
```

```
begin
```

if $((diml + 1) \leq dimf)$ and $(pos \leq dimf)$ then

* verificamos si hay espacio en la estructura y que la posición sea válida.

for i:=diml down to pos do

v[i+1] := v[i]; desde la última posición

v[pos] := num; hacemos lugar, corriendo todos los elementos una posición.

diml := diml + 1,

asignamos el valor

y aumentamos la dimensión lógica.

```
end;
```

```
end;
```

```
procedure eliminarVector (var v:vector, var diml:integer, pos:integer);
```

```
var i:integer
```

```
begin
```

if $(pos > 1)$ and $(pos \leq diml)$ then begin → verificamos que sea una posición válida.

for i:= pos to (diml - 1) do

→ desde la posición a borrar hasta la diml - 1, pisamos los elementos.

v[i] := v[i+1];

diml := diml - 1;

```
end;
```

```
end;
```

incrementamos
la dimensión lógica.

Búsquedas

```
procedure búsqueda Vector Desordenado (v:vector, diml:integer, numm:integer, var ok:boolean);  
var i:integer  
begin  
    i:=1; ok:=false;  
    while (i <= diml) and (ok = false) do begin  
        if (v[i] = numm) then  
            ok := true  
        i:= i+1  
    end  
end
```

Vector Ordenado

```
procedure búsqueda Mejorada (v:vector, diml:integer, numm:integer, var ok:boolean);  
var i:integer  
begin  
    i:=1; ok:=false;  
    while (i <= diml) and (v[i] <= numm) do  
        i:= i+1;  
    if (pos <= diml) and (v[pos] = numm) then  
        ok := true;  
end
```

procedure **busqueda Dicotómica** (var v:vector, diml:integer, num: integer, var ok:boolean);

var

 inicio, medio, ultimo : integer;

begin

 inicio := 1;

 ultimo := diml;

 medio := (inicio + ultimo) div 2;

 ok := false;

 while (inicio <= ultimo) and (num <= v[medio]) do begin

 mientras no se termine la estructura y el valor del medio sea \leftrightarrow del buscado.

 if (num < v[medio]) then Si es menor

 ultimo := medio - 1

 else

 inicio := medio + 1 \rightarrow SINO, es mayor.

 medio := (inicio + ultimo) div 2 \rightarrow calculamos el nuevo medio.

 end

 if (inicio <= ult) and (num = v[medio]) then

 OK := True

end

pseudo Código \rightarrow . calculamos el elemento que está en el medio.

. si el elemento es el que busco, la búsqueda terminó.

. si no es el elemento que busco, entonces

\hookrightarrow comparo con el elemento del medio y elijo el extremo que más me convenga para seguir la búsqueda.

Ordenación de vectores

1. Selección \rightarrow Sencillo, N^2

2. Intercambio \rightarrow N^2 ejecutari.

3. inserción Si los datos ya están ordenados, el algoritmo solo hace comparaciones.

procedure **ordenar Inserción** (var v:vector);

var

 i, j, actual : integer

begin

 for i:=2 to dimv do begin

 actual := v[i]

 j := i - 1

```

while ( $j > 0$ ) and ( $v[j] < \text{actual}$ ) do begin
     $v[j+1] := v[j]$ 
     $j := j - 1$ 
end
 $v[j+1] := \text{actual};$ 
end

```

procedure Selección Ordenación (var v:vector);

var aux, i, j, p:integer

begin for $i := 1$ to ($\text{dimf} - 1$) do begin
 $p := i$
 for $j := i + 1$ to dimf do
 if ($v[j] < v[p]$) then → si el siguiente es mayor, los ordena.
 $p := j$

$\text{aux} := v[p]$ → guardo elemento para no perderlo

$v[p] := v[i]$ } el elemento z pasa al del
 $v[i] := \text{aux}$ elemento i .

end end

Listas

procedure **agregar Adelante** (var L:Lista , d:dato)

var nuevo : lista

begin

new(nuevo)

nuevo^.dato := d

nuevo^.sig := L → engancha con el primero

L:=nuevo → pasa a ser el primer elemento.

end

procedure **agregar Atrás** (var L,ult:lista , d: dato)

var

nuevo : lista

begin

new(nuevo)

nuevo^.dato := d

if (L = nil) then begin → si la lista está
vacía, el nuevo
es el primero y el
último.

L := nuevo

Ult := nuevo

else

Ult^.sig := nuevo enganchamos

Ult := nuevo el anterior último

y lo actualizamos.

end

procedure agregarOrdenado (var l:lista , d: dato)

var
anterior, actual, nuevo : lista

begin

nuevo (nuevo)

nuevo^.dato : d

anterior := L

actual := L

while (L <= nil) and (actual^.dato < dato) do begin

anterior := actual

actual := actual^.sig

end

if (anterior = actual) then → la lista está vacía.

L := nuevo

else

anterior^.sig := nuevo

nuevo^.sig := actual

} engancha en el
medio.

end