ALGORITMOS PASCAL

Listas

Recorrer lista

```
procedure recorrerLista (l : lista);
begin
    while l <> nil do
        begin
            write(l^.elem);
            l := l^.sig;
        end;
end;
```

Agregar Adelante

```
procedure agregarAdelante (var l : lista ; e : elemento);
var
    nue : lista;
begin
    new(nue);
    nue^.elem := e;
    nue^.sig := l;
    l := nue;
end;
```

Agregar Atrás

```
procedure agregarAtras (var l,ult : lista ; e : elemento);
var
    nue : lista;
begin
    new(nue);
    nue^.elem := e;
    nue^.sig := nil;
```

```
    if (l = nil) then
        l := nue
    else
        ult^.sig := nue;
    ult := nue;
end;
```

Búsqueda desordenada

```
function buscar (l : lista ; valor : integer): boolean;
var
    encontre : boolean;
begin
    encontre := false;
    while ((l <> nil) and (encontre = false) do
        begin
            if (l^.elem = valor) then
                encontre := true
            else
                l := l^.sig;
        end;
    buscar := encontre;
end;
```

Búsqueda ordenada

```
function buscar (l : lista ; valor : integer): boolean;
var
    encontre : boolean;
begin
    encontre := false;
    while ((l <> nil) and (l^.elem < valor)) do
        l := l^.sig;
    if ((l <> nil) and ( l^.elem = valor)) then
        encontre := true
    buscar := encontre;
end;
```

Eliminar una vez

```
procedure eliminarUnaVez (var l : lista ; valor : integer);
var
    act,ant : lista;
begin
    act := l;
    while ((act <> nil) and (act^.elem <> valor)) do
        begin
            ant := act;
            act := act^.sig;
        end;
    if (act <> nil) then
        begin
            if (act = l) then
                l := l^.sig
            else
                ant^.sig := act^.sig;
            dispose(act);
        end;
end;
```

Eliminar varias veces

```
procedure eliminarVariasVeces(var l: lista; valor: intege
r);
var
  act, ant: lista;
begin
  act := l;
  while (act <> nil) do
  begin
    if (act^.elem <> valor) then
    begin
      ant := act;
      act := act^.sig;
    end
    else
```

```
    begin
      if (act = l) then
        l := l^.sig
      else
        ant^.sig := act^.sig;
      dispose(act);  // Libera la memoria del nodo eliminad
o
      act := ant^.sig;  // Avanza al siguiente nodo
    end;
  end;
end;
```

Eliminar Ordenado si esta una vez

```
procedure eliminarOrdenado (var l : lista ; valor : intege
r);
var
    ant,act : lista;
begin
    act := l;
    ant := nil;
    while ((act <> nil) and (act^.elem < valor)) do
        begin
            ant := act;
            act := act^.sig;
        end;
    if (act <> nil) and (act^.elem = valor) then
        begin
            if ( act = l) then
                l := l^.sig
            else
                ant^.sig := act^.sig;
            dispose (act);
        end;
end;
```

Insertar Ordenado

```
procedure insertarOrdenado (var l : lista; e : elemento);
var
    nue,ant,act : lista;
begin
    new(nue);
    nue^.elem := e;
    ant := l;
    act := l;
    while (act <> nil) and (e.num < act^.dato.num) do
        begin
            ant := act;
            act := act^.sig;
        end;
    if (act = ant) then
        l := nue
    else
        ant^.sig := nue;
    nue^.sig := act;
end;
```

Vectores

Agregar

```
procedure agregar (var v : vector; var dimL : integer ; val
or : integer; var ok : boolean);
begin
    ok := false;
    if ((dimL + 1) <= dimF)) then
        begin
            ok := true;
            dimL := dimL + 1;
            v[dimL] := dimL;
        end;
end;
```

Insertar

```
procedure insertar (var v : vector; var dimL : integer; var
pude : boolean;
                        num,pos : integer);
var
    i : integer;
begin
    pude := false;
    if ((dimL + 1) <= fisica) and (pos >= 1) and (pos <= di
mL)) then
    begin
        for i := dimL downto pos do
            v[i+1] := v[i];
        pude := true;
        v[pos] := valor;
        dimL := dimL + 1;
    end;
end;
```

Eliminar

```
Procedure eliminar (var v : vector ; var dimL : integer; va
r ok : boolean; pos : integer);
var
    i : integer;
begin
    ok := false;
    if ((pos >= 1) and (pos <= dimL)) then
        begin
            for i := pos to dimL do
                v[i] := v [i+1];
            pude := true;
            dimL := dimL - 1;
        end;
end;
```

Búsqueda

```
function busqueda (v : vector ; dimL : integer ; valor : in
teger): boolean;
var
    pos : integer;
    esta : boolean;
begin
    pos := 1;
    esta := false;
    while ((pos <= dimL) and (esta = false)) do
        begin
            if (v[pos] = valor) then
                esta := true
            else
                pos := pos + 1;
        end;
        busqueda := esta;
end;
```

Búsqueda ordenada

```
function busquedaOrdenada (v : vector ; dimL : integer ; va
lor : integer) : boolean;
var
    pos : integer;
begin
    pos := 1;
    while ((pos <= dimL) and (v[pos] < valor)) do
        pos := pos + 1;
    if ((pos <= dimL) and (v[pos] = valor))then
        busquedaOrdenada := true
    else
        busquedaOrdenada := false;
end;
```

Búsqueda dicotómica

```
function dicotomica (v : vector ; dimL, valor : integer): b
oolean;
var
    pri,ult,medio : integer;
    ok : boolean;
begin
    ok := false;
    pri := 1;
    ult := dimL;
    medio := (pri + ult) DIV 2;
    while ((pri <= ult) and (valor <> v[medio])) do
        begin
            if (valor < v[medio]) then
                ult := medio - 1
            else
                pri := medio + 1;
            medio := (pri + ult) DIV 2;
        end;
    if (pri <= ult) and (valor = v[medio]) then
        ok := true;
    dicotomica := ok;
end;
```

Ordenar Selección

```
procedure ordenarSeleccion (var v : vector; dimL : integer);
var
    i,j,pos : integer;
    item : tipoElem;
begin
    for i := 1 to dimL - 1 do
    begin
        pos := i;
        for j := i + 1 to dimL do
            if v[j] < v[pos] then
                pos := j;
        item := v[pos];
```

```
        v[pos]:= v[i];
        v[i] := item;
    end;
end;
```

Ordenar Inserción

```
procedure ordenarInsercion (var v : vector; dimL : integer);
 var
    i,j : integer;
    actual : tipoElem;
 begin
    for i := 2 to dimL do //supone que el 1er ya está ordenad
    begin
        actual := v[i];
        j := i - 1;
        while (j > 0) and (v[j] > actual) do
        begin
            v[j+1] := v[j];                //corrimiento
            j := j -1;
        end;
        v[j+1] := actual;              //pone donde debe ir
    end;
 end;
```