## 1. Estructuras de Datos

### 1.1 Arreglos

**Definición (copia fiel):** > Un arreglo es una estructura de datos compuesta que permite acceder a cada componente por una variable índice. Dicho índice da la posición del componente dentro de la estructura de datos. La estructura arreglo se almacena en posiciones contiguas de memoria.

**Ejemplo:**

Type arregloEntero = array [1..20] of integer;  
Var v: arregloEntero;

### 1.2 Listas

**Definición (copia fiel):** > Una lista es una estructura de datos lineal compuesta por nodos. Cada nodo de la lista posee el dato que almacena la lista y la dirección del siguiente nodo. Toda lista puede recorrerse a partir de su primer elemento. Los elementos no necesariamente están en posiciones contiguas de memoria. Para generar nuevos elementos en la lista, o eliminar alguno se deben utilizar las operaciones de new y dispose respectivamente.

**Ejemplo:**

Type  
 lista = ^nodo;  
 nodo = record  
 dato: integer;  
 sig: lista;  
 end;  
Var L: lista;

## 2. Ordenación

**Definiciones generales (copia fiel):** > CONSIDERACIONES al momento de implementar un algoritmo de Ordenación: Tiempo de ejecución. Facilidad para la escritura del mismo. Memoria utilizada en su ejecución. Complejidad de las estructuras auxiliares que necesite. Requiere el mismo tiempo si los datos ya están ordenados, si están al azar, si se encuentran en el orden exactamente inverso al que yo los quiero tener.

### 2.1 Selección

**Definición (copia fiel):** > Este algoritmo consta de N vueltas, donde N es la dimensión lógica del arreglo. En la primera Vuelta, se recorre todo el arreglo desde la posición 1 hasta el final (dimL) y se guarda en qué posición se encuentra el elemento más chico del arreglo. Al terminar el recorrido se intercambia el elemento de la posición 1 con el elemento ubicado en la posición en la cual se encontró el menor valor. Esto se repite hasta recorrer todo el arreglo, dando así un total de N vueltas (N = dimensión lógica del arreglo - 1), ya que el último elemento en la última vuelta ya está ubicado en su posición.

**Ejemplo de ejecución (Vuelta 1):**

Arreglo inicial: [23, 1, 100, 4, 7]  
Mínimo encontrado = 1 en posición 2  
Intercambio: posición 1 ⇄ posición 2  
Resultado: [1, 23, 100, 4, 7]

**Pseudocódigo (Pascal):**

Procedure seleccion(var v: array of integer; dimLog: integer);  
Var i, j, pos, item: integer;  
Begin  
 for i := 1 to dimLog-1 do begin  
 pos := i;  
 for j := i+1 to dimLog do  
 if v[j] < v[pos] then pos := j;  
 item := v[pos];  
 v[pos] := v[i];  
 v[i] := item;  
 end;  
End;

### 2.2 Inserción

**Definición (copia fiel):** > La idea general de este algoritmo es ir considerando subconjuntos de datos del vector e ir ordenándolos. En la primera Vuelta, se trabaja el subconjunto formado por el primer elemento del arreglo que obviamente se considera ordenado. En la segunda Vuelta, se trabaja el subconjunto formado por el primer y segundo elemento del arreglo y se ordena ese subconjunto de manera de encontrar en qué posición debe estar el segundo elemento para que el arreglo siga ordenado. Esto se repite hasta recorrer todo el arreglo, dando así un total de N vueltas.

**Ejemplo de ejecución (subconjunto creciente):**

Arreglo inicial: [3, 5, 2, 1, 4, 6]  
Vuelta 2 (inserción de 5): [3, 5]  
Vuelta 3 (inserción de 2): desplazar [3,5] → [2,3,5]  
...  
Arreglo final: [1,2,3,4,5,6]

**Pseudocódigo (Pascal):**

Procedure insercion(var v: array of integer; dimLog: integer);  
Var i, j, actual: integer;  
Begin  
 for i := 2 to dimLog do begin  
 actual := v[i];  
 j := i - 1;  
 while (j > 0) and (v[j] > actual) do begin  
 v[j+1] := v[j];  
 j := j - 1;  
 end;  
 v[j+1] := actual;  
 end;  
End;

## 3. Recursión

**Definición (copia fiel):** > La recursividad es una técnica de resolución de problemas que consiste en dividir un problema en instancias más pequeñas del mismo problema (también llamados subproblemas) hasta que obtengamos un subproblema lo suficientemente pequeño que tenga una solución trivial o directa. La recursividad consiste en resolver un problema por medio de un módulo (procedimientos o funciones) que se llama a sí mismo, evitando el uso de bucles y otros iteradores. Cuando el problema se va achicando llega a un punto que no puede achicarse más, esa instancia se denomina caso base.

### 3.1 Ejemplo: Factorial

**Solución recursiva (copia fiel):**

Function potencia(x, n: integer): integer;  
Begin  
 if n = 0 then potencia := 1  
 else if n = 1 then potencia := x  
 else potencia := x \* potencia(x, n-1);  
End;

**Ejemplo de llamada:**

potencia(4,3)  
→ 4 \* potencia(4,2)  
→ 4 \* (4 \* potencia(4,1))  
→ 4 \* (4 \* 4)  
= 64

### 3.2 Ejemplo: Impresión de lista enlazada

**Solución recursiva (copia fiel):**

Procedure imprimir(pri: lista);  
Begin  
 if pri <> nil then begin  
 write(pri^.dato);  
 imprimir(pri^.sig);  
 end;  
End;

## 4. Árboles

### 4.1 Definición y declaración

**Definición (copia fiel):** > Es una estructura de datos jerárquica. Está formada por nodos. El nodo principal del árbol se denomina raíz y los nodos que no tienen hijos se denominan hojas del árbol.

**Declaración en Pascal:**

Type  
 arbol = ^nodo;  
 nodo = record  
 dato: integer;  
 HI, HD: arbol;  
 end;  
Var a: arbol;

### 4.2 Árbol Binario de Búsqueda (ABB)

**Definición (copia fiel):** > Un árbol binario de búsqueda (ABB) agrega los elementos por sus hojas. Dichos elementos quedan ordenados (todos por el mismo criterio). Esta operación lleva un tiempo de ejecución de O(log n).

### 4.3 Operaciones básicas

#### Creación y carga de datos

**Procedimiento agregar (copia fiel):**

Procedure agregar(var a: arbol; num: integer);  
Begin  
 if a = nil then begin  
 new(a);  
 a^.dato := num;  
 a^.HI := nil;  
 a^.HD := nil;  
 end  
 else if num <= a^.dato then  
 agregar(a^.HI, num)  
 else  
 agregar(a^.HD, num);  
End;

#### Recorridos

* **EnOrden:**

Procedure enOrden(a: arbol);  
Begin  
 if a <> nil then begin  
 enOrden(a^.HI);  
 write(a^.dato);  
 enOrden(a^.HD);  
 end;  
End;

* **PreOrden:**

Procedure preOrden(a: arbol);  
Begin  
 if a <> nil then begin  
 write(a^.dato);  
 preOrden(a^.HI);  
 preOrden(a^.HD);  
 end;  
End;

* **PostOrden:**

Procedure postOrden(a: arbol);  
Begin  
 if a <> nil then begin  
 postOrden(a^.HI);  
 postOrden(a^.HD);  
 write(a^.dato);  
 end;  
End;

#### Búsqueda

* **Buscar booleano (copia fiel):**

Function buscar(a: arbol; x: integer): boolean;  
Begin  
 if a = nil then buscar := false  
 else if a^.dato = x then buscar := true  
 else if x > a^.dato then buscar := buscar(a^.HD, x)  
 else buscar := buscar(a^.HI, x);  
End;

* **Buscar nodo:**

Function buscarNodo(a: arbol; x: integer): arbol;  
Begin  
 if a = nil then buscarNodo := nil  
 else if a^.dato = x then buscarNodo := a  
 else if x > a^.dato then buscarNodo := buscarNodo(a^.HD, x)  
 else buscarNodo := buscarNodo(a^.HI, x);  
End;