Segunda Unidad

Estructuras de datos lineales estáticas

Sumario

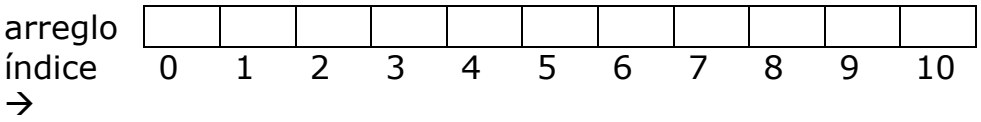
Comprende y sintetiza los conceptos asociados a los arreglos. Se abstrae, Identifica y clasifica las características y acciones básicas (primitivas) relacionadas a los arreglos, además se analiza y compara los algoritmos de ordenación y búsqueda en este tipo de estructura.

- Arreglos
- Métodos de ordenación y búsqueda

Lección 3

Arreglos

Un arreglo es un conjunto finito de elementos, todos del mismo tipo, que son representados mediante una variable del mismo nombre, y se registran en posiciones consecutivas de memoria. Las posiciones consecutivas de memoria se denomina celdas o casillas, cada una de las casillas tiene asociado un número llamado indice o dirección, que se enumera consecutivamente 0, 1, 2, 3,…., n-1.
El tamaño o dimensión de un arreglo es el número de celdas que lo conforman. Por ejemplo, un arreglo de dimensión 10.



En cada una de las casillas se puede almacenar un elemento del conjunto finito.

3.1. Operaciones con Arreglos

Las operaciones que se pueden realizar con arreglos para resolver un problema son los siguientes:

- Lectura/ escritura
- Asignación
- Actualización(inserción, eliminación, modificación)
- Recorrido (acceso secuencial)
- Ordenación
- Búsqueda

3.2. Tipos de Arreglos

Los arreglos pueden ser unidimensionales o multidimensionales

3.2.1. Arreglos Unidimensionales o vectores

"La forma mas simple de arreglo es un arreglo unidimensional que puede definirse de manera abstracta como un conjunto finito ordenado de elementos homogéneos. Por "finito" entendemos que hay un numero especifico de elementos en el arreglo; numero que puede ser grande o pequeño, pero debe existir" (Tenenbaum, 1993, p.14).

3.2.1.1. Declaración de un vector

Igual que sucede con otras variables, es necesario declarar un vector antes de utilizarlo, por ejemplo en C++, se especifica el tipo de dato de sus elementos, luego el nombre del vector y después entre corchetes ([ ]) la longitud del vector.

tipo nombreDelVector[tamaño];

Donde:

tipo : tipo de dato del vector puede ser int, double, etc.
nombreDelVector : nombre identificador de la variable
tamaño : número entero que indica la cantidad de elementos del vector.

Por ejemplo, para declarar un vector *x* de tipo entero de tamaño 10:

```
int x[10];
```

3.2.1.2. Acceso a los elementos de un vector

Un índice describe la posición de un elemento dentro de un vector, un índice se enumera desde *0* hasta *tamaño-1*, siendo tamaño el número de elementos del vector.

Para acceder a un elemento del vector, se especifica el nombre del vector y un índice encerrado por corchetes, es decir:

nombreDelVector [índice]

Por ejemplo: Un vector entero *x* de 10 elementos:

x[n]=	5	9	7	4	6	10	12	13	1	26
índice →	0	1	2	3	4	5	6	7	8	9

Primer elemento del vector: x[0] = 5
Segundo elemento del vector: x[1] = 9
Tercer elemento del vector: x[2] = 7
.
.
.
Ultimo elemento del vector: x[9] = 26

- Un elemento de un vector se puede utilizar de la misma forma que cualquier otra variable. Por ejemplo, las siguientes operaciones son válidas:

```
c[3] = 10          // Asignación de un valor
z = x[5]
a = x[3] + x[5]
c[k] = a[k] * b[k]
```

- Los índices pueden alterar o cambiar su valor como resultado de hacer una operación aritmética. Por ejemplo.

```
i = 5
x[i + 4] = 50              // asigna el valor de 50 a x[9]
```

- El índice puede tomar un valor específico, que al ser utilizado en el vector nos da su valor (contenido). Por ejemplo:

```
i = 8
p=x[i]       // nos da el valor (contenido) del elemento x[8] y lo asigna a p
```

3.2.2. Arreglos bidimensionales o matrices

Llamados también *matrices* o *tablas*. Es un conjunto de elementos del mismo tipo de datos que se almacena bajo un mismo nombre, se necesita especificar dos índices para identificar a cada elemento del arreglo.

3.2.2.1. Declaración de una matriz

Para declarar una matriz en C++, es necesario especificar el tipo datos de sus elementos, luego el nombre de la matriz y después entre corchetes ([ ][ ]) la longitud de la matriz. El primer par de corchetes contiene el índice que representa a la fila y el segundo par de corchetes contiene el índice que representa a la columna. Su sintaxis es:

tipo nombreDeLaMatriz[fila][columna];

Donde:

tipo	:tipo de dato de la matriz que puede ser int, double,etc.
nombreDeLaMatriz	:nombre identificador de la variable.
fila	:número entero que indica número de filas de la matriz.

columna :número entero que indica la número de columnas
de la matriz.

tipo tipo de dato de la matriz que puede ser int, double, etc.
nombreDeLaMatriz nombre identificador de la variable
fila número entero que indica número de filas de la matriz
columna número entero que indica la número de columnas de la
matriz

Ejemplo:

```
int x[4][5];
```

x[4][5] =

	0	1	2	3	4
0	30	8	2	10	3
1	7	5	18	22	6
2	1	9	11	14	4
3	24	17	6	20	12

3.2.2.2. Acceso a los elementos de una matriz

Se puede acceder a los elementos de una matriz, especificando el nombre de la matriz y dos índices([][]), el primero indica la fila y el segundo la columna, es decir:

```
nombreDeLaMatriz[fila][columna]
```

Por ejemplo si tenemos un matriz de 3 filas y 4 columnas.

X[3][4] =

	0	1	2	3
0	10	13	20	7
1	2	3	18	6
2	1	9	11	8

Primer elemento de la matriz: x[0][0] = 10
Segundo elemento de la matriz: x[0][1] = 13
Tercer elemento de la matriz: x[0][2] = 20
.
.
.
Ultimo elemento: x[2][3] = 8

3.2.3. Arreglos Multidimensionales

Son los arreglos con tres o más dimensiones. La sintaxis, de un arreglo de tres dimensiones es la siguiente:

tipo nombreArreglo[fila][columna][profundidad];

Por ejemplo:

```
int a[5][4][2];
```

los datos pueden ordenarse de la siguiente manera como se observa en la figura de abajo.

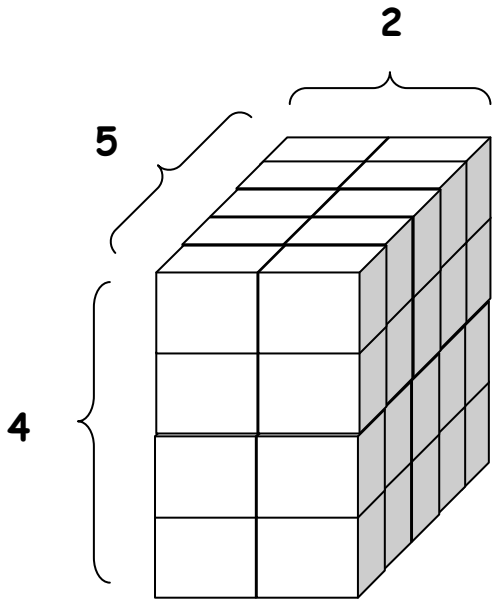


Figura 3.1. Arreglo multidimensional

3.3. Ventajas y desventajas de usar arreglos

Ventajas

- El costo de acceso a un elemento del vector es constante, no hay diferencias de costo entre accesar al primer elemento, al último o a cualquier elemento del vector.
- Se puede utilizar para implementar otras estructuras de datos, como pilas, colas, arboles, etc.

Desventajas

- En primer lugar hay que definir el tamaño del arreglo, lo que puede ocasionar un desperdicio de espacio en memoria, si se definen vectores muy grandes para contener pocos elementos.
- El tamaño de un vector es fijo. Si el espacio reservado es menor que el necesario puede ocurrir problemas si no se conoce previamente el máximo de elementos a almacenar.
- Es muy lenta la busqueda de un elemento en un arreglo desordenado
- Es muy lento el insertar elementos de manera ordenada.

3.4. Primitivas

primitiva	FILAS: para la fila F	ARREGLOS: para el arreglo T con número de elemento N
Inicio	Inicio(F)	i ←0
Leer	Leer(F, val)	val ← T[i]
Avanzar	Avanza con cada lectura	i ← i +1
Verificar final	Ultimo (F)	i < N
escribir	Escribir(F, val)	T [i] ← val
terminar	Cerrar(F)	No existe

3.5. Ejercicios resueltos

Ejemplo 01:

Escriba la especificación del TAD, el algoritmo y su implementación en C++, para registrar en un tercer vector, la concatenación de dos vectores. Por ejemplo:

Entrada:

a[] = | 2 | 9 | 7 | 3 |

b[] = | 8 | 5 |

Salida:

c[]=	2	9	7	3	8	5

Solución:

Especificación del TAD y los algoritmos

Especificación VECTOR
variable
num : entero.
métodos
menu : no retorna valor.
ingresarDatos() : no retorna valor.
mostrarDatos() : no retorna valor.
registrarDatos(a, n, x) : no retorna valor.
concatenarVector(a, n1, b, n2, c, n3) : no retorna valor.
mostrarVector(a,n) : no retorna valor.

Significado
menu muestra las opciones a escoger.
ingresarDatos ingresa datos al vector.
mostrarDatos muestra los datos del vector.
registrarDatos añade *x* del tipo VECTOR al vector *a,* con una cantidad *n* de elementos.
concatenarVector tiene como precondición a los vectores a y b cada uno con su respectivo número de elementos n1, n2, y como postcondición al vector c, con su número de elementos n3.
mostrarVector muestra el contenido del vector *a*, con una cantidad de *n* elementos.
Fin_especificacion

```
Procedimiento concatenarVector (a, n1, b, n2, c, n3)
    // Definir variables
    Entero: i
    Desde i ← 0  hasta i<n1 con incremento 1 Hacer
        c[n3] ← a[i]
        n3 ← n3+1
    Fin_desde
    Desde i ← 0 hasta i<n2 con incremento 1 Hacer
        c[n3] ← b[i]
        n3 ← n3+1
    Fin_desde
Fin_procedimiento
```

Implementación del TAD

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
class VECTOR
{
	int num;
	public:
		void menu()
		{
			cout<< "\n MENU DE OPCIONES \n";
			cout<< "----------------\n" ;
			cout<<"<1> Ingresar vector A \n";
			cout<<"<2> Ingresar vector B \n";
			cout<<"<3> Concatenar vector \n";
			cout<<"<4> Mostrar vector \n";
			cout<<"<5> Salir \n";
		}

		void ingresarDatos()
		{
			fflush(stdin);
			cout<<"\n leer numero   : ";cin>>num;
		}

		void mostrarDatos()
		{
			cout<<num<<setw(5);
		}

		void registrarDatos(VECTOR a[100],int &n, VECTOR x )
		{
			a[n]=x;
			n++;
		}
		void concatenarVector(VECTOR a[50], int n1, VECTOR
b[50],
                                      int n2, VECTOR c[50], int &n3)
		{
			int i;
			for(i=0;i<n1;i++)
			{
				c[n3]=a[i];
				n3++;
			}
			for(i=0;i<n2;i++)
			{
```

```
                        c[n3]=b[i];
                        n3++;
                    }

            }
            void mostrarVector(VECTOR a[50], int n)
            {
                for(int i=0; i<n; i++)
                    a[i].mostrarDatos();
                cout<<"\n";
            }
};

int main()
{
    char opcion;
    VECTOR a[50], b[50], c[50], x;
    int n1=0, n2=0, n3=0;
    do
    {
        x.menu();
        cout<<"\n Ingrese opcion : ";
        opcion=cin.get();
        switch(opcion)
        {
            case '1':
                x.ingresarDatos();
                x.registrarDatos(a,n1,x);
                break;
            case '2':
                x.ingresarDatos();
                x.registrarDatos(b,n2,x);
                break;
            case '3':
                x.concatenarVector(a,n1,b,n2,c,n3);
                cout<<"\n Se concateno exitosamente\n ";
                break;
            case '4':
                cout<<"Vector A:"<<"\n";
                x.mostrarVector(a,n1);
                cout<<"Vector B:"<<"\n";
                x.mostrarVector(b,n2);
                cout<<"Vector C concatenacion "<<"\n";
                x.mostrarVector(c,n3);
                break;
        }
        cin.ignore();
    }
```

```
        while( opcion !='5');
        system("PAUSE");
        return 0;
}
```

Ejemplo 02

Escriba la especificación del TAD, el algoritmo y su implementación en C++, que permita registrar en un tercer vector, la intersección de dos vectores A y B. La intersección esta formada por todos los elementos comunes que pertenecen a ambos vectores. Por ejemplo:

Entrada:

a[] =	3	7	11	15	19

b[] =	4	7	13	15	18	19	20

Salida:

c[]=	7	15	19

Solución:

Especificación del TAD y los algoritmos

Especificación VECTOR

variable

num : entero.

métodos

menu() : no retorna valor.
ingresarDatos() : no retorna valor.
mostrarDatos() : no retorna valor.
registrarDatos(a, n, x) : no retorna valor.
intersectarVector(a, n1, b, n2, c, n3) : no retorna valor.
mostrarVector(a, n) : no retorna valor.

significado

menu muestra las opciones a escoger.
ingresarDatos ingresa datos al vector.
mostrarDatos muestra los datos del vector.
registrarDatos añade *x* del tipo VECTOR al vector *a,* con una cantidad *n* de elementos.
intersectarVector tiene como precondición a los vectores a y b cada uno con su respectivo número de elementos n1, n2, y como postcondición el vector c, con su número de elementos n3.

mostrarVector muestra el contenido del vector a, con una cantidad de n elementos.

Fin_especificacion

```
Procedimiento intersectarVector (a, n1, b, n2, c, n3)
        // Definir variables
        Logico: salir
        Entero: i , j
        Desde i ←0 Hasta i<n1 con incremento 1 Hacer
                salir ← false
                Desde j ← 0 Hasta j<n2 y no salir con incremento 1 Hacer
                        Si (a[i].num =b[j].num) entonces
                           salir ← true
                        Fin_si
                Fin_desde
                Si  (salir = true) entonces
                        c[n3]← a[i]
                        n3 ← n3+ 1
                Fin_si
        Fin_desde
Fin_procedimiento
```

Implementación del TAD

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
class VECTOR
{
        int num;
        public:
        void menu()
        {
                cout<< "\n MENU DE OPCIONES \n";
                cout<< "---------------\n" ;
                cout<<"<1> Ingresar vector A \n";
                cout<<"<2> Ingresar vector B \n";
                cout<<"<3> Intersectar vector \n";
                cout<<"<4> Mostrar  vector\n";
                cout<<"<5> Salir \n";
        }

        void ingresarDatos()
        {
                fflush(stdin);
                cout<<"\n leer numero   : ";cin>>num;
```

```
        }
        void mostrarDatos()
        {
                cout<<num<<setw(5);
        }
        void registrarDatos(VECTOR a[100],int &n, VECTOR x )
        {
                a[n]=x;
                n++;
        }
        void intersectarVector(VECTOR a[50],int n1, VECTOR b[50],int n2,
                                        VECTOR c[50],int &n3)
        {
                bool salir;
                int i=0,j;
                for(i=0;i<n1;i++)
                {
                        salir =false;
                        for(j=0;j<n2 && !salir;j++)
                        {
                                if (a[i].num ==b[j].num )
                                        salir =true;
                        }
                        if (salir==true)
                        {
                                c[n3]=a[i];
                                n3++;
                        }
                }
        }
        void mostrarVector(VECTOR a[50],int n)
        {
                for(int i=0; i<n; i++)
                        a[i].mostrarDatos();
                cout<<"\n";
        }
};
int main()
{
        char opcion;
        VECTOR a[50], b[50], c[50], x;
        int n1=0,n2=0,n3=0;
        do
        {
                x.menu();
                cout<<"\n Ingrese opcion : ";
                opcion=cin.get();
                switch(opcion)
```

```
            {
                  case '1':
                        x.ingresarDatos();
                        x.registrarDatos(a,n1,x);
                        break;
                  case '2':
                        x.ingresarDatos();
                        x.registrarDatos(b,n2,x);
                        break;
                  case '3':
                        x.intersectarVector(a,n1,b,n2,c,n3);
                        cout<<"\n Se intersecto exitosamente\n";
                        break;
                  case '4':
                        cout<<"Vector  A:"<<"\n";
                        x.mostrarVector(a,n1);
                        cout<<"Vector B:"<<"\n";
                        x.mostrarVector(b,n2);
                        cout<<"Vector C interseccion "<<"\n";
                        x.mostrarVector(c,n3);
                        break;
            }
            cin.ignore();
      }
      while( opcion !='5');
      system("PAUSE");
      return 0;
}
```

Ejemplo 03

Escriba la especificación del TAD, el algoritmo y su implementación en C++, para ingresar N números enteros en un vector, calcular la multiplicación de todos los elementos que se encuentren antes del número menor, luego reemplazar todos los números impares por la multiplicación calculada y en caso de no existir ningún reemplazo mostrar un mensaje correspondiente.

Entrada:

a[]=	5	12	10	**2**	18	7	25	16	4

Salida:
Multiplicación: 600

a[]=	**600**	12	10	2	18	**600**	**600**	16	4

Solución:

Especificación del TAD y los algoritmos

Especificación VECTOR
variable
num : entero.
métodos
menu() : no retorna valor.
ingresarDatos() : no retorna valor.
mostrardatos() : no retorna valor.
registrarDatos(a,n, x) : no retorna valor.
menorVector(a, n, mult) : no retorna valor.
reemplazaVector(a,n1,mult) : no retorna valor.
mostrarVector(a,n) : no retorna valor.

significado
menu muestra las opciones a escoger.
ingresarDatos ingresa datos al vector.
mostrarDatos muestra los datos del vector.
registrarDatos añade *x* del tipo VECTOR al vector *a,* con una cantidad *n* de elementos.
menorVector tiene como precondición al vector a y su número de elementos n1, como postcondición a la multiplicación. *reemplazaVector* tiene como precondición al vector a y su número de elementos n1 y postcondición la multiplicación.
mostrarVector muestra el contenido del contenido del vector a, con una cantidad de n elementos.

Fin_especificacion

```
Procedimiento menorVector(a,  n, multi)
    // Definir variables
    Entero: i, m,  posi ←0
    m ← a[0].num
    Desde i ← 0 Hasta i < n con incremento 1 Hacer
        Si (a[i].num < m) entonces
            m ← a[i].num
            posi ← i
        Fin_si
    Fin_desde
    Desde i ←0 Hasta i < posi con incremento 1 Hacer
        multi ← multi * a[i].num
    Fin_desde
Fin_procedimiento

Procedimiento reemplazaVector(a, n1, multi)
    // Definir e inicializar variables
    Entero: i, sw← 0
```

```
        Desde i ←0 Hasta i < n1 con incremento 1 Hacer
                Si (a[i].num  Mod 2 = 1) entonces
                        a[i].num ← multi
                        sw ← 1
                Fin_desde
                Si (sw=0) entonces
                        escribir ("no existe ningun reemplazo ")
                Fin_si
        Fin_desde
Fin_procedimiento
```

Implementacion del TAD

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

class VECTOR
{
        int num;
        public:
                void menu()
                {
                        cout<< "\n MENU DE OPCIONES \n";
                        cout<< "----------------\n" ;
                        cout<<"<1> Ingresar vector A \n";
                        cout<<"<2> Reemplaza Vector  \n";
                        cout<<"<3> Mostrar vector \n";
                        cout<<"<4> Salir \n";
                }
                void ingresarDatos( )
                {
                        fflush(stdin),
                        cout<<"\n leer numero   : ";cin>>num;

                }
                void registrarDatos(VECTOR a[100],int &n, VECTOR x )
                {
                        a[n]=x;
                        n++;
                }
                void menorVector(VECTOR a[50],int n,int &multi)
                {
                        int i,m,posi=0;
                        m = a[0].num;
                        for (i=0;i < n;i++)
                        {
```

```
                        if (a[i].num < m)
                        {
                                m = a[i].num;
                                posi = i;
                        }
                }
                for (i=0;i < posi;i++)
                {
                        multi=multi*a[i].num;

                }
        }
        void reemplazaVector(VECTOR a[50],int n1,int multi)
        {
                // Definir e inicializar variables
                int i,sw=0;
                for (i=0;i < n1;i++)
                {
                        if (a[i].num  % 2 == 1)
                        {
                                a[i].num =multi;
                                sw=1;
                        }
                }
                if (sw==0)
                        cout<< "no existe ningun reemplazo ";
        }
        void mostrarVector(VECTOR a[50],int n)
        {
                for(int i=0; i<n; i++)
                        cout<<a[i].num<<setw(6);
                cout<<"\n";
        }
};
int main()
{
        char opcion;
        VECTOR a[50], x;
        int n1=0, multi=1;
        do
        {
                x.menu();
                cout<<"\n Ingrese opcion : ";
                opcion=cin.get();
                switch(opcion)
                {
                        case '1':
                                x.ingresarDatos();
```

```
                    x.registrarDatos(a,n1,x);
                    break;
                case '2':
                    x.menorVector(a,n1,multi);
                    cout<<"Vector A:"<<"\n";
                    x.mostrarVector(a,n1);
                    x.reemplazaVector(a,n1,multi);
                    cout<<"\n Se reemplazo exitosamente\n";
                    break;
                case '3':
                    cout<<"Vector A:"<<"\n";
                    x.mostrarVector(a,n1);
                    break;
            }
            cin.ignore();
        }
        while( opcion !='4');
        system("PAUSE");
        return 0;
}
```

3.6. Ejercicios propuestos

1. Escriba la especificación del TAD, el algoritmo y su implementación en C++, que permita ingresar N números enteros a un vector y obtenga dos nuevos vectores. Un vector que contenga los números pares y otro vector que contenga los números impares.

   Entrada:

   a[] =

2	3	4	5	6	8	9	10	11

   Salida:

   b[] =

2	4	6	8	10

   c[]=

3	5	9	11

2. Escriba la especificación del TAD, el algoritmo y su implementación en C++, para registrar en un tercer vector, la unión de los vectores A y B. La unión de los vectores A y B es el conjunto formado por todos los elementos que pertenecen a A o a B o a ambos. Por ejemplo:

   Entrada:

   a[] =

0	1	2	3	4	5

   b[] =

5	6	8

Salida:

c[]=	0	1	2	3	4	5	6	8

3. Escriba la especificación del TAD, el algoritmo y su implementación en C++, para ingresa N y M números enteros a dos vectores tal como A y B, luego se traslada todos los números impares tanto del vector A como del vector B a un nuevo vector C. Por ejemplo:

   Entrada:

a[]=	7	3	14	26	9

b[]=	11	18	17	8	5	20	21

   Salida:

c[]=	7	3	9	11	17	5	21

4. Escriba la especificación del TAD, el algoritmo y su implementación en C++, para ingresar N números enteros a un vector, luego invertir sus elementos de la siguiente manera:

   Entrada:

a[]=	3	7	2	8	10

   Salida:

a[]=	10	8	2	7	3

5. Escriba la especificación del TAD, el algoritmo y su implementación en C++, que permita registrar la información a partir de una determinada posición en un segundo vector, esta posición puede variar en cada corrida del programa, pero siempre se mostrara todos los elementos a partir de esa posición. Por ejemplo:

CORRIDAS	ENTRADA		SALIDA
	POSICION	VECTOR	
1	2	3, 8, 12, 5, 9	12, 5, 9, 3, 8
2	1	3, 8, 12, 5, 9	8, 12, 5, 9, 3
3	3	3, 8, 12, 5, 9	5, 9, 3, 8, 12
4	4	3, 8, 12, 5, 9	9, 3 , 8, 12, 5, 9

Se asume que el primer elemento esta en la posición cero.

Lección 4

Métodos de ordenación y búsqueda

La ordenación y la búsqueda son dos procesos que frecuentemente se realizan para resolver problemas por computadora. La operación de búsqueda se realiza para recuperar información normalmente se efectúa sobre elementos ordenados.

Se define la ordenación como una lista A de N elementos, A_0, A_1, A_2,……A_N . "Ordenar significa reagrupar o reorganizar un conjunto de datos u objetos en una secuencia especifica." (Cairó, 2002, p. 319). Clasificar u ordenar significa cambiar de posición estos elementos de tal forma que los elementos quedan de acuerdo con una distribución preestablecida.

- Ascendente $A_0 <= A_1 <= A_2 <=, \ldots <= A_N$
- Descendente $A_0 >= A_1 >= A_2 >=, \ldots >= A_N$

Los métodos de ordenación se clasifican en:

- Ordenación interna (ordenación de vectores) cuando los elementos del vector se encuentran en memoria principal.
- Ordenación externa (ordenación en archivos) cuando los elementos del vector se encuentran en archivos almacenados en dispositivos de almacenamiento secundario como discos, cintas, etcétera.

Un algoritmo de búsqueda es un algoritmo que acepta un argumento A y trata de encontrar un registro cuya llave es A. El algoritmo puede retornar el registro completo o con frecuencia retorna un puntero a ese registro. Es posible que la búsqueda por algún argumento en particular en una tabla no tenga éxito; es decir , que no exista registro en la tabla con ese argumento como llave. En este caso, el algoritmo debe retornar un registro "nulo" o un "puntero nulo" (Tenenbaum, 1986).

4.1. Métodos de ordenación por selección

Implica dos tipos de métodos:

- Método por selección de elementos
- Método de ordenación por conteo

4.1.1. Método de ordenación por selección simple

Sea A el vector con N elementos a clasificar en forma ascendente, se busca el elemento más pequeño del vector, empezando la búsqueda desde A[0] hasta A[N-1], sea A[j] este elemento, entonces se pone A[j] en el lugar de A[0] y A[0] en el lugar de A[j].

Se busca, después el elemento más pequeño del vector, pero se empieza la búsqueda a partir de A[1] hasta A[N-1], sea A[k] este elemento, se pone A[k] en el lugar de A[1] y A[1] y en el lugar de A[k] y así sucesivamente.

Ejemplo:

Sea A el vector a clasificar en forma ascendente.

8	40	26	12	1	4	14
0	1	2	3	4	5	6

- En la primera pasada, comenzamos por el primer elemento A[0] = 8. Luego el menor elemento encontrado desde A[0] hasta A[6] es A[4]=1, entonces cambiamos A[0] con A[4].

1	40	26	12	8	4	14
0	1	2	3	4	5	6

- En la segunda pasada, el egundo elemento es A[1] = 40. Luego el menor elemento encontrado a partir de A[1] es A[5] = 4, entonces cambiamos A[1] con A[5].

1	4	26	12	8	40	14
0	1	2	3	4	5	6

- En la tercera pasada, el tercer elemento es A[2] = 26. Luego el menor elemento encontrado a partir de A[2] es A[4] = 8, entonces cambiamos A[2] con A[4].

1	4	8	12	26	40	14
0	1	2	3	4	5	6

- En la cuarta pasada, el cuarto elemento es A[3] = 12. Luego el menor elemento encontrado a partir de A[3] no existe, entonces no hay cambio alguno.

1	4	8	12	26	40	14
0	1	2	3	4	5	6

- En la quinta pasada, el quinto elemento es A[4] = 26. Luego el menor elemento encontrado a partir de A[4] es A[6] = 14, entonces cambiamos A[4] con A[6].

1	4	8	12	14	40	26
0	1	2	3	4	5	6

- En la sexta pasada, el sexto elemento es A[5] = 40. Luego el menor elemento encontrado a partir de A[5] es A[6] = 26, entonces cambiamos A[5] con A[6].

1	4	8	12	14	26	40
0	1	2	3	4	5	6

El vector está ordenado completamente en N-1 = 6 pasadas.

Vector original	8	40	26	12	1	4	14	posi	Cambio entre A[i] y A[posi]
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]		
1ra Pasada i=0	1	40	26	12	8	4	14	4	A[0] y A[4]
2da Pasada i=1	1	4	26	12	8	40	14	5	A[1] y A[5]
3ra Pasada i=2	1	4	8	12	26	40	14	4	A[2] y A[4]
4ta Pasada i=3	1	4	8	12	26	40	14		No hay cambio
5ta Pasada i=4	1	4	8	12	14	40	26	6	A[4] y A[6]
6ta Pasada i=5	1	4	8	12	14	26	40	6	A[5] y A[6]

Se tiene un vector de N elementos para clasificar, primero se escoge el menor elemento del vector y se ubica en la posición 0. Luego quedan N-1 elementos para clasificar de los cuales se evalúa el menor y se pone en la posición 1 y así sucesivamente hasta que quede un solo elemento el cual se coloca en la última posición del arreglo. Aquí los algoritmos:

```
Procedimiento sortSimple (a, n)
        // Definir variables
        entero: i, posi
        i ← 0
        Desde (i ← 0)  hasta ( i < n) con incremento 1 Hacer
            posi ← posiminimo (a, n, i)
            Si (posi >0) entonces
                cambio (a, i, posi)
            Fin_si
        Fin_desde
Fin_procedimiento

Función posiminimo (a, n, i)
        // Definir e inicializar variables
        entero: k, posi ← 0, cambio ←0
        k ← i
        Mientras (i < n) Hacer
                Si (a[i] < a[k])  entonces
                        k ←  i
                        cambio ← 1
                Fin_si
            i ← i + 1
        Fin_mientras
        Si (cambio = 0) entonces
                k ← -1
        Fin_si
        retornar k
Fin_función

Procedimiento cambio (a, i, j)
        // Definir variable
        entero: temp
        temp ← a[i]
        a[i] ← a[j]
        a[j] ← temp
Fin_procedimiento
```

4.1.2. Método de ordenación por conteo

Dado un vector A con N elementos a clasificar en forma ascendente. Inicialmente todos los contadores son nulos. Se considera el primer elemento A_0, se compara A_0 con A_1, si A_0 es mayor que A_1 se suma una unidad al contador C_0 asociado a A_0 sino se suma una unidad al contador C_1 asociado a A_1. Luego se compara A_2 con A_0 y A_1. Según el resultado de las comparaciones se incrementan los contadores correspondientes a los elementos chequeados cuando todos los

elementos han sido tratados de la misma manera. Los contadores indicaran el rango del elemento correspondiente.

Ejemplo:

Sea A el vector a clasificar en forma ascendente.

A_0	A_1	A_2	A_3	A_4	A_5	A_6
8	40	26	12	1	4	14
0	1	2	3	4	5	6

Contadores C_0 C_1 C_2 C_3 C_4 C_5 C_6 asociados:

1. $C_0 = C_1 = C_2 = C_3 = C_4 = C_5 = C_6 = 0$.
2. Primer elemento = 8
3. Segundo elemento = 40. Luego comparamos 40 con todos los anteriores y le sumamos 1 al contador asociado del elemento mayor:

   $8 < 40$, $C_1 = C_1 + 1 = 1$ (40 es el mayor, su contador asociado es C_1)

4. Tercer elemento = 26.Luego comparamos 26 con todos los anteriores y le sumamos 1 al contador asociado del elemento mayor:

   $8 < 26$, $C_2 = C_2 + 1 = 1$ (26 es el mayor, su contador asociado es C_2)
   $40 > 26$, $C_1 = C_1 + 1 = 2$ (40 es el mayor, su contador asociado es C1).

5. Cuarto elemento = 12. Luego comparamos 12 con todos los anteriores y le sumamos 1 al contador asociado del elemento mayor:

   $8 < 12$, $C_3 = C_3 + 1 = 1$ (12 es el mayor, su contador asociado es C3).
   $40 > 12$, $C_1 = C_1 + 1 = 3$ (40 es el mayor, su contador asociado es C1).
   $26 > 12$, $C_2 = C_2 + 1 = 2$ (26 es el mayor, su contador asociado es C2).

6. Quinto elemento = 1. Luego comparamos 1 con todos los anteriores, le sumamos 1 al contador asociado del elemento mayor:

8 > 1, $C_0 = C_0 + 1 = 1$ (8 es el mayor, su contador asociado es C_0).
40 > 1, $C_1 = C_1 + 1 = 4$ (40 es el mayor, su contador asociado es C_1).
26 > 1, $C_2 = C_2 + 1 = 3$ (26 es el mayor, su contador asociado es C_2).
12 > 1, $C_3 = C_3 + 1 = 2$ (12 es el mayor, su contador asociado es C_3).

7. Sexto elemento = 4. Luego comparamos 4 con todos los anteriores, le sumamos 1 al contador asociado del elemento mayor:

8 > 4, $C_0 = C_0 + 1 = 2$ (8 es el mayor, su contador asociado es C_0).
40 > 4, $C_1 = C_1 + 1 = 5$ (40 es el mayor, su contador asociado es C_1).
26 > 4, $C_2 = C_2 + 1 = 4$ (26 es el mayor, su contador asociado es C_2).
12 > 4, $C_3 = C_3 + 1 = 3$ (12 es el mayor, su contador asociado es C_3).
1 < 4, $C_5 = C_5 + 1 = 1$ (4 es el mayor, su contador asociado es C_5).

8. Séptimo elemento = 14. Luego comparamos 14 con todos los anteriores, le sumamos 1 al contador asociado del elemento mayor :

8 < 14, $C_6 = C_6 + 1 = 1$ (11 es el mayor, su contador asociado es C_6).
40 > 14, $C_1 = C_1 + 1 = 6$ (40 es el mayor, su contador asociado es C_1).
26 > 14, $C_2 = C_2 + 1 = 5$ (26 es el mayor, su contador asociado es C_2).
12 < 14, $C_6 = C_6 + 1 = 2$ (14 es el mayor, su contador asociado es C_6).
1 < 14, $C_6 = C_6 + 1 = 3$ (14 es el mayor, su contador asociado es C_6).
4 < 14, $C_6 = C_6 + 1 = 4$ (14 es el mayor, su contador asociado es C_5).

Se obtiene

$C_0 = 2$
$C_1 = 6$
$C_2 = 5$
$C_3 = 3$
$C_4 = 0$

$C_5 = 1$
$C_6 = 4$

C_0 indica un valor de 2, entonces el lugar del elemento A_0 es la segunda posición. C_1 indica un valor de 6, entonces el lugar del elemento A_1 es la sexta posición y así sucesivamente, obtenemos el vector ordenado.

1	4	8	12	14	26	40
0	1	2	3	4	5	6

Aquí los algoritmos:

```
Procedimiento incializarVector (c, n)
        // Definir variables
        entero: i
        Desde (i ← 0) hasta (i < n) con incremento de 1 Hacer
                c[i] ← 0
        Fin_desde
Fin_procedimiento

Procedimiento conteo (a , c , n)
        // Definir variables
        entero : i, j
        i ← 0
        Mientras  (i < n-1)  Hacer
            j ← i + 1
            cuentac (a, c, i, j)
            i ← i + 1
        Fin_mientras
Fin_procedimiento

Procedimiento cuentac (a,  c ,  j)
        // Definir variable
        entero: i
        limitei ← j - 1
        i ← 0
        Mientras (i <= limitei) Hacer
            Si (a[i] > a[j]) entonces
                c[i] ← c[i] + 1
            Sino
                c[j] ← c[j] + 1
            Fin_si
            i ← i + 1
        Fin_mientras
Fin_procedimiento
```

```
Procedimiento posicion (a, c, b, n)
        // Definir variables
        entero: i, p
        i ← 0
        Mientras (i < n) Hacer
            p ← c[i]
            b[p] ← a[i]
            i ← i + 1
        Fin_mientras
Fin_procedimeinto
```

4.2. Métodos de ordenación por Comparación e Intercambio

En este algoritmo de clasificación, los elementos tomados de dos en dos, se comparan y se intercambian si no están en el orden preestablecido. Este proceso se repite hasta que se ha analizado todos sus elementos y ya no hay intercambios. Entre estos algoritmos se encuentran:

- Método de burbuja
- Método Par e Impar

4.2.1. Método de ordenación en burbuja.

Recibe este nombre por que los elementos mayores "burbujean" gradualmente (suben) hacia la parte superior del vector. "El método de intercambio directo, conocido coloquialmente con el nombre de la burbuja, es el mas utilizado entre los estudiantes principiantes de computación por su fácil comprensión y programación. Pero es preciso señalar que es probablemente el método mas ineficiente." (Cairó, 2002, p. 322).

Sea A el vector con N elementos que se desea clasificar en forma ascendente, para cada elemento se determina el mayor elemento de A y se guarda en la última posición, luego quedan N-1 elementos para clasificar y se guarda el siguiente mayor en la posición N-1 y así sucesivamente hasta que quede un solo elemento que se guarda en la posición cero.

En la primera pasada, se efectúa comparaciones a pares de elementos adyacentes y se intercambian entre sí hasta colocar el mayor elemento en la última posición.

En la segunda pasada, se realizan de nuevo comparaciones a pares sucesivos adyacentes A[i] y A[i+1], pero se trata el vector con solamente N-1 elementos puesto que el ultimo ya esta en su sitio. Al cabo de N-1 pasadas el vector ya esta ordenado.

Sea A el vector a clasificar en forma ascendente

18	40	26	15	12	9	8
0	1	2	3	4	5	6

- Primera Pasada.

$A_0 < A_1$ (18 < 40) No hay intercambio
$A_1 > A_2$ (40 > 26) Si hay intercambio
$A_2 > A_3$ (40 > 15) Si hay intercambio
$A_3 > A_4$ (40 > 12) Si hay intercambio
$A_4 > A_5$ (40 > 9) Si hay intercambio
$A_5 > A_6$ (40 > 8) Si hay intercambio

El vector queda:

18	26	15	12	9	8	40
0	1	2	3	4	5	6

- Segunda Pasada.

$A_0 < A_1$ (18 < 26) No hay intercambio
$A_1 > A_2$ (26 > 15) Si hay intercambio
$A_2 > A_3$ (26 > 12) Si hay intercambio
$A_3 > A_4$ (26 > 9) Si hay intercambio
$A_4 > A_5$ (26 > 8) Si hay intercambio

El vector queda:

18	15	12	9	8	26	40
0	1	2	3	4	5	6

- Tercera Pasada.

$A_0 > A_1$ (18 > 15) Si hay intercambio.
$A_1 > A_2$ (18 > 12) Si hay intercambio.
$A_2 > A_3$ (18 > 9) Si hay intercambio.
$A_3 > A_4$ (18 > 8) Si hay intercambio.

El vector queda:

15	12	9	8	18	26	40
0	1	2	3	4	5	6

- Cuarta Pasada.

$A_0 > A_1$ (15 > 12) Si hay intercambio.
$A_1 > A_2$ (15 > 9) Si hay intercambio.
$A_2 < A_3$ (15 > 8) Si hay intercambio.

El vector queda:

12	9	8	15	18	26	40
0	1	2	3	4	5	6

- Quinta Pasada.

$A_0 > A_1$ (12 > 9) Si hay intercambio
$A_1 > A_2$ (12 > 8) Si hay intercambio

El vector queda:

9	8	12	15	18	26	40
0	1	2	3	4	5	6

- Sexta Pasada.

$A_0 > A_1$ (9 > 8) Si hay intercambio

El vector queda:

8	9	12	15	18	26	40
0	1	2	3	4	5	6

Tabla que muestra las diferentes pasadas por el método de burbuja.

Vector original	18	40	26	15	12	9	8
Primera pasada	18	26	15	12	9	8	40
Segunda pasada	18	15	12	9	8	26	40
Tercera pasada	15	12	9	8	18	26	40
Cuarta pasada	12	9	8	15	18	26	40
Quinta pasada	9	8	12	15	18	26	40
Sexta pasada	8	9	12	15	18	26	40

Aquí los algoritmos:

Procedimiento burbuja (a, n)

```
    // Definir variable
    entero: k
    k ← n – 1
    Mientras (k >= 0)  Hacer
        desplaza (a, k)
        k ← k-1
    Fin_mientras
Fin_procedimeinto

Procedimiento desplaza (a , k)
    // Declarar e inicializar variables
    entero: i, j, temp
    i ← 0
    Mientras (i < k)  Hacer
        j ← i + 1
        Si (a[i]>a[j]) entonces
                temp ← a[i]
                a[i] ← a[j]
                a[j] ← temp
        Fin_si
        i ← i + 1
    Fin_mientras
Fin_procedimiento
```

4.2.2. Método de ordenación par e impar.

En este método cada pasada sobre el vector se realiza en dos etapas:

a) En la primera etapa cada elemento de posición impar es comparado con su sucesor y se intercambian si el sucesor fuera menor.

b) En la segunda etapa cada elemento de posición par es comparado con su sucesor y se intercambia si el sucesor fuera menor. Las pasadas sobre el vector son repetidas hasta que en una pasada no se haya realizado ningún intercambio.

Ejemplo:

Sea A el vector a clasificar en forma ascendente

8	40	26	12	1	4	14
0	1	2	3	4	5	6

- Primera pasada.

a) Primera etapa: impares

A[1] > A[2]	40 > 26	intercambio (40 y 26)
A[3] > A[4]	12 > 1	intercambio (12 y 1)
A[5] > A[6]	4> 14	no hay intercambio

El vector queda

8	26	40	1	12	4	14
0	1	2	3	4	5	6

b) Segunda etapa: pares

A[0] < A[1]	8 < 26	No hay intercambio
A[2] > A[3]	40 > 1	intercambio (40 y 1)
A[4] < A[5]	12 < 4	intercambio (12 y 4)

8	26	1	40	4	12	14
0	1	2	3	4	5	6

En esta primera pasada ha habido permutaciones entonces se realiza una nueva pasada al vector.

- Segunda pasada.

a) Primera etapa: impares

A[1] > A[2]	26 > 1	intercambio (26 y 1)
A[3] > A[4]	40 > 4	intercambio (40 y 4)
A[5] < A[6]	12 < 14	No hay intercambio

El vector queda

8	1	26	4	40	12	14
0	1	2	3	4	5	6

b) Segunda etapa: pares

A[0] > A[1]	8 >1	intercambio (8 y 1)
A[2] > A[3]	26 > 4	intercambio (26 y 4)
A[4] > A[5]	40 > 12	intercambio (40 y 12)

1	8	4	26	12	40	14
0	1	2	3	4	5	6

En esta segunda pasada ha habido permutaciones entonces se realiza una nueva pasada al vector

- Tercera pasada.

a) Primera etapa: impares

A[1] < A[2]	8 > 4	intercambiop (8 y 4)
A[3] > A[4]	26 > 12	intercambio (26 y 12)
A[5] > A[6]	40 > 14	intercambio (40 y 14)

El vector queda

1	4	8	12	26	14	40
0	1	2	3	4	5	6

b) Segunda etapa: pares

A[0] < A[1]	1 < 4	No hay intercambio
A[2] < A[3]	8 < 12	No hay intercambio
A[4] > A[5]	26 > 14	intercambio (26 y 14)

1	4	8	12	14	26	40
0	1	2	3	4	5	6

En esta tercera pasada ha habido permutaciones entonces se realiza una nueva pasada al vector

- Cuarta pasada.

a) Primera etapa: impares

A[1] < A[2]	4 < 8	No hay intercambio
A[3] > A[4]	12 < 14	No hay intercambio
A[5] < A[6]	26 < 40	No hay intercambio

El vector queda

1	4	8	12	14	26	40
0	1	2	3	4	5	6

b) Segunda etapa: pares

A[0] < A[1]	1 < 8	No hay intercambio
A[2] < A[3]	8 < 12	No hay intercambio
A[4] < A[5]	14 < 26	No hay intercambio

1	4	8	12	14	26	40
0	1	2	3	4	5	6

En esta etapa no se ha producido ningún intercambio y el vector queda completamente ordenado.

Aquí los algoritmos:

```
Procedimiento parImpar (a, n)
   // Definir variables
   entero: sw1 ← 0, sw2 ← 0
   Mientras  (sw1=0  or  sw2 = 0) Hacer
      sw1 ← par (a, 0, n)
   sw2 ← par (a, 1, n)
   Fin_mientras
Fin_procedimiento

Función par (a, i, n) : entero
   // Definir e inicializar variables
   entero: k, j, sw ← 0
   k ← i
   Mientras (k < n-1) Hacer
   j ← k + 1
   Si (a[k] > a[j]) entonces
         permuta (a, k, j)
         sw ← 1
   Fin_si
   K ←k + 2
   Fin_mientras
   Si (sw = 1) entonces
   sw ← 0
   Sino
   sw ←1
   Fin_si
   retornar sw
Fin_función

Procedimiento permuta (a, i, j)
   // Definir variable
   entero: temp
```

```
        temp ← a[i]
        a[i] ← a[j]
        a[j] ← temp
    Fin_procedimiento

    Procedimiento mostrarVector (vector, num)
        // Definir variable
        entero: i
        Desde (i ← 0) hasta (i < num) con incremento 1 Hacer
            Escribir (vector[i])
        Fin_desde
    Fin_procedimiento
```

4.3. Métodos de ordenación por inserción

Cada elemento es insertado en la posición adecuada con respecto al resto de elementos ya ordenados. Entre estos algoritmos se encuentran:

- Inserción con desplazamiento (insersión directa).
- Insersión binaria.
- Insersión por el método de Shell.
- Ordenación por particiones quick sort

4.3.1. Método de ordenación por inserción con desplazamiento

Sea A el vector con N elementos a clasificar en forma ascendente para un elemento en posición i (i >=1). Guardamos a[i] en una variable auxiliar (aux=a[i]), aux busca su lugar en las posiciones anteriores y cuando lo encuentra desplaza hacia la derecha los elementos restantes y coloca aux en la posición correspondiente.

8	40	26	12	1	4	14
0	1	2	3	4	5	6

- Primera pasada.

a) Comenzamos con el elemento 40.
b) Guardamos 40 que es el elemento activo en una variable auxiliar, aux=40.
c) Comparamos 40 con todos los anteriores:

   - 8 < 40, el elemento 8 esta ordenado y el vector sigue igual:

8	40	**26**	12	1	4	14
0	1	2	3	4	5	6

- Segunda pasada.

a) Comenzamos con el elemento 26.
b) Guardamos 26 que es el elemento activo en una variable auxiliar, aux=26.
c) Comparamos 26 con todos los anteriores:

- 8 < 26, el elemento 8 esta ordenado y el vector sigue igual.
- 40 > 26, 40 esta en la posición = 1, entonces desplazamos hacia la derecha(40) y ponemos aux en la posición 1. El vector queda como sigue:

8	26	40	**12**	1	4	14
0	1	2	3	4	5	6

- Tercera pasada.

a) Comenzamos con el elemento 12.
b) Guardamos 12 que es el elemento activo en una variable auxiliar, aux=12.
c) Comparamos 12 con todos los anteriores:

- 8 < 12, el elemento 8 esta ordenado, entonces el vector sigue igual.
- 26>1, 26 esta en la posición =1, entonces desplazamos hacia la derecha 26 y ponemos aux en la posición 1. El vector queda como sigue:

8	12	26	40	**1**	4	14
0	1	2	3	4	5	6

- Cuarta pasada.

a) Comenzamos con el elemento 1.
b) Guardamos 1 que es el elemento activo en una variable auxiliar, aux=1.
c) Comparamos 1 con todos los anteriores:

- 8 > 1, 8 esta en la posición = 0, entonces desplazamos hacia la derecha 8 y ponemos aux en la posición 0. El vector queda como sigue:

1	8	12	26	40	**4**	14
0	1	2	3	4	5	6

- Quinta pasada.

a) Comenzamos con el elemento 4.
b) Guardamos 4 que es el elemento activo en una variable auxiliar, aux=4.
c) Comparamos 4 con todos los anteriores:

- 1 < 4 el elemento 1 esta ordenado.
- 8>4, 8 esta en la posición = 1, entonces desplazamos hacia la derecha (8) y ponemos aux en la posición 1. El vector queda como sigue:

1	4	8	12	26	40	**14**
0	1	2	3	4	5	6

- Sexta pasada.

a) Comenzamos con el elemento 14
b) Guardamos 14 que es el elemento activo en una variable auxiliar, aux = 14
c) Comparamos 14 con todos los anteriores:

- 1 < 14, el elemento 1 esta ordenado
- 4 < 14, el elemento 4 esta ordenado
- 8 < 14, el elemento 8 esta ordenado
- 12 < 14, el elemento 12 esta ordenado
- 26 > 14, 26 esta en la posición = 4, entonces desplazamos hacia la derecha(26) y ponemos aux en la posición en la posición 4. El vector queda ordenado:

1	4	8	12	14	26	40
0	1	2	3	4	5	6

Especificación del TAD y los algoritmos

Aquí los algoritmos:

Procedimiento insersion (a, n)

```
    // Definir e inicializar variables
    entero: i, posi ← 0
    Desde i ← 1 Hasta i<n con incremento 1 Hacer
        posi ← lugar(a, i)
        if  (posi >0)
            desplaza (a, posi, i)
        Fin_si
    Fin_desde
Fin_procedimiento

Función lugar(a, i): entero
        // Definir e inicializar variables
        entero: k, posi ← 0, sw ← 1, aux
        aux ← a[i]
        Desde k ←0 Hasta (k <=I && sw= 1 con incremento 1 Hacer
            Si (a[k] > aux ) entonces
                        posi ← k
                        sw ← 0
                Sino
                        posi ← - 1
            Fin_si
        Fin_desde
        retornar posi
Fin_función

Procedimiento desplaza (a , posi, j)
    // Definir e inicializar variable
    entero: k, aux
    aux ← a[i]
    Desde k← i Hasta (k>=posi) con disminución 1 Hacer
                a[k] ← a[k-1]
    Fin_desde
    a[posi] ← aux
Fin_procedimiento
```

4.3.2. Método de ordenación por inserción binaria

La búsqueda binaria consiste en localizar el lugar de inserción, desplazamos elementos y luego insertarmos de la siguiente manera:

- Guardamos el contenido del índice actual i en una variable auxiliar digamos aux.
- Hacemos inf = 0, sup = i, tomamos la parte entera del valor medio:
  medio = (inf + sup) / 2.
- Comparamos aux con el contenido del valor medio. Si aux >= a[medio] el lugar de inserción se encuentra en el extremo derecho del intervalo para lo cual hacemos inf = medio + 1. Si

aux < a[medio] el lugar de inserción se encuentra en el extremo izquierdo del intervalo para lo cual hacemos sup = medio-1.

- Continua la búsqueda del lugar de inserción mientras inf sea menor que sup.
- Desplazamos elementos hacia la derecha desde la posición sup hasta el indice activo i.
- Colocar *aux* en la posición de inserción sup.

Sea A el vector a clasificar en forma ascendente:

8	40	26	12	1	4	14
0	1	2	3	4	5	6

- En la primera pasada el indice activo i = 1, aux=40 ya que es el elemento activo y 40 se compara con los anteriores (mientras inf < sup), para lo cual:

  - inf ← 0, sup ← 1, m ← (0 + 1)/2 = 0 (tomamos la parte entera)
  - Como 40 es mayor que 8 entonces inf = 1, se sale del proceso repetitivo ya que no se cumple inf < sup (1 < 1) y no podemos desplazar ya que *sup* es mayor que *i* .
  - El vector queda igual:

8	40	26	12	1	4	14
0	1	2	3	4	5	6

- En la segunda pasada el indice activo i = 2, aux= 26 ya que es el elemento activo y 26 se compara con los anteriores (mientras inf < sup), para lo cual:

  - inf ← 0, sup ← 2, m ← (0 + 2) / 2 = 1.
  - Como 26 no es mayor que 40 (no se cumple la condición), sup = 1 y repetimos otra vez el proceso repetitivo ya que inf < sup (0<1).
  - inf ← 0, sup ← 1, m ← (0 + 1) / 2 = 0 (tomamos la parte entera).
  - Como 26 es mayor que 8 entonces inf = 1 y se sale del proceso repetitivo ya que no se cumple inf < sup (1<1).
  - Desplazamos hacia la derecha a partir de la posición sup = 1 hasta el índice activo (i = 2) y luego colocamos aux, en la posición correspondiente.
  - El vector queda:

8	26	40	12	1	4	14
0	1	2	3	4	5	6

- En la tercera pasada el indice activo i = 3, aux=12 ya que es el elemento activo y 12 se compara con los anteriores (mientras inf < sup), para lo cual:

  - inf ← 0, sup ← 3, m ← (0 + 3) / 2 = 1 (tomamos la parte entera)
  - Como 12 no es mayor que 26 (no se cumple la condición) sup =1 y repetimos otra vez el proceso repetitivo ya que inf < sup (0 < 1).
  - inf ← 0, sup ← 1, m ← (0 + 1) / 2 = 0 (tomamos la parte entera).
  - Como 12 es mayor que 8 (se cumple la condición) entonces inf = 1 y se sale del proceso repetitivo ya que no se cumple inf < sup (1<1).
  - Desplazamos hacia la derecha a partir de la posición sup = 1 hasta el índice activo (i=3) y luego colocamos *aux*, en la posición correspondiente.
  - El vector queda:

8	12	26	40	1	4	14
0	1	2	3	4	5	6

- En la cuarta pasada el indice activo i = 4, aux=1 ya que es el elemento activo y 1 se compara con los anteriores (mientras inf < sup), para lo cual:

  - inf ← 0, sup ← 4, m ← (0 + 4) / 2 = 2.
  - Como 1 no es mayor que 26 (no se cumple la condición) sup = 2 y repetimos otra vez el proceso ya que inf < sup (0 < 2).
  - inf ← 0, sup ← 2, m ← (0 + 2) / 2 = 1
  - Como 1 no es mayor que aux >=12 (no se cumple la condición) sup=1 y repetimos otra vez el proceso ya que inf < sup (0 < 1).
  - inf ← 0, sup ← 1, m ← (0 + 1) / 2 = 0
  - Comoi 1 no es mayor que 8 (no se cumple la condición) sup = m =0 y se sale del proceso repetitivo ya que no se cumple inf < sup (0<0).
  - Desplazamos hacia la derecha a partir de la posición sup = 0 hasta el índice activo (i=4) y luego colocamos *aux*, en la posición correspondiente.
  - El vector queda:

1	8	12	26	40	4	14
0	1	2	3	4	5	6

- En la quinta pasada el indice activo i = 5, aux=4 ya que es el elemento activo y 1 se compara 4 con los anteriores (mientras inf < sup), para lo cual:

  - inf ← 0, sup ← 5, m ← (0 + 5) / 2 = 2 (tomamos la parte entera).
  - Como 4 no es mayor que 12 (no se cumple la condición) sup = m =2 y repetimos otra vez el proceso repetitivo ya que inf < sup (0 < 2).
  - inf ← 0, sup ← 2, m ← (0 + 2) / 2 = 1.
  - Como 4 no es mayor que 8 (no se cumple la condición) sup = m = 1 y repetimos otra vez el proceso repetitivo ya que inf < sup (0 < 1).
  - inf ← 0, sup ← 1, m ← (0 + 1) / 2 = 0
  - Como 4 es mayor que 1 (se cumple la condición) inf = m+1=1 y se sale del proceso repetitivo ya que no se cumple inf < sup (1<1).
  - Desplazamos hacia la derecha a partir de la posición sup = 1 hasta el índice activo (i = 5) y luego colocamos aux, en la posición correspondiente.
  - El vector queda:

1	4	8	12	26	40	14
0	1	2	3	4	5	6

- En la sexta pasada el indice activo i = 6, aux = 14 ya que es el elemento activo y 14 se compara con los anteriores (mientras inf < sup), para lo cual:

  - inf ← 0, sup ← 6, m ← (0 + 6) / 2 = 3
  - Como 14 es mayor que 12 (se cumple la condición) inf = m +1 = 4 y repetimos otra vez el proceso repetitivo ya que inf < sup (4 < 6).
  - inf ← 4, sup ← 6, m ← (4 + 6) / 2 = 5.
  - Como 14 no es mayor que 40 (no se cumple la condición) sup=m=5 y repetimos otra vez el proceso ya que inf < sup (4 < 5).
  - inf ← 4, sup ← 5, m ← (4 + 5) / 2 = 4 (tomamos la parte entera)

- Como 14 no es mayor que 26 (no se cumple la condición) sup = 4 y se sale del proceso repetitivo ya que no se cumple inf < sup (3<3).
- Desplazamos hacia la derecha a partir de la posición sup = 4 hasta el índice activo (i=6) y luego colocamos *aux*, en la posición correspondiente.
- El vector queda:

1	4	8	12	14	26	40
0	1	2	3	4	5	6

Tabla que muestra las diferentes pasadas por el método de inserción binaria

i	aux	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]
1	40	8	40	26	12	1	4	14
2	26	8	26	40	12	1	4	14
3	12	8	12	26	40	1	4	14
4	1	1	8	12	26	40	4	14
5	4	1	4	8	12	26	40	14
6	14	1	4	8	12	14	26	40

Aquí los algoritmos:

```
Procedimiento sortBinario(a ,  n)
        // Definir variables
        Entero: i, j, inf, sup, m
        ORDENAR : aux
        i ← 1
        Mientras (i < n) Hacer
                aux ← a[i]
                inf ← 0
                sup ← i
                Mientras (inf < sup) Hacer
                        m ← (inf + sup) / 2
                        si (aux.num ≥ a[m].num) entonces
                                inf ← m + 1
                            sino
                                sup ← m
                           Fin_si
                Fin_mientras
                j ← i
                Mientras (j >= sup +1) Hacer
                        a[j] ← a[j-1]
                        j ← j - 1
                Fin_mientras
```

```
                a[sup] ← aux
                i ← i + 1
            Fin_mientras
        Fin_procedimiento
```

4.3.3. Método de insersion de shell

Es una mejora del método de inserción por desplazamiento. El método de shell consiste de los siguientes pasos:

- Se divide el vector de n elementos en dos, consideramos un incremento o salto de n/2.
- Se compara cada elemento con el salto, si no estan ordenadas se intercambian.
- Se divide nuevamente el vector de n elementos(n/4), se compara nuevamente cada elemento con el sato(n/4) y si no estan ordenadas se intercambian.
- El algoritmo termina cuando cuando el tamaño del salto es 1 y no se ha realizado ningun intercambio.

Sea A el vector a clasificar en forma ascendente:

8	40	26	12	1	4	14
0	1	2	3	4	5	6

- En la primera pasada
  - salto= n/2 = 7/2 = 3
  - i = 0
  - comparamos a(0) con a(3) no hay cambio
  - comparamos a(1) con a(4) hay cambio

8	1	26	12	40	4	14
0	1	2	3	4	5	6

  - comparamos a(2) con a(5) hay cambio

8	1	4	12	40	26	14
0	1	2	3	4	5	6

  - comparamos a(3) con a(6) no hay cambio
  - ya no podemos seguir comparando, por que desborda los limites del array.

- En la segunda pasada

  Paso 1

  - salto= salto/2 = 3/2 = 1.
  - I = 0.
  - comparamos a(0) con a(1) hay cambio.

1	8	4	12	40	26	14
0	1	2	3	4	5	6

  - comparamos a(1) con a(2) hay cambio.

1	4	8	12	40	26	14
0	1	2	3	4	5	6

  - comparamos a(2) con a(3) no hay cambio.
  - comparamos a(3) con a(4) no hay cambio.
  - comparamos a(4) con a(5) hay cambio.

1	4	8	12	26	40	14
0	1	2	3	4	5	6

  - comparamos a(5) con a(6) cambio.

1	4	8	12	26	14	40
0	1	2	3	4	5	6

  Paso 2

  - comparamos a(0) con a(1) no hay cambio.
  - Comparamos a(1) con a(2) no hay cambio.
  - comparamos a(2) con a(3) no hay cambio.
  - comparamos a(3) con a(4) no hay cambio.
  - comparamos a(4) con a(5) hay cambio.

1	4	8	12	14	26	40
0	1	2	3	4	5	6

  - comparamos a(5) con a(6) no hay cambio.

Paso 3

- comparamos a(0) con a(1) no hay cambio.
- comparamos a(1) con a(2) no hay cambio.
- comparamos a(2) con a(3) no hay cambio.
- comparamos a(3) con a(4) no hay cambio.
- comparamos a(4) con a(5) no hay cambio.
- comparamos a(5) con a(6) no hay cambio.

como no ha habido cambio el archivo esta ordenado.

Aquí el algoritmo:

1	8	12	14	18	26	40
0	1	2	3	4	5	6

```
Procedimiento sortShell( a , n)
        Entero: i,salto,sw,salir
        ORDENAR : aux
        Salto←n/2
        Mientras(salto ≥ 1) Hacer
                repetir
                        Sw←0
                        i←0
                        repetir
                                si (a[i].num > a[i+salto].num) entonces
                                        aux ← a[i]
                                        a[i] ← a[i+salto]
                                        a[i+salto] ← aux
                                        sw←1
                                Fin_si
                                si(i+salto=n-1) entonces
                                        salir ←1
                                    sino
                                        i←i+1
                                        salir←0
                                Fin_si
                        Hasta_que(no salir)
                Hasta_que(no(sw=0))
                salto←salto/2
        Fin_mientras
Fin_procedimiento
```

"La ordenación de Shell(Shellsort, en ingles), llamada asi por su inventor, Donald Shell, fue uno de los primeros algoritmos en romper la barrera del tiempo, aunque no fue sino hasta varios años después de su descubrimiento que se demostró una cota de tiempo subcuadratica"(Allen, 1995, 127).

4.3.4. Método de ordenación por particiones quick sort

"El algoritmo de ordenacion inventado por Hoare, que se suele conocer con el nombre de quicksort u ordenacion rapida, tambien esta basado en el principio de divide y vencerás. A diferencia de ordenar por fusión, la mayor parte del trabajo no recursivo que hay que hacer se invierte en construir los subcasos, y no en combinar sus soluciones"(Brassard, 1998, 258).

- Se toma un elemento x de una posición cualquiera de un vector, este elemento se llama pivote, el pivote puede ser el primer elemento, el elemento central o ultimo elemento, generalmente se toma el elemento central.
- Se recorre el vector de izquierda a derecha buscando un elemento mayor que el pivote (sea a[i] este elemento), y de derecha a izquierda buscando un elemento menor que el pivote (sea a[j] este elemento), luego intercambiamos estos elementos a[i] y a[j].
- Se continua con el mismo proceso, mientras el indice i sea menor que j.
- En este punto todos los valores a la izquierda son menores que el pivote y todos los valores a la derecha son mayores que el pivote.
- Se ordena cada subvector independientemente.

Sea A el vector a clasificar en forma ascendente:

8	40	26	12	1	4	14
0	1	2	3	4	5	6

Calculo del pivote:

li =0
ls =n-1
m = (li+ls)/2
pivote = a[m]=12

8	40	26	12	1	4	14
0	1	2	3	4	5	6

> 12 **< 12**

- En la primera pasada
  - Empezamos la busqueda desde la izquierda (i=0), donde el primer elemento mayor que 12 es 40 (i=1).
  - Luego empezamos la busqueda desde la derecha (j=6) donde el primer elemento menor que 12 es 4 (j= 5).
  - Intercambiamos 40 y 4, obtenemos el siguiente vector.

8	40	26	12	1	40	14
0	1	2	3	4	5	6

> 12 **< 12**

Luego se incrementa el indice de i (i=1) y se disminuye el subíndice de j (j=4).

- En la segunda pasada
  - Empezamos la busqueda desde la izquierda (i=1), donde el primer elemento mayor que 12 es 26 (i=2).
  - Luego empezamos la busqueda desde la derecha (j=4), donde el primer elemento menor que 12 es 1 (j=4).
  - Intercambiamos 26 y 1, obtenemos el siguiente vector.

8	40	1	12	26	40	14
0	1	2	3	4	5	6

> 12 **< 12**

Luego se incrementa el indice de i (i=3) y se disminuye el subíndice de j (j=3). Como i< j el proceso termina

3	10	8	12	26	17	15
0	1	2	3	4	5	6

> 12 **< 12**

Luego ordenamos independientemente cada subvector, obtenemos

3	8	10	12	15	17	26
0	1	2	3	4	5	6

Vector original	8	40	26	12	1	4	14	Cambio entre A[i] y A[j]
	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	
Pri.Pasada i=0, j=6	8	4	26	12	1	40	14	A[1] y A[4]
Seg.Pasada i=1,j=4	8	4	1	12	26	40	14	A[2] y A[4]
Ter.Pasada i=3,j=3	Cuando i <j el proceso termina							

Aquí el algoritmo:

```
Procedimiento quickSort(a,  n)
{
        entero i,j,li,ls,medio,pivote
        ORDENAR : temp
        i←0
          j←n-1
        li←0
        ls←n-1
        medio←(li+ls)/2
        pivote←a[medio].num
        Mientras(i<j) Hacer
              Mientras(a[i].num<pivote) Hacer
                        i←i+1
                 Fin_mientras
              Mientras(a[j].num>pivote) Hacer
                        j←j-1
                 Fin_mientras
              si (i<j) entonces
                    temp←a[i]
                    a[i]←a[j]
                    a[j]←temp
                    i←i+1
                    j←j-1
              Fin_si
        Fin_mientras
        Desde(i←0  hasta i<medio con incremento 1 Hacer
              Desde(j←i+1 hasta j<medio+1 con increment 1
Hacer
                    si(a[i].num>a[j].num) entonces
                          temp←a[i]
                          a[i]←a[j]
                          a[j]←temp
                    Fin_si
              Fin_desde
        Fin_desde
        Desde(i←medio hasta  i<n-1 con incremento 1 Hacer
              Desde(j←i+1 hasta j<n con incremento 1 Hacer
                    si(a[i].num>a[j].num) entonces
                          temp←a[i]
```

```
                        a[i]←a[j]
                        a[j]←temp
                    Fin_si
                Fin_desde
            Fin_desde
        Fin_procedimiento
```

4.4. Búsqueda Secuencial

Consiste en revisar cada elemento hasta encontrar el dato buscado. Cuando se habla de búsqueda en vectores debe de distinguirse entre vectores desordenados y vectores ordenados.

4.4.1. En arreglos desordenados

La búsqueda secuencial en arreglos desordenados consiste básicamente, en recorrer el arreglo de izquierda a derecha hasta que encuentre el elemento buscado. Normalmente cuando una función de búsqueda concluya con éxito, interesa conocer si el elemento pertenece o no al vector.

Sea A el vector

8	40	26	12	1	4	14

Elemento a buscar 26

Comparamos cada elemento del vector con el elemento a buscar (26), como encuentra el elemento a buscar devuelve verdad, ya que 26 pertenece al vector.

Para cada elemento del vector, determina si es igual al elemento buscado, esta búsqueda es realizado por la función buscaVectorDesordenado, que devuelve verdad si elemento pertenece al vector y falso si elemento no pertenece al vector.

Especificación del TAD y los algoritmos

Especificación BUSCA
    variable
        nro: entero.
    métodos
        menu : no retorna valor.
        ingresarDatos() : no retorna valor.
        mostrarDatos() : no retorna valor.
        registrarDatos(a, n, x) : no retorna valor.
        buscaVectorDesordenado(a,n, ele) : retorna valor.
        visualizarVector(a, n) : no retorna valor.

Significado

menu muestra las opciones a escoger.
ingresarDatos ingresa datos al vector.
mostrarDatos muestra los datos del vector.
registrarDatos registra los datos de *x* en el vector *a*.
buscaVectorDesordenado tiene como precondición al vector a, su número de elementos y elemento a buscar, y como postcondición devuelve verdad si elemento pertenece al vector y falso en caso contrario.
mostrarVector muestra el vector ordenado.

Fin_especificacion

Implementación del TAD

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
class BUSCA

{
	int num;
	public:
		void menu()
		{
			cout<< "\nMENU DE OPCIONES\n";
			cout<< "---------------\n" ;
			cout<<"<1> Ingresar\n";
			cout<<"<2> Busqueda\n";
			cout<<"<3> Mostrar vector\n";
			cout<<"<4> Salir \n";
		}
		void ingresarDatos()
		{
			fflush(stdin);
			cout<<"\n ingresar numero: ";
			cin>>num;
		}
		void mostrarDatos()
		{
			cout<<num<<setw(5);
		}
		void registrarDatos(BUSCA a[50], int &n, BUSCA x)
		{
			a[n]=x;
			n++;
		}
		bool buscaVectorDesordenado (BUSCA a[50] , int n, int ele)
```

```
		{
			// Definir variables
			int i;
			bool  r;
			i = 0;
			while (a[i].num != ele  && i < n)
				i = i + 1;
			r = (a[i].num == ele);
			return r;
		}
		void mostrarVector(BUSCA a[50],int num)
		{
			// Definir variable
			int i;
			for(i = 0;i < num;i++)
				a[i].mostrarDatos();
		}
};
int main()
{
	char opcion;
	BUSCA a[50],x;
	int indice=0,ele;
	do
	{
		x.menu();
		cout<<"\ningrese opcion : ";
		opcion=cin.get();
		switch(opcion)
		{
			case '1':
				x.ingresarDatos();
				x.registrarDatos(a,indice,x);
				break;
			case '2':
				cout<<"Ingrese elemento a buscar: ";
				cin>>ele;
				x.buscaVectorDesordenado(a,indice, ele);
				if(x.buscaVectorDesordenado(a,indice, ele))
					cout<<"Elemento se encuentra en el
vector";
				else
					cout<<"Elemento no se encuentra en el
vector";
				break;
			case '3':
				x.mostrarVector(a,indice);
				break;
```

```
            }
            cin.ignore();
        }
        while( opcion !='4');
        system("PAUSE");
        return 0;
}
```

4.4.2. En arreglos ordenados

Cuando el arreglo esta ordenado se impone la condición para controlar la búsqueda, donde

A[0] <= elemento a buscar <= A[N-1]

Si es falso el elemento no esta en el rango.

Sea A el vector

1	4	8	12	18	26	40

Elemento a buscar 26

Comparamos cada elemento del vector con el elemento a buscar, como encuentra el elemento a buscar devuelve verdad, ya que 26 pertenece al vector.

Para cada elemento del vector a se determina si es igual al elemento buscado, esta búsqueda es realizado por la función buscaVectorOrdenado, que devuelve verdad si elemento pertenece al vector y falso si elemento no pertenece al vector.

Especificación del TAD y los algoritmos

Especificación BUSCA
variable
num : entero.
operaciones
menu :no retorna valor.
ingresarDatos() :no retorna valor.
mostrarDatos() : no retorna valor.
mostrarDatos() : no retorna valor.
registrarDatos(a, n, x) : no retorna valor.
buscaVectorOrdenado(a, n, ele) : retorna valor.
mostrarVector(a, n) : no retorna valor.
significado
menu muestra las opciones a escoger.
ingresarDatos ingresa datos al vector.

mostrarDatos muestra los datos del vector.
registrarDatos registra los datos de *x* en el vector *a.*
buscaVectorOrdenado tiene como precondición al vector a, su número de elementos y elemento a buscar, y como postcondición devuelve verdad si elemento pertenece al vector y falso en caso contrario.
mostrarVector muestra el vector ordenado.

Fin_especificacion

```
Funcion buscaVectorOrdenado ( a , n, ele): logico
        // Definir variables
        Entero:  i
        Logico:  r
        i ← 0
        Mientras (a[i].num < ele  ^  i < n) Hacer
                i ← i + 1
                r ← (a[i].num =ele)
                retornar r
         Fin_mientras
Fin_funcion
```

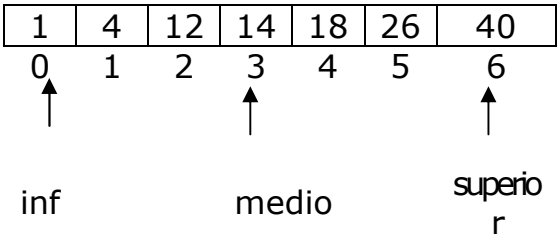
4.4.2.1. Búsqueda binaria

Consiste en dividir el intervalo de búsqueda en dos partes, comparando el elemento buscado con el elemento medio del vector. En caso de que el elemento a buscar sea mayor que el elemento medio, se continúa la búsqueda en la segunda mitad del vector. Si por el contrario el elemento a buscar es menor que el elemento medio, la búsqueda continúa en la primera mitad del vector, en ambos casos se redefinen los extremos del intervalo, disminuyendo el espacio de búsqueda, repitiendo el proceso hasta que se encuentre el valor a buscar o hasta que el elemento a buscar no se encuentre en el vector.

Pasos a efectuar para una búsqueda binaria:

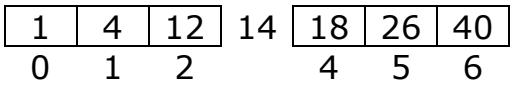
- Hacemos inf = 0 y sup = n1- 1
- Para Calcular el medio:
- medio = (inf + sup) / 2
- Si estamos en la primera mitad del vector hacemos sup = medio - 1
- Si estamos en la segunda mitad del vector hacemos inf = medio + 1

Sea A el vector:



Elemento a buscar: 26

- inf = 0 y sup = 6
- medio = (inf + sup) / 2 = (0 + 6) / 2 = 3
  a[3] =14 se divide el vector en dos partes



Como elemento buscado 26 es mayor que elemento central 14, la búsqueda continua en la segunda mitad del vector.

- inf = 4 y sup = 6
- medio = (inf + sup) / 2 = (4 + 6) / 2 = 5, a[5] = 26,
- Como 26 = a[5] la búsqueda acaba, devolviendo verdad, ya que 26 pertenece al vector.

Algoritmo

Para cada elemento del vector a se determina si es igual al elemento buscado, esta búsqueda es realizado por la función *binario*, que devuelve verdad si elemento pertenece al vector y falso si elemento no pertenece al vector.

Aquí el algoritmo:

```
Funcion buscaBinaria(a, n, ele): logico
        // Definir variables
        entero: inf, sup, m
        logico: recorre
        inf ← 0
        sup ← n-1
        si (ele ≥ a[m]) ∧ (ele ≤ a[n-1]) entonces
              recorre←verdadero
           sino
              recorre←falso
        fin_si
        Mientras (recorre) Hacer
```

```
                m ← (inf + sup) / 2
                Si (ele > a[m]) entonces
                        inf ← m + 1
                Sino
                        sup ← m-1
                Fin_si
                 si (inf ≤ sup ∧ a[m]≠ ele) entonces
                            recorre←verdadero
                        sino
                            recorre←falso
                 fin_si
            Fin_mientras
             Si a[m]= ele
                    retornar verdadero
                Sino
                    retornar falso
              Fin_si
        Fin_procedimiento
```

4.5. Ejercicios resueltos

Ejemplo 01

Escriba el programa en C++ para el método de ordenación por selección simple.

Solución:

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;
class ORDENAR
{
        int num;
        public:
                void menu()
                {
                        cout<< "\n MENU DE OPCIONES \n";
                        cout<< "----------------\n" ;
                        cout<<"<1> Ingresar vector \n";
                        cout<<"<2> Sort Simple \n";
                        cout<<"<3> Mostrar vector \n";
                        cout<<"<4> Salir \n";
                }
                void ingresarDatos( )
                {
                        fflush(stdin);
                        cout<<" Ingrese numero: ";
                        cin>>num;
```

```
}
void mostrarDatos()
{
	cout<<num<<setw(5);
}

void registrarDatos(ORDENAR a[50], int &n, ORDENAR x )
{
	a[n]=x;
	n++;
}
void sortsimple(ORDENAR a[50],int n)
{
	// Definir variables
	int i, posi;
	i=0;
	for(i = 0;i < n;i++)
	{
		posi = posiminimo (a, n, i);
		if (posi>0)
			cambio (a, i, posi);
	}
}
int posiminimo(ORDENAR a[50],int n,int i)
{
	// Definir e inicializar variables
	int k, posi = 0, cambio =0;
	k = i; // a[k] posicion del elemento actual
	while (i < n)
	{
		if (a[i].num < a[k].num)
		{
			k = i;
			cambio =1;
		}
		i = i + 1;
	}
	if (cambio =0 )
		k=-1;
	return k;
}
void cambio(ORDENAR a[50],int i,int j)
{
	// Definir variable
	ORDENAR temp;
	temp = a[i];
	a[i] = a[j];
	a[j] = temp;
```

```
            }
            void mostrarVector(ORDENAR a[50],int num)
            {
                  // Definir variable
                  int i;
                  for(i = 0;i < num;i++)
                        a[i].mostrarDatos();
            }
};

int main()
{
      char opcion;
      ORDENAR a[50],x;
      int indice=0;
      do
      {
            x.menu();
            cout<<"\n Ingrese opcion : ";
            opcion=cin.get();
            switch(opcion)
            {
                  case '1':
                        x.ingresarDatos();
                        x.registrarDatos(a,indice,x);
                        break;
                  case '2':
                        x.sortsimple(a,indice);
                        break;
                  case '3':
                        x.mostrarVector(a,indice);
                        break;
            }
            cin.ignore();
      }
      while( opcion !='4');
      system("PAUSE");
      return 0;
}
```

Ejemplo 02

Escriba el programa en C++ para el método de ordenación por conteo.

Solución

```
#include <iostream>
#include <iomanip>
```

```
#include <cstdlib>
using namespace std;

class ORDENAR
{
	int num;
	public:
		void menu()
		{
			cout<< "\n MENU DE OPCIONES \n";
			cout<< "---------------\n" ;
			cout<<"<1> Ingresar datos \n";
			cout<<"<2> Sort Conteo \n";
			cout<<"<3> Mostrar vector \n";
			cout<<"<4> Salir \n";
		}

		void ingresarDatos()
		{
			fflush(stdin);
			cout<<"Ingrese numero: ";
			cin>>num;
		}
		void mostrarDatos()
		{
			cout<<num<<setw(5);
		}

		void registrarDatos(ORDENAR a[50], int &n, ORDENAR x )
		{
			a[n]=x;
			n++;
		}
		void inicializarVector(ORDENAR c[50], int n)
		{
			// Definir variables
			int i;
			for(i = 0;i < n;i++)
			{
				c[i].num=0;
			}
		}
		void conteo (ORDENAR a[50] , ORDENAR c[50] , int n)
		{
			// Definir variables
			int i, j;
			i = 0;
			while (i< n-1)
```

```
                {
                        j = i + 1;
                        cuentac (a, c, j);
                        i = i + 1;
                }
        }
        void cuentac (ORDENAR a[50] , ORDENAR c[50] , int  j)
        {
                // Definir variable
                int i,limitei;
                i = 0;
                limitei = j -1;
                while (i <= limitei)
                {
                        if (a[i].num > a[j].num)
                                c[i].num = c[i].num + 1;
                        else
                                c[j].num = c[j].num + 1;
                        i = i + 1;
                }
        }
        void posicion (ORDENAR a[50], ORDENAR c[50], ORDENAR
b[50], int n)
        {
                // Definir variables
                int i,p;
                i = 0;
                while (i < n)
                {
                        p = c[i].num;
                        b[p] = a[i];
                        i = i + 1;
                }
        }

        void mostrarVector(ORDENAR a[50],int num)
        {
                // Definir variable
                int i;
                for(i = 0;i < num;i++)
                        a[i].mostrarDatos();
        }
};
int main()
{
        char opcion;
        ORDENAR a[50],b[50],c[50],x;
        int indice=0;
```

```
        do
        {
                x.menu();
                cout<<"\n Ingrese opcion : ";
                opcion=cin.get();
                switch(opcion)
                {
                        case '1':
                                x.ingresarDatos();
                                x.registrarDatos(a, indice, x);
                                x.inicializarVector(c, indice);
                                break;
                        case '2':
                                x.conteo(a,c,indice);
                                x. posicion (a, c,b,indice);
                                cout<<”\nSe ordeno exitosamente\n”;
                                break;
                        case '3':
                                x.mostrarVector(b,indice);
                                break;
                }
                cin.ignore();
        }
        while( opcion !='4');
        system("PAUSE");
        return 0;
}
```

Ejemplo 03

Escriba el programa en C++ para el método de ordenación por inserción con desplazamiento.

Solución

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
using namespace std;

class ORDENAR
{
        int num;
        public:
        void menu()
        {
                cout<< "\n MENU DE OPCIONES \n";
                cout<< "----------------\n" ;
                cout<<"<1> Ingresar \n";
```

```
        cout<<"<2> Sort Insersion con desplazamiento \n";
        cout<<"<3> Mostrar vector  \n";
        cout<<"<4> Salir \n";
}
void ingresarDatos()
{
        fflush(stdin);
        cout<<" Ingresar numero: ";
        cin>>num;
}
void mostrarDatos()
{
        cout<<num<<setw(5);
}
void registrarDatos(ORDENAR a[50], int &n, ORDENAR x )
{
                a[n]=x;
                n++;
}
void insersion (ORDENAR a[50], int n)
{
        // Definir e inicializar variables
        int i, j =0, posi =0;
        for (i=1;i<n;i++)
        {
                posi = lugar(a,i);
                if (posi >=0)
                        desplaza (a, posi, i);
        }
}
int lugar(ORDENAR a[50] , int i)
{
        // Definir e inicializar variables
        int k,posi = 0, sw=1;
        ORDENAR aux;
        aux = a[i];
        for (k =0;k<=i && sw==1;k++)
        {
                if (a[k].num > aux.num )
                {
                        posi = k;
                        sw=0;
                }
                else
                        posi=-1;
        }
        return posi;
}
```

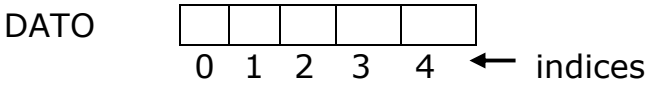
```
	void desplaza (ORDENAR a[50] , int posi, int i)
	{
		// Definir e inicializar variable
		int k;
		ORDENAR aux;
		aux = a[i];
		for (k=i;k>posi;k--)
			a[k] = a[k-1];
		a[posi] = aux;
	}
	void mostrarVector(ORDENAR a[50],int num)
	{
		// Definir variable
		int i;
		for(i = 0;i < num;i++)
			a[i].mostrarDatos();
	}
};
int main()
{
	char opcion;
	ORDENAR a[50],x;
	int indice=0;
	do
	{
		x.menu();
		cout<<"\n Ingrese opcion : ";
		opcion=cin.get();
		switch(opcion)
		{
			case '1':
				x.ingresarDatos();
				x.registrarDatos(a,indice,x);
				break;
			case '2':
				x.insersion(a,indice);
				cout<<"\n Se ordeno exitosamente\n";
				break;
			case '3':
				x.mostrarVector(a,indice);
				break;
		}
		cin.ignore();
	}
	while( opcion !='4');
	system("PAUSE");
	return 0;
}
```

4.6. Ejercicios propuestos

1. Escriba un programa en C++ para el método de ordenación por inserción binaria.
2. Escriba un programa en C++ para el método de ordenación por Shell.
3. Escriba un programa en C++ para el método de ordenación quicksort.
4. Escriba un programa en C++ para el método de búsqueda en un arreglo ordenado.
5. Escriba un programa en C++ para el método de búsqueda binaria.

Resumen

Un arreglo es un conjunto finito de elementos, todos del mismo tipo, representados por una variable, y que se registran en posiciones consecutivas de memoria. Las posiciones consecutivas de memoria se denominan celdas donde cada una tiene asociado un número llamado indice o dirección, que se enumera consecutivamente 0, 1, 2, 3,…., n-1. El tamaño o dimensión de un arreglo es el total de celdas que lo conforman. Por ejemplo, observe abajo el arreglo de nombre DATO de dimensión 5.



Los arreglos según el número de dimensiones se denominan vectores si son de una dimensión, arreglos bidimensionales o matrices si son de dos dimensiones, arreglos tridimensionales si son de tres dimensiones, etc. Al igual que con otros TAD los arreglos se basan en primitivas, funciones básicas necesarias para construir los algoritmos que posibilitan operaciones como inserción, eliminación o búsqueda de un elemento en el arreglo.

En los arreglos se han desarrollado una serie de algoritmos para clasificación u ordenación como métodos de ordenación por selección, métodos de ordenación por comparación e intercambio y métodos de ordenación por inserción e intercambio. De igual manera existen algoritmos para buscar un elemento a través de una búsqueda secuencial o por medio de una búsqueda binaria.

Como principal ventaja de un arreglo es que el acceso a cualquier elemento es muy rápido, y como desventaja más importante es que por ser una estructura estatica, no puede crecer en tiempo de ejecución.

Lectura

Recuperacion de Información

La recuperación de información es una de las más importantes aplicaciones de las computadoras. Se nos da un nombre y se nos pide un número telefónico asociado. Recibimos un número de cuenta y debemos proporcionar las transacciones que ocurren en esa cuenta. Nos dan un nombre o número de empleado y debemos encontrar los registros personales de él.

En estos ejemplos y en un sin número de otros estamos dando un fragmento de información, el cual denominaremos llave, y estamos preguntando para encontrar un registro que contiene más información asociada con la llave. Una regla que, por simplicidad, adoptamos a lo largo de este capítulo es que, dada una llave, debería haber solo un registro con esa llave. Por otro lado, es muy posible que dada una llave, no haya un registro en absoluto que la tenga.

La búsqueda de llaves para localizar registros es a veces la acción que requiere más tiempo en un programa y, por tanto, la manera en que los registros se reacomodan y la selección del método usado en la búsqueda pueden influir mucho en la ejecución del programa. El problema de la búsqueda cae en forma natural dentro de dos casos. Si hay muchos registros, cada uno quizá bastante extenso, será necesario almacenarlos en archivos en disco o cinta externos a la memoria de la computadora. Esos casos se denominan búsqueda externa e interna, respectivamente.

Robert L. Kruse (1988) *Estructura de Datos y Diseño de Programas.* México D. F., Prentice-Hall Hispanoamericana S.A. pp.87-88.

Autoevaluación

1. Dada la siguiente definición de clase, señale usted que alternativa contiene las instrucciones correctas en el main:

```
#include "iostream.h"
class A{
  int x;
 public:
  void met(A []){   }
 };
```

a)
```
int main(){
  A b[2];
  A a;
  a.met(b);
}
```

b)
```
int main(){
  A b[];
  A a;
```

c)
```
int main(){
  A b[2];
  A a;
```

d) ninguna

2. Señale usted la alternativa que tiene las instrucciones correctas para encontrar el complemento de los números pares en el arreglo de tipo A.

a)
```
class A{
  int x;
  public:
  :
void Complemento(A b[],int N){
    if(b[i].x%2!=0)
      cout<<"  "<<b[i].x;   }
}
```

b)
```
class A{
  int x;
  public:
  :
 void Complemento(A b[],int N){
   for(int i=0; i<N; i++)
     if(b[i]%2==0)
       cout<<"  "<<b[i];  }
}
```

c)
```
class A{
  int x;
  public:
  :
void Complemento(A b[],int N){
   for(int i=0; i<N; i++)
     if(b[i].x%2!=0)
       cout<<"  "<<b[i].x;
  }
}
```

d) ninguna

3. Dada el arreglo con la secuencia siguiente: 6, 10, 12, 15, 20, 23, 45, 46, 50, 55, 66, 70, 77, 80, 90, diga usted cual de las alternativas contiene el numero de divisiones necesarias para encontrar el valor 77 en una búsqueda binaria.

   a) 3 b) 4 c) 5 d) ninguna

4. Dado el arreglo con la secuencia siguiente: 20, 7, 30, 5, 12, 17, 2, diga usted cual de las alternativas contiene el número total de intercambios necesarios para poder ordenar el arreglo mediante el método burbuja.

   a) 14 b) 5 c) 20 d) ninguna

Claves: 1:a; 2:c; 3:b; 4:a;

Enlaces

http://arreglos.galeon.com/
http://sistemas.itlp.edu.mx/tutoriales/estru1/17.htm
http://www.slideshare.net/lcahuich/14-ordenacion
http://fcc98.tripod.com/tutores/ed1/ed1.html#CONTE

Bibliografía

Tenenbaum A. M., Langsam Y., Augenstein, M.A., (1993) *Estructura de Datos en C*, 2ª. Ed., México D.F., Prentice Hall Hispanoamericana S.A. Cap. 2: Recursividad ala estrcutura de datos, pp. 117 -173

Allen Weiss, M., (1995) *Estructura de Datos y Algoritmos*, México D.F., Addison-Wesley Iberoamericana. Cap. 8: Algoritmos de ordenación, pp. 135-168.

Brassard, G., Bratley, P., (1998) *Fundamentos de Algoritmia*, 2ª. Ed. México D.F., Prentice Hall. Cap. 4: Analisis de Algortimos, pp. 111-166. Cap. 4: Analisis de Algortimos, pp. 111-166.

Tenenbaum A. M., Langsam Y., Augenstein, M. A., (1986) *Estructura de Datos en Pascal*, México D.F., Prentice Hall Hispanoamericana S.A. Cap. 2: Estructuras de datos estáticas, pp. 59-112.

Cairó O., Guardati M.C. S., (2002) *Estructura de Datos*, 2ª. Ed. México D.F., Mc Graw-Hill / Interamericana Editores S.A. Cap. 2: Arreglos multidimensionales representados en arreglos unidimensionale, pp. 71-125.