



Introducción

Uno de los procedimientos más comunes y útiles en el procesamiento de datos, es la clasificación u ordenación de los mismos. Se considera ordenar al proceso de reorganizar un conjunto dado de objetos en una secuencia determinada. Cuando se analiza un método de ordenación, hay que determinar cuántas comparaciones e intercambios se realizan para el caso más favorable, para el caso medio y para el caso más desfavorable.

La colocación en orden de una lista de valores se llama **Ordenación**. Por ejemplo, se podría disponer una lista de valores numéricos en orden ascendente o descendente, o bien una lista de nombres en orden alfabético. La localización de un elemento de una lista se llama **búsqueda**.

Ordenación

Los métodos de ordenación se suelen dividir en dos grandes grupos:

- **directos** *intercambio, selección, inserción*
- **indirectos (avanzados)** *Shell, ordenación rápida, ordenación por mezcla, Radixsort*

En el caso de listas pequeñas, los métodos directos se muestran eficientes, sobre todo porque los algoritmos son sencillos; su uso es muy frecuente. Sin embargo, en listas grandes estos métodos se muestran ineficaces y es preciso recurrir a los métodos avanzados.

En esta práctica sólo analizaremos los métodos directos.

Intercambio

El método de intercambio se basa en comparar los elementos del arreglo e intercambiarlos si su posición actual o inicial es contraria inversa a la deseada. Pertenece a este método el de la burbuja clasificado como intercambio directo. Aunque no es muy eficiente para ordenar listas grandes, es fácil de entender y muy adecuado para ordenar una pequeña lista de unos 100 elementos o menos.

1. Ordenación por intercambio directo (Método de la Burbuja).

Este método se basa en el siguiente algoritmo: se van recorriendo una a una todas las posiciones del vector, desde la primera hasta la penúltima. Estando en cada una de estas posiciones, se recorren, a su vez, todas las posiciones siguientes y se compara su valor con el de la posición actual. Si se encuentra un valor menor se intercambia con el de esta posición.

Para implementar este algoritmo son necesarios dos bucles: el primero, bucle *i*, recorre el vector desde la posición *i=0* hasta *i=SIZE-1*. El segundo bucle, bucle *j*, recorre el vector desde la posición *j=i+1* hasta el final. Para que quede más claro, vamos a ver con un ejemplo como funciona este algoritmo. Supongamos que queremos ordenar los siguientes cinco números: 7,3,5,1,4. Estos números se almacenarán en un vector de la siguiente manera:

<u>vector[0]</u>	<u>vector[1]</u>	<u>vector[2]</u>	<u>vector[3]</u>	<u>vector[4]</u>
7	3	5	1	4

Vamos a recorrer las posiciones del vector desde *i=0* hasta *i=3*.

i = 0 {7 3 5 1 4}

Recorremos el vector desde *j=1* hasta *j=4* y comparamos **vector [0]=7** con **vector [j]**. Si **vector [j]<vector [0]** intercambiamos los valores de posición. Vamos a ver cómo quedaría el vector inicial una vez que termina cada **bucle j**.

Practica10. Algoritmos de ordenación y búsqueda



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

$j = 1$ {3 7 5 1 4} Se intercambia 3 con 7

$j = 2$ {3 7 5 1 4} No se intercambia 3 con 5

$j = 3$ {1 7 5 3 4} Se intercambia 1 con 3

$j = 4$ {1 7 5 3 4} No se intercambia 1 con 4

$i = 1$ {1 7 5 3 4}

Recorremos el vector desde $j=2$ hasta $j=4$ y comparamos **vector** [1]=7 con **vector** [j].

$j = 2$ {1 5 7 3 4} Se intercambia 5 con 7

$j = 3$ {1 3 7 5 4} Se intercambia 3 con 5

$j = 4$ {1 3 7 5 4} No se intercambia 3 con 4

$i = 2$ {1 3 7 5 4}

$j = 3$ {1 3 5 7 4} Se intercambia 5 con 7

$j = 4$ {1 3 4 7 5} Se intercambia 4 con 5

$i = 3$ {1 3 4 7 5}

$j = 4$ {1 3 4 5 7} Se intercambia 5 con 7 ¡Números ordenados!

Ya se ve que no es necesario que el **bucle** i llegue hasta el valor 4.

2. Selección.

Los métodos de ordenación por selección se basan en dos principios básicos: Seleccionar el elemento más pequeño (o más grande) del arreglo. Colocarlo en la posición más baja (o más alta) del arreglo. A diferencia del método de la burbuja, en este método el elemento más pequeño (o más grande) es el que se coloca en la posición final que le corresponde.

Algoritmo

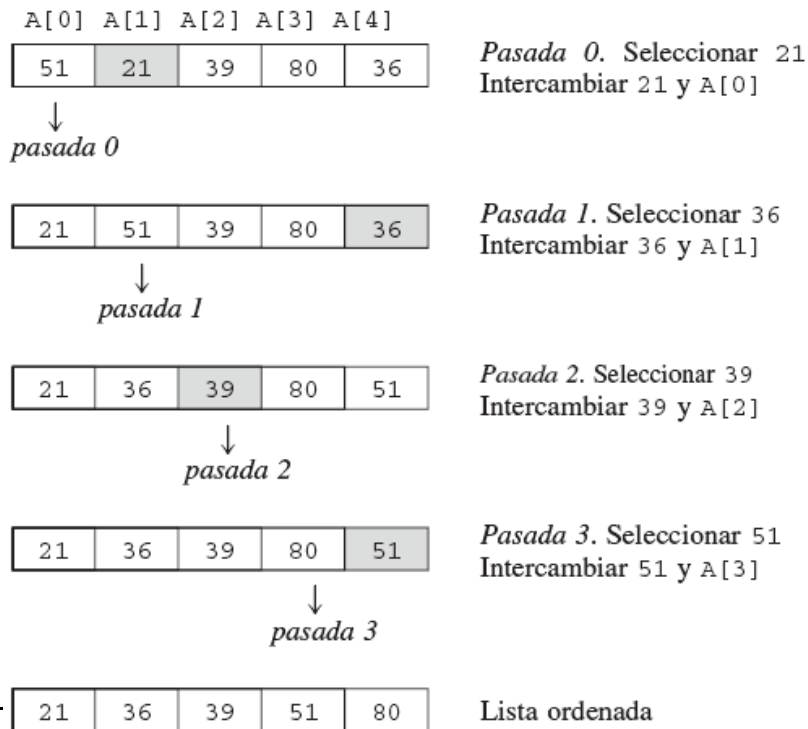
Considérese el algoritmo para ordenar un arreglo A de enteros en orden ascendente, es decir, del número más pequeño al mayor. Es decir, si el arreglo A tiene n elementos, se trata de ordenar los valores del arreglo de modo que el dato contenido en A[0] sea el valor más pequeño, el valor almacenado en A[1] el siguiente más pequeño, y así hasta A[n-1], que ha de contener el elemento de mayor valor. El algoritmo se apoya en pasadas que intercambian el elemento más pequeño sucesivamente con el primer elemento de la lista, A[0] en la primera pasada. En síntesis, se busca el elemento más pequeño de la lista y se intercambia con A[0], primer elemento de la lista. A[0] A[1] A[2]... A[n-1]

Después de terminar esta primera pasada, el frente de la lista está ordenado y el resto de la lista A[1], A[2]...A[n-1] permanece desordenado. La siguiente pasada busca en esta lista desordenada y *selecciona* el elemento más pequeño, que se almacena entonces en la posición A[1]. De este modo los elementos A[0] y A[1] están ordenados y la sublista A[2], A[3]...A[n-1] desordenada; entonces, se selecciona el elemento más pequeño y se intercambia con A[2]. El proceso continúa $n - 1$ pasadas y en ese momento la lista desordenada se reduce a un elemento (el mayor de la lista) y el array completo ha quedado ordenado.

Un ejemplo práctico ayudará a la comprensión del algoritmo. Consideremos un arreglo A con 5 valores enteros 51, 21, 39, 80, 36:



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA



Algoritmo.

1. Seleccionar el elemento más pequeño de la lista A; intercambiarlo con el primer elemento A[0]. Ahora la entrada más pequeña está en la primera posición del vector.
2. Considerar las posiciones de la lista A[1], A[2], A[3], A[4], seleccionar el elemento más pequeño e intercambiarlo con A[1]. Ahora las dos primeras entradas de A están en orden.
3. Continuar este proceso encontrando o seleccionando el elemento más pequeño de los restantes elementos de la lista, intercambiándolos adecuadamente.

3. Inserción

El fundamento de este método consiste en insertar los elementos no ordenados del arreglo en subarreglos del mismo que ya estén ordenados. Dependiendo del método elegido para encontrar la posición de inserción tendremos distintas versiones del método de inserción.

El método de ordenación por inserción es similar al proceso típico de ordenar tarjetas de nombres (cartas de una baraja) por orden alfabético, que consiste en insertar un nombre en su posición correcta dentro de una lista o archivo que ya está ordenado. Así el proceso en el caso de la lista de enteros A = 50, 20, 40, 80, 30.



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

- 50 • Comienzo con 50
- Procesar 20 20 50 • Se inserta 20 en la posición 0
• 50 se mueve a posición 1
- Procesar 40 20 40 50 • Se inserta 40 en la posición 1
• Se mueve 50 a posición 2
- Procesar 80 20 40 50 80 • El elemento 80 está bien ordenado
- Procesar 30 20 30 40 50 80 • Se inserta 30 en posición 1
• Se desplaza a la derecha la sublista derecha

Algoritmo de ordenación por inserción

El algoritmo correspondiente a la ordenación por inserción contempla los siguientes pasos:

1. El primer elemento A[0] se considera ordenado; es decir, la lista inicial consta de un elemento.
2. Se inserta A[1] en la posición correcta, delante o detrás de A[0], dependiendo de que sea menor o mayor.
3. Por cada bucle o iteración i (desde i=1 hasta n-1) se explora la sublista A[i-1] . .A[0] buscando la posición correcta de inserción; a la vez se mueve hacia abajo (a la derecha en la sublista) una posición todos los elementos mayores que el elemento a insertar A[i], para dejar vacía esa posición.
4. Insertar el elemento a la posición correcta.

Búsqueda

La búsqueda es una operación que tiene por objeto la localización de un elemento dentro de la estructura de datos. A menudo un programador estará trabajando con grandes cantidades de datos almacenados en arreglos y pudiera resultar necesario determinar si un arreglo contiene un valor que coincide con algún valor clave o buscado.

Siendo el arreglo de una dimensión o lista una estructura de acceso directo y a su vez de acceso secuencial, encontramos dos técnicas que utilizan estos dos métodos de acceso, para encontrar elementos dentro de un array: búsqueda lineal y búsqueda binaria.

4. Búsqueda Secuencial

La búsqueda secuencial es la técnica más simple para buscar un elemento en un arreglo. Consiste en recorrer el arreglo elemento a elemento e ir comparando con el valor buscado (clave). Se empieza con la primera casilla del arreglo y se observa una casilla tras otra hasta que se encuentra el elemento buscado o se han visto todas las casillas. El resultado de la búsqueda es un solo valor, y será la posición del elemento buscado o cero. Dado que el arreglo no está en ningún orden en particular, existe la misma probabilidad de que el valor que se encuentra ya sea en el primer elemento, como en el último. Por lo tanto, en promedio, el programa tendrá que comparar el valor buscado con la mitad de los elementos del arreglo. El método de búsqueda lineal funciona bien con arreglos pequeños o para arreglos no ordenados. Si el arreglo está ordenado, se puede utilizar la técnica de alta velocidad de búsqueda binaria, donde se reduce sucesivamente la operación eliminando repetidas veces la mitad de la lista restante.



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

5. Búsqueda Binaria

La búsqueda binaria es el método más eficiente para encontrar elementos en un arreglo ordenado. El proceso comienza comparando el elemento central del arreglo con el valor buscado. Si son iguales finaliza la búsqueda. Si no ocurre así, el elemento buscado será mayor o menor en sentido estricto que el central del arreglo. Si el elemento buscado es mayor se procede a hacer búsqueda binaria en el subarreglo superior, si el elemento buscado es menor que el contenido de la casilla central, se debe cambiar el segmento a considerar al segmento que está a la izquierda de tal sitio central.

Se desea buscar el elemento 225 y ver si se encuentra en el conjunto de datos siguiente:

Table with 8 columns: a[0] to a[7] and values: 13, 44, 75, 100, 120, 275, 325, 510

El punto central de la lista es el elemento a[3]=(100). El valor que se busca es 225, mayor que 100; por consiguiente, la búsqueda continúa en la mitad superior del conjunto de datos de la lista, es decir, en la sublista:

Table with 4 columns: a[4] to a[7] and values: 120, 275, 325, 510

Ahora el elemento mitad de esta sublista a[5]=(275). El valor buscado, 225, es menor que 275 y, por consiguiente, la búsqueda continúa en la mitad inferior del conjunto de datos de la lista actual; es decir, en la sublista de un único elemento:

Table with 1 column: a[4] and value: 120

El elemento mitad de esta sublista es el propio elemento a[4]=(120). Al ser 225 mayor que 120, la búsqueda debe continuar en una sublista vacía. Se concluye indicando que no se ha encontrado la clave en la lista.

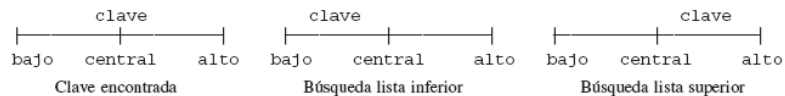
Algoritmo

Suponiendo que la lista está almacenada como un array, los índices de la lista son: bajo = 0 y alto = n-1 y n es el número de elementos del array, los pasos a seguir:

- 1. Calcular el índice del punto central del array

central = (bajo + alto) / 2 (división entera)

- 2. Comparar el valor de este elemento central con la clave:



- Si a[central] < clave, la nueva sublista de búsqueda tiene por valores extremos de su rango bajo = central+1..alto.
• Si clave < a[central], la nueva sublista de búsqueda tiene por valores extremos de su rango bajo..central-1.



El algoritmo se termina bien porque se ha encontrado la clave o porque el valor de bajo excede a alto y el algoritmo devuelve el indicador de fallo de -1 (búsqueda no encontrada).



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

Ejemplo

Sea el array de enteros A (-8, 4, 5, 9, 12, 18, 25, 40, 60), buscar la clave, clave = 40.

1.	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	
	-8	4	5	9	12	18	25	40	60	bajo = 0 alto = 8
					↑ central					

$$central = \frac{bajo + alto}{2} = \frac{0 + 8}{2} = 4$$

clave (40) > a[4] (12)

2. Buscar en sublista derecha

18	25	40	60	bajo = 5 alto = 8
		↑		

$$central = \frac{bajo + alto}{2} = \frac{5 + 8}{2} = 6 \quad (\text{división entera})$$

clave (40) > a[6] (25)

3. Buscar en sublista derecha

40	60	bajo = 7 alto = 8
	↑	

$$central = \frac{bajo + alto}{2} = \frac{7 + 8}{2} = 7$$

clave (40) = a[7] (40) (búsqueda con éxito)

4. El algoritmo ha requerido 3 comparaciones frente a 8 comparaciones ($n - 1, 9 - 1 = 8$) que se hubieran realizado con la búsqueda secuencial.



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

Ejercicios:

1. Adiciona a tu librería Arreglos los siguientes métodos:
 - a) Método **genera** que asigna valores al vector (envías como parámetro el número de elementos)
 - b) Método **menor** que nos devuelve el elemento más pequeño y su posición. Deberá recibir como parámetro el vector.
 - c) Método **mayor** que nos devuelve el mayor y su posición. Deberá recibir como parámetro el vector.
 - d) Método **escribe** que escriba los elementos del vector. Deberá recibir como parámetro el vector.

2. **Esta práctica puedes hacerla en equipo de máximo 3 personas. El equipo que termine la práctica completa tiene un punto sobre segundo Parcial.** Crea una nueva solución llamada practica10_algoritmos. En la clase Program deberás tener un menú principal que muestre las opciones:
 - 1) Métodos de ordenación
 - 2) Métodos de búsqueda
 - 3) Salir

También contarás con dos submenús que muestren lo siguiente:

```
submenu_ordenacion( )  
1) Método de la Burbuja  
2) Método de selección  
3) Método de inserción
```

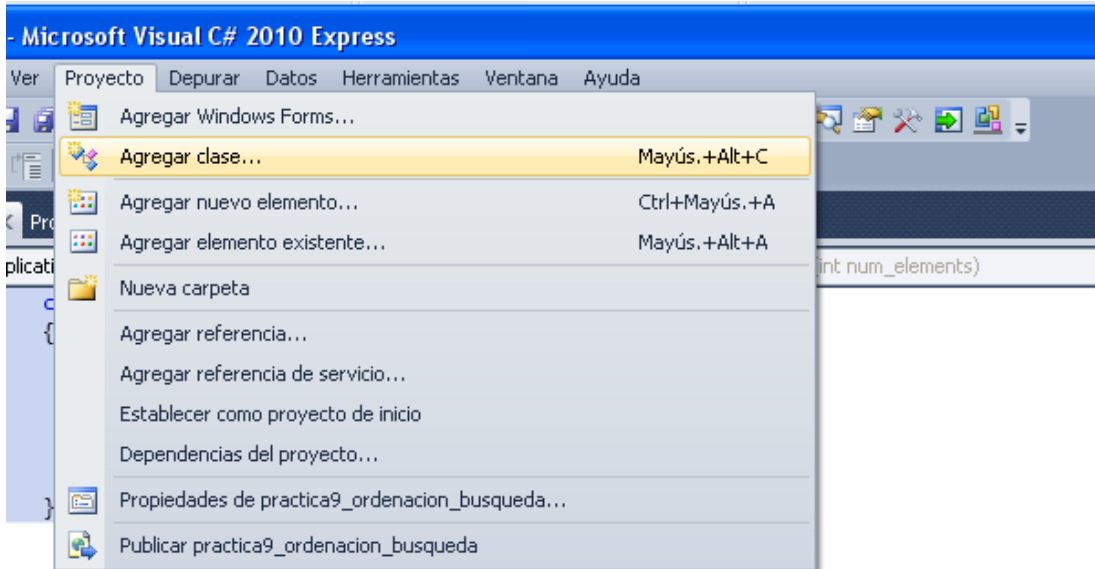
```
submenu_busqueda( )  
1) Búsqueda secuencial  
2) Búsqueda binaria
```

3. Posteriormente Crea una nueva clase llamado AlgOrdBusq (Esta clase contendrá los métodos para algoritmos de ordenación y búsqueda). Dentro de la clase deberás tener dos subclases:
 - Ordenacion
 - Busqueda

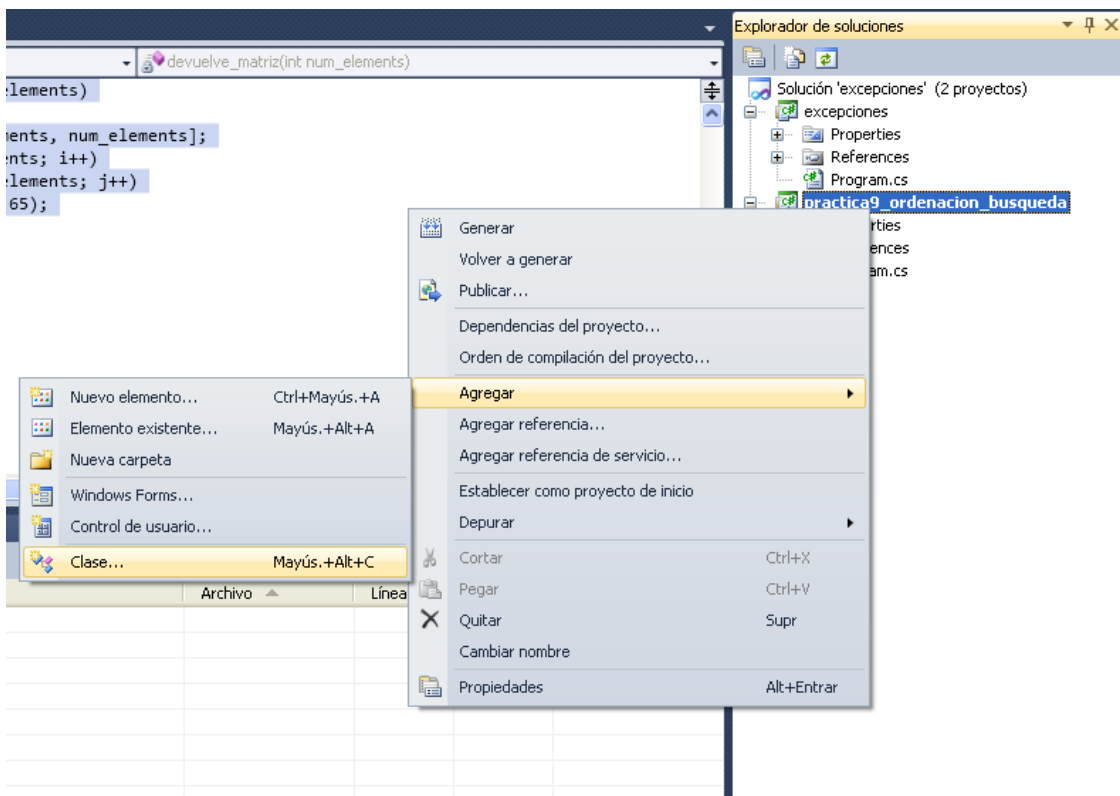
Recuerda que para agregar nueva clase ve al menú principal Proyecto> Agregar clase.



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA



O bien desde tu explorador de soluciones selecciona el proyecto con el botón derecho >Agregar>Clase

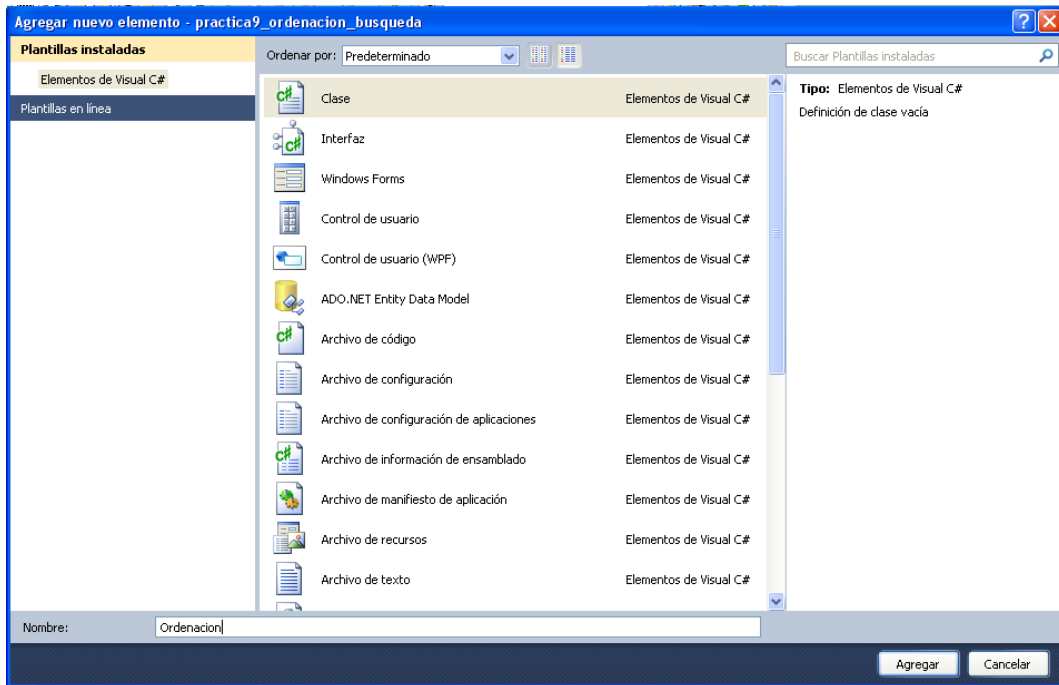


Posteriormente elige clase y cambia el nombre

Practica10. Algoritmos de ordenación y búsqueda



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA



4. La clase Ordenacion deberá contener implementados los tres métodos de ordenación del menú burbuja()
seleccion()
insercion()
5. La clase Busqueda deberá contener implementados los dos métodos de búsqueda:
secuencial()
binaria() *

*El método binaria deberá recibir el vector ordenado por cualquiera de los métodos de ordenación antes de ser ejecutado.

NOTAS:

- a) Deberás utilizar tu librería de validaciones y arreglos.
- b) Recuerde que para llamar a un método de otra clase, el método deberá estar público y se hará la llamada de la siguiente manera:

```
nombre_clase Objeto = new nombre_clase( );  
Objeto.metodo( );
```

```
Ejemplo:  
Busqueda objetoB = new Busqueda();  
objetoB.Secuencial(vector,variable_busqueda);
```

Practica10. Algoritmos de ordenación y busqueda



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

- c) Sus aplicaciones deben ser consideradas para un arreglo de N elementos.
- d) El código debe contener nombre de los autores, fecha y cual es el propósito de la aplicación.
- e) Debe contener los comentarios necesarios para claridad del programa.
- f) Debe contener validación de datos y manejo de excepciones.
- g) Debe presentar al usuario una bienvenida con introducción de lo que realiza su aplicación.



ANEXO. EJEMPLO DE MÉTODOS CON ARREGLOS

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ejemplo_metodos_arreglos
{
    class Program
    {
        static void Main(string[] args)
        {
            int i;
            int[] miArregloEnteroFijo = new int[10];
            int[] unvector;
            char[,] unamatriz;
            for (i = 0; i <= 9; i++)
                miArregloEnteroFijo[i] = i;

            Program miObjeto = new Program();
            Console.WriteLine("*****LLAMADA A INFORMA*****");
            miObjeto.informeArreglo(miArregloEnteroFijo);
            Console.WriteLine("*****");
            miObjeto.muestraElementos(miArregloEnteroFijo);
            Console.WriteLine("*****");
            miObjeto.muestraElementos('a', 'b', 'c', 'g', 'z', 'Q', 'R');
            Console.WriteLine("*****DEVULVE UN VECTOR*****");
            unvector=miObjeto.devuelve_vector();
            miObjeto.muestraElementos(unvector);
            Console.WriteLine("*****DEVULVE UNA MATRIZ*****");
            unamatriz = miObjeto.devuelve_matriz(4);
            miObjeto.muestraElementos(unamatriz);
        }

        void informeArreglo(Array arreglo)
        {
            byte dimension;
            if (arreglo is Array)
            {
                Console.WriteLine("Informe del arreglo creado:");
                Console.WriteLine("Tipo de arreglo : {0}", arreglo.GetType());
                Console.WriteLine("Dimensiones del arreglo: {0}", arreglo.Rank);
                for (dimension = 0; dimension < arreglo.Rank; dimension++)
                {
                    Console.WriteLine("La dimension {0}, tiene {1} elementos",
dimension, arreglo.GetLength(dimension));
                    Console.WriteLine("El indice inferior es {0} y el superior {1}",
arreglo.GetLowerBound(dimension), arreglo.GetUpperBound(dimension));
                }
            }
        }

        void muestraElementos(int[] arreglo)
        {
```



TEMA2. METODOLOGÍA DE LA PROGRAMACIÓN ESTRUCTURADA

```
int i = 0;
Console.WriteLine("Los elementos de mi arreglo son:");
foreach (int miValor in arreglo)
{
    Console.WriteLine("Elemento del arreglo {0} es {1}", i, miValor);
    i++;
}

void muestraElementos(params char[] arreglo)
{
    int i = 0;
    Console.WriteLine("Los elementos de mi arreglo son:");
    foreach (char miValor in arreglo)
    {
        Console.WriteLine("Elemento del arreglo {0} es {1}", i, miValor);
        i++;
    }
}

void muestraElementos(char[,] arreglo)
{
    int i = 0;
    Console.WriteLine("Los elementos de mi matriz son:");
    foreach (char miValor in arreglo)
    {
        Console.WriteLine("Elemento del arreglo {0} es {1}", i, miValor);
        i++;
    }
}

int[] devuelve_vector()
{
    int[] A = new int[5];
    for (int i = 0; i < 5; i++)
        A[i] = i*5;
    return A;
}

char[,] devuelve_matriz(int num_elements)
{
    char[,] B = new char[num_elements, num_elements];
    for (int i = 0; i < num_elements; i++)
        for (int j = 0; j < num_elements; j++)
            B[i, j] = (char)(i + 65);
    return B;
}
}
```