

**ESCUELA SUPERIOR DE CÓMPUTO**

**PRACTICA 1:** “Recursividad”

**ASIGNATURA:** Estructuras de Datos

**Profesor:** Cruz García Daniel

**Integrantes:**

* De los Santos Montiel Emmanuel
* Del Carmen Hernández Rodrigo Alfredo
* Guzmán Suárez Jesús Bryan

**Grupo:** 1CM3

**Fecha de Entrega:** 19-Diciembre-2020

**INTRODUCCIÓN**

**¿QUÉ ES RECURSIVIDAD?**

La recursividad es un concepto fundamental en matemáticas y en computación. Es un recurso muy poderoso que permite expresar soluciones simples y naturales a ciertos tipos de problemas. Es importante considerar que no todos los problemas son naturalmente recursivos. Un objeto recursivo es aquel que aparece en la definición de sí mismo, así como el que se llama a sí mismo.

**EJEMPLOS DE USOS DE LA RECURSIVIDAD**



**¿Por qué escribir Programas Recursivos?**

* Son más cercanos a la descripción matemática.
* Generalmente más fáciles de analizar.
* Se adaptan mejor a las estructuras de datos recursivas.
* Los algoritmos recursivos ofrecen soluciones estructuradas, modulares y elegantemente simples.

**Donde o cuando usar la Recursividad**

* Para simplificar el código
* Cuando la estructura de datos es recursiva ejemplo: Árboles.

**Donde o cuando No usar la Recursividad**

* Cuando los métodos usen arreglos largos.
* Cuando el método cambia de manera impredecible de campos.
* Cuando las iteraciones sean la mejor opción.

**DESARROLLO DE LA PRÁCTICA**

**ALGORITMOS:**

**EJ. 1:**

**EJ. 2:**

**EJ. 3: “Pseudocodigo Búsqueda Binaria de un Conjunto de n Datos”**

Entradas:

* Arreglo: números []= {1, 2, 4, 8, 15, 16, 20, 50,100,450}
* longitudDelArreglo = sizeof(numeros) / sizeof(numeros[0]);
* Variables: busqueda,x;

Salidas:

* resultadoBusquedaRecursiva = busquedaBinariaRecursiva(numeros, busqueda, 0, longitudDelArreglo - 1);

Inicio.

1. Teniendo el arreglo de números, seleccionar 1 número a buscar.
2. Introducido el número se buscará por medio de recursividad que posición ocupa dentro del arreglo.
3. Finalmente cuando lo encuentra se imprimirá el valor en donde esta almacenado dicho número.

Fin.

**EJ. 4: “Pseudocodigo Ordenamiento por la técnica Merge\_Sort”**

Entradas

* Variable “i”
* Arreglo: A [] = {2, -40, 5, -7, 1, 2, 3, 6,10,100,80};
* r = sizeof(A)/sizeof (A [0]) - 1, p = 0;

Salidas

* merge(array, p, q, r);

Inicio

1. Ponemos los números que queremos dentro del arreglo, o como viene en el programa, ya vienen escritos.
2. Se mostrarán los números en pantalla, pero antes se mandarán a una función en donde los separaremos en subarreglos.
3. Se realizara la técnica de Merge\_Sort, la cual ordenara los datos ingresados.
4. Se imprimirán los números ya en orden ascendente.

Fin

**“Pseudocodigo Método de la Burbuja”**

Entradas

* Variable “x”
* Arreglo: arreglo [] = {30, 28, 11, 96, -5, 21, 18, 12, 22, 30, 97, -1, -40, -500}
* longitud = sizeof arreglo / sizeof arreglo [0];

Salidas

* burbuja (int arreglo [], int longitud);

Inicio

1. Se ingresarán los valores que desea ordenar, o como viene en el programa ya pueden venir escritos.
2. Se imprimirán en pantalla aún sin ordenar.
3. Se mandará a llamar la función Burbuja, la cual hará que se vayan leyendo los valores y los ordenara en orden ascendente.
4. Finalmente se mostrarán en pantalla con el orden correcto.

Fin

**EJ. 5:**

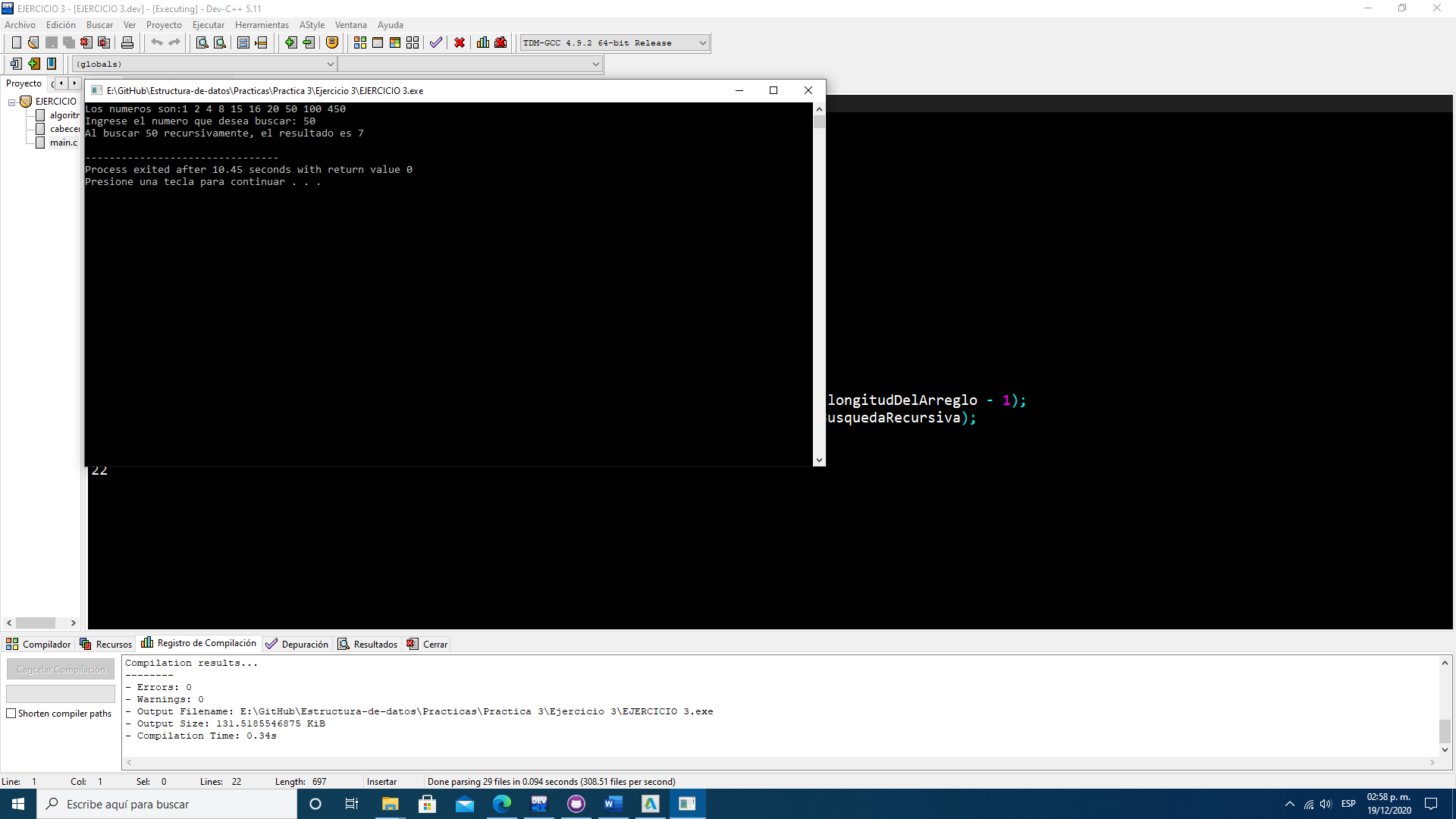
**EJ. 6:**

**PRUEBAS Y SALIDAS.**

**EJ. 1:**

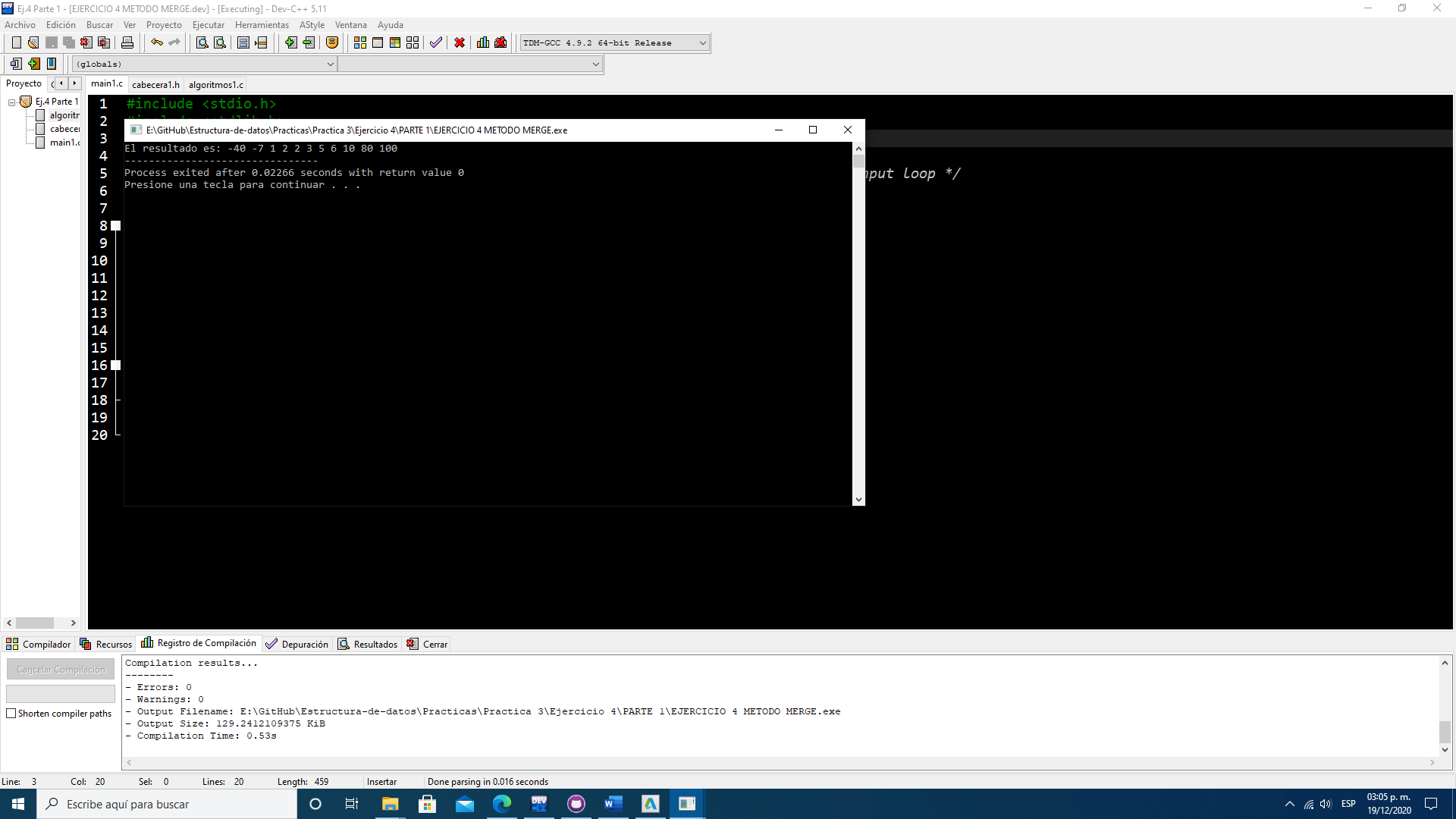
**EJ. 2:**

**EJ. 3:**

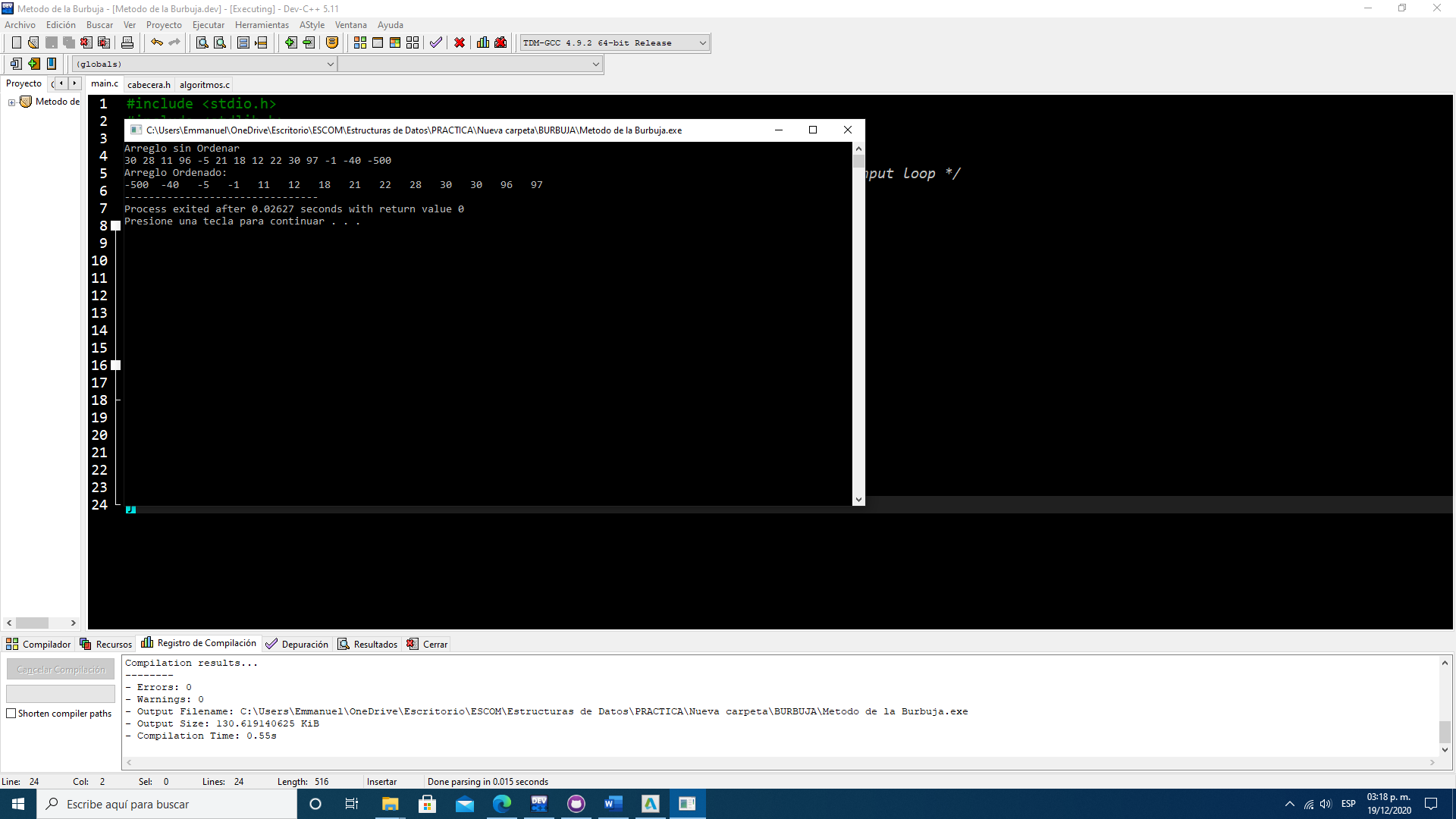


**EJ. 4:**

**Primera Parte “Merge\_Sort”**



**Segunda Parte “Método de la Burbuja”**



**EJ. 5:**

**EJ. 6:**

**CONCLUSIONES**

**Emmanuel:** En conclusión, puedo decir que se cumplieron los objetivos de la práctica, ya que, al realizarla pudimos poner en práctica los conocimientos que adquirimos durante las clases de Recursividad, además de que en mi caso, pude investigar un poco más sobre algunos métodos de ordenamiento que no tenía muy claros, para así poder ponerlos en práctica y poder resolver los problemas de la práctica.

**BIBLIOGRAFIA**

* Recursividad. (2011, abril). UV. <https://www.uv.mx/personal/ocastillo/files/2011/04/Recursividad.pdf>
* Cruz, D. (2020, diciembre). Recursividad. Ea Kdemy. <https://comunidad.eakdemy.com/login/index.php?id=11>

**ANEXOS**

**EJ. 1:**

**EJ. 2:**

**EJ. 3:**

**Main**

#include <stdio.h>

#include <math.h> // Para floor

#include "cabecera.h"

int main(){

int numeros[] = {1, 2, 4, 8, 15, 16, 20, 50,100,450};

int longitudDelArreglo = sizeof(numeros) / sizeof(numeros[0]);

int busqueda,x;

printf("Los numeros son:");

for( x = 0; x < longitudDelArreglo; x++)

{

printf("%d ", numeros[x]);

}

printf("\n");

printf("Ingrese el numero que desea buscar: ");

scanf("%d", & busqueda);

int resultadoBusquedaRecursiva = busquedaBinariaRecursiva(numeros, busqueda, 0, longitudDelArreglo - 1);

printf("Al buscar %d recursivamente, el resultado es %d\n", busqueda, resultadoBusquedaRecursiva);

return 0;

}

**Cabecera**

#include <stdio.h>

#include <math.h> // Para floor

int busquedaBinariaRecursiva(int arreglo[], int busqueda, int izquierda, int derecha);

**Algoritmos**

#include <stdio.h>

#include <math.h> // Para floor

#include "cabecera.h"

int busquedaBinariaRecursiva(int arreglo[], int busqueda, int izquierda, int derecha)

{

if (izquierda > derecha) return -1;

int indiceDeLaMitad = floor((izquierda + derecha) / 2);

int valorQueEstaEnElMedio = arreglo[indiceDeLaMitad];

if (busqueda == valorQueEstaEnElMedio){

return indiceDeLaMitad;

}

if (busqueda < valorQueEstaEnElMedio){

// Entonces está hacia la izquierda

derecha = indiceDeLaMitad - 1;

}else{

// Está hacia la derecha

izquierda = indiceDeLaMitad + 1;

}

return busquedaBinariaRecursiva(arreglo, busqueda, izquierda, derecha);

}

**EJ. 4:(Primera Parte “Merge\_Sort”)**

**Main**

#include <stdio.h>

#include <stdlib.h>

#include "cabecera1.h"

int main()

{

int i;

int A[] = {2, -40, 5, -7, 1, 2, 3, 6,10,100,80};

int r = sizeof(A)/sizeof(A[0]) - 1, p = 0;

merge\_sort(A, p, r);

printf("El resultado es: ");

for (i = 0; i < r + 1; i++)

{

printf("%i ", \*(A + i));

}

return 0;

}

**Cabecera**

#include <stdio.h>

#include <stdlib.h>

void merge(int \*array, int p, int q, int r);

void merge\_sort(int \*array, int p, int r);

**Algoritmos**

#include <stdio.h>

#include <stdlib.h>

#include "cabecera1.h"

void merge(int \*array, int p, int q, int r)

{

// Declaracion de variables

int i, j, k;

int n\_1 = (q - p) + 1;

int n\_2 = (r - q);

int \*L, \*R;

// Asignación de memoria

L = (int\*)malloc(n\_1 \* sizeof(int));

R = (int\*)malloc(n\_2 \* sizeof(int));

// Copia de datos del arreglo A en los subarreglos L y R

for (i = 0; i < n\_1; i++)

{

L[i] = \*(array + p + i);

}

for (j = 0; j < n\_2; j++)

{

R[j] = \*(array + q + j + 1);

}

i = 0;

j = 0;

for (k = p; k < r + 1; k++)

{

if (i == n\_1)

{

\*(array + k) = \*(R + j);

j = j+ 1;

}

else if(j == n\_2)

{

\*(array + k) = \*(L + i);

i = i + 1;

}

else

{

if (\*(L + i) <= \*(R + j))

{

\*(array + k) = \*(L + i);

i = i + 1;

}

else

{

\*(array + k) = \*(R + j);

j = j + 1;

}

}

}

}

void merge\_sort(int \*array, int p, int r)

{

if (p < r)

{

// Dividir el problema en subproblemas

int q = (p + r)/2;

// Resolver el problema de manera recursiva hasta llegar a una solucion trivial

merge\_sort(array, p, q);

merge\_sort(array, q + 1, r);

merge(array, p, q, r);

}

}

**Segunda Parte (Método de la Burbuja)**

**Main**

#include <stdio.h>

#include <stdlib.h>

#include "cabecera.h"

int main()

{

int x;

int arreglo[] = {30, 28, 11, 96, -5, 21, 18, 12, 22, 30, 97, -1, -40, -500};

int longitud = sizeof arreglo / sizeof arreglo[0];

printf("Arreglo sin Ordenar\n");

for( x = 0; x < longitud; x++)

{

printf("%d ", arreglo[x]);

}

printf("\n");

burbuja(arreglo, longitud);

return 0;

}

**Cabecera**

#include <stdio.h>

#include <stdlib.h>

void intercambiar(int \*a, int \*b);

void burbuja(int arreglo[], int longitud);

**Algoritmos**

#include <stdio.h>

#include <stdlib.h>

#include "cabecera.h"

void intercambiar(int \*a, int \*b)

{

int temporal = \*a;

\*a = \*b;

\*b = temporal;

}

void burbuja(int arreglo[], int longitud)

{

int x,indiceActual;

for (x = 0; x < longitud; x++)

{

for (indiceActual = 0; indiceActual < longitud - 1;indiceActual++)

{

int indiceSiguienteElemento = indiceActual + 1;

if (arreglo[indiceActual] > arreglo[indiceSiguienteElemento])

{

intercambiar(&arreglo[indiceActual], &arreglo[indiceSiguienteElemento]);

}

}

}

printf("Arreglo Ordenado:\n");

for( x = 0; x < longitud; x++)

printf("%4d ", arreglo[x]);

}

**EJ. 5:**

**EJ. 6:**