

**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: implemente un programa recursivo que permita obtener el exponente de un numero xn utilizando el siguiente algoritmo revisado con la serie de Fibonacci.**

**Imagen que contiene Texto

Descripción generada automáticamente**

**Diagrama

Descripción generada automáticamente**

**EJ. 6: MULTIPLICACION DE DOS NUMEROS POR ALGORITMO KARATSUBA**

El algoritmo dado por Anatoly Alexeevitch Karatsuba en 1960 (Karatsuba, 1962) para la multiplicación no es únicamente aplicable a dicha operación, se puede aplicar a cualquier operación algebraica que se construya sobre una operación que cumpla la propiedad distributiva con respecto a otra que componga a la misma (Ayuso, 2013-2017). De ahí que en el presente documento propongamos un algoritmo de exponenciación entre enteros basado en dicho concepto.

El algoritmo de Karatsuba es un [procedimiento para multiplicar números grandes](https://es.wikipedia.org/wiki/Algoritmo_de_multiplicaci%C3%B3n) eficientemente, que fue descubierto por [Anatolii Alexeevitch Karatsuba](https://es.wikipedia.org/wiki/Anatolii_Alexeevitch_Karatsuba" \o "Anatolii Alexeevitch Karatsuba) en 1960 y publicado en 1962.[1](https://es.wikipedia.org/wiki/Algoritmo_de_Karatsuba#cite_note-kara1962-1)​[2](https://es.wikipedia.org/wiki/Algoritmo_de_Karatsuba#cite_note-kara1995-2)​ El algoritmo consigue reducir la múltiplicación de dos números de n dígitos a como máximo {\displaystyle 3n^{\log \_{2}3}\approx 3n^{1.585}} multiplicaciones de un dígito. Es, por lo tanto, más rápido que el algoritmo [clásico](https://es.wikipedia.org/wiki/Algoritmo_de_multiplicaci%C3%B3n), que requiere n2 productos de un dígito. Si n = 210 = 1024, en particular, el cómputo final exacto es 310 = 59.049 y (210)2 = 1.048.576, respectivamente.

**Diagrama

Descripción generada automáticamente**

**Diagrama, Esquemático

Descripción generada automáticamente**

**Diagrama, Esquemático

Descripción generada automáticamente**

**Diagrama, Esquemático

Descripción generada automáticamente**

**Diagrama

Descripción generada automáticamente**

**Diagrama, Dibujo de ingeniería

Descripción generada automáticamente**

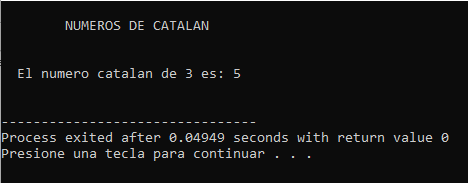
**PRUEBAS Y SALIDAS.**

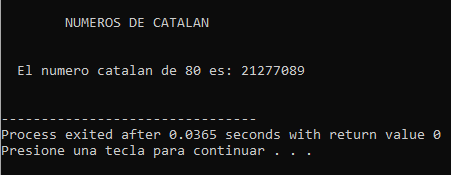
**EJ. 1:**

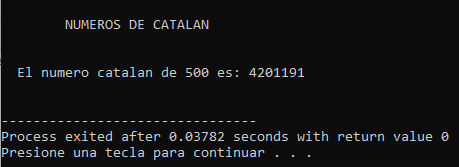
**Tabla de resultados Programa Interactivo**

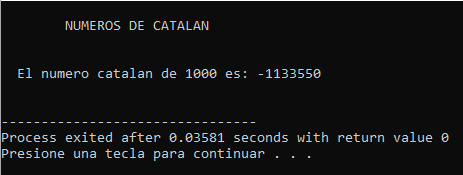
|  |  |
| --- | --- |
| **VALOR DE N** | **TIEMPO DE EJECUCIÓN** |
| 3 | 0.04949 |
| 80 | 0.0365 |
| 500 | 0.03782 |
| 1,000 | 0.03581 |
| 20,000 | 0.05561 |
| 2,960,000 | 0.2153 |
| 18 | 0.04622 |
| 800,000,000,000 | 11.24 |
| 5,000,000 | 0.2146 |
| 50 | 0.05227 |
| 900,000 | 0.1266 |
| 500 | 0.03615 |
| 15,000 | 0.04864 |
| 12,632,000 | 0.2902 |
| 63 | 0.04224 |
| 1 | 0.04661 |
| 1,500,000 | 0.1317 |
| 18,560,000 | 0.4032 |
| 2,000,000,000 | 20.1 |
| 0 | 0.06029 |
| 300,000,000,000 | 0.07632 |

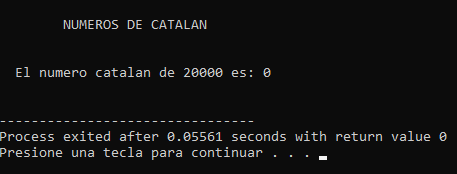
**Capturas de pantalla Interactivo**

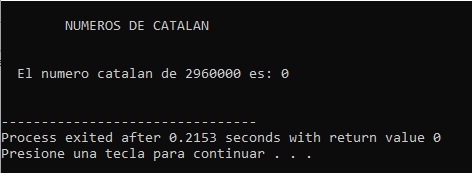


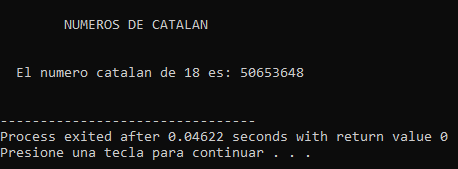


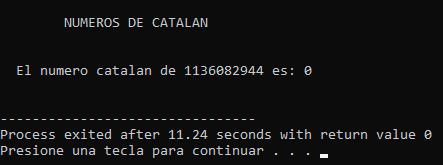


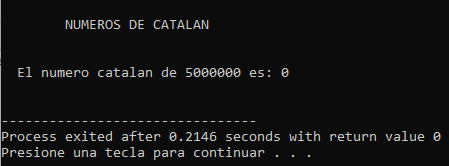


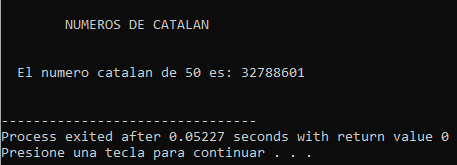


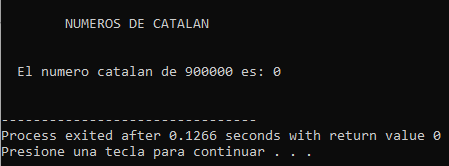


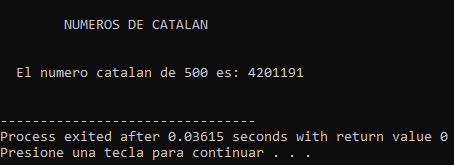


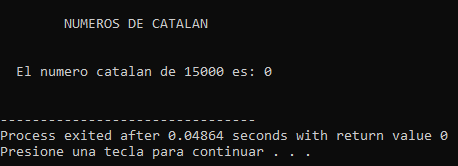


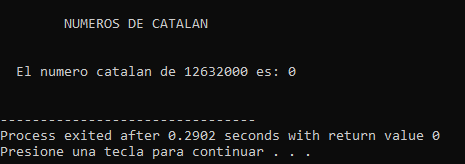


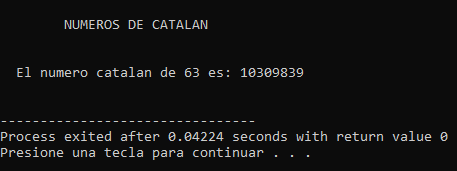


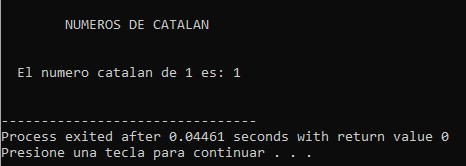


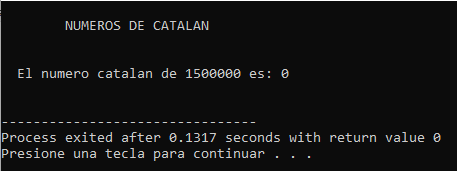


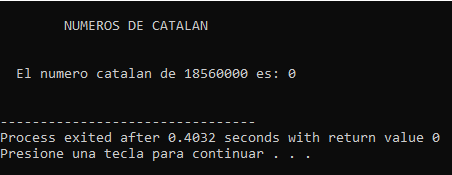


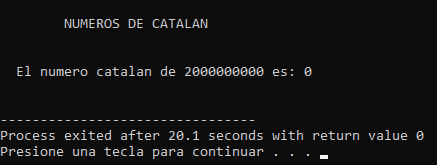


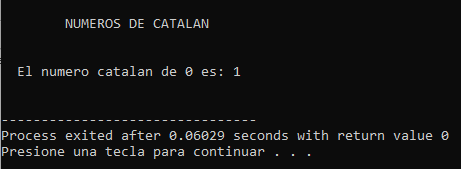


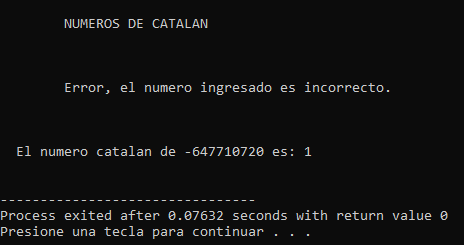








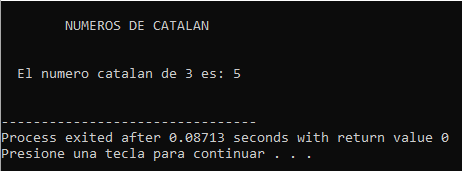


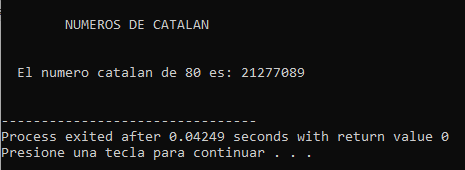


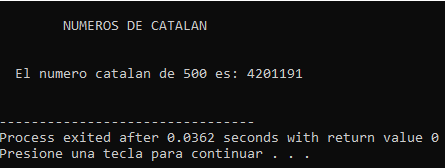
**Tabla de resultados Programa Recursivo**

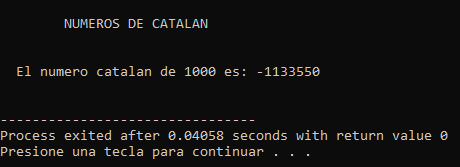
|  |  |
| --- | --- |
| **VALOR DE N** | **TIEMPO DE EJECUCIÓN** |
| 3 | 0.08713 |
| 80 | 0.04249 |
| 500 | 0.0362 |
| 1,000 | 0.04058 |
| 20,000 | 0.06988 |
| 2,960,000 | 2.457 |
| 18 | 0.03571 |
| 800,000,000,000 | 1.348 |
| 5,000,000 | 1.249 |
| 50 | 0.03592 |
| 900,000 | 1.349 |
| 500 | 0.03551 |
| 15,000 | 0.04864 |
| 12,632,000 | 1.49 |
| 63 | 0.03544 |
| 1 | 0.03447 |
| 1,500,000 | 0.1317 |
| 18,560,000 | 1.301 |
| 2,000,000,000 | 1.166 |
| 0 | 0.03608 |
| 300,000,000,000 | 0.03354 |

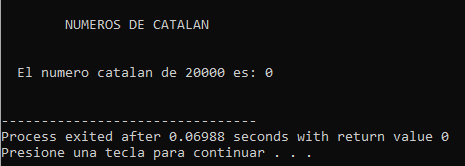
**Capturas de pantalla Interactivo**

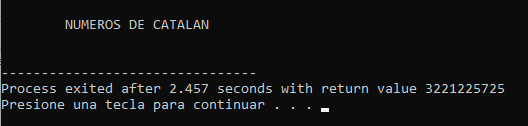


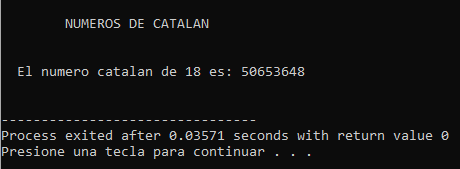


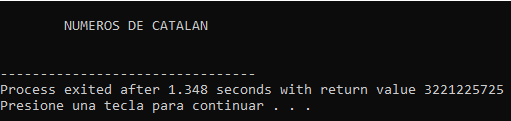


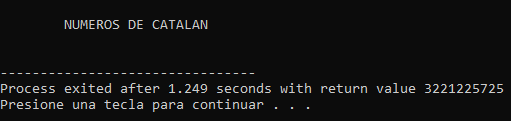


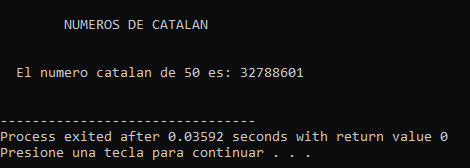


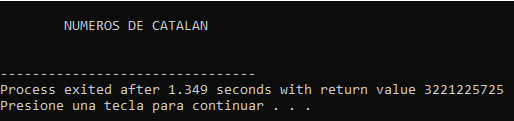


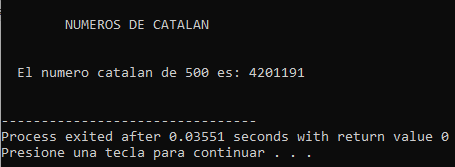


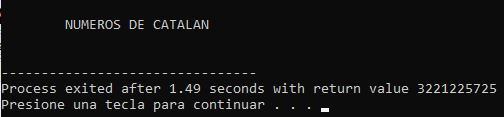


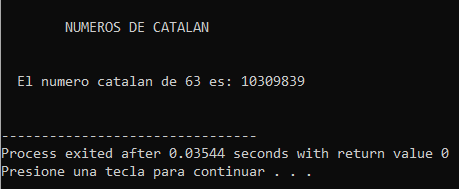


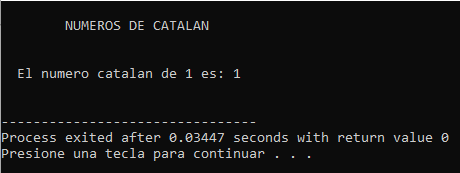


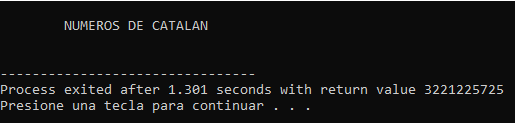


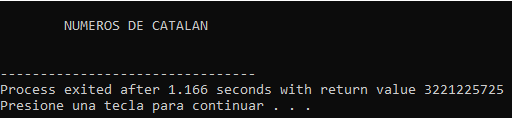


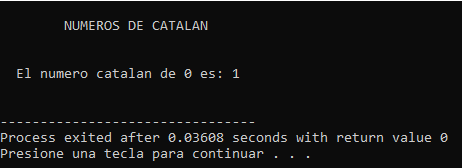




}





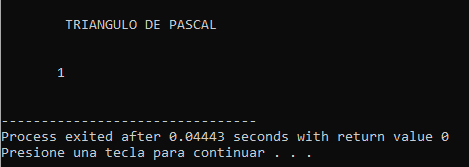


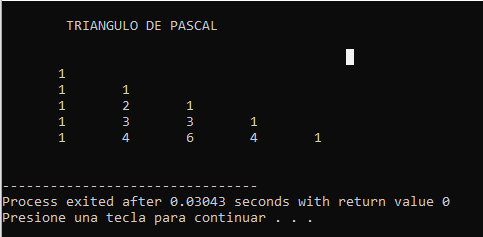
**EJ. 2:**

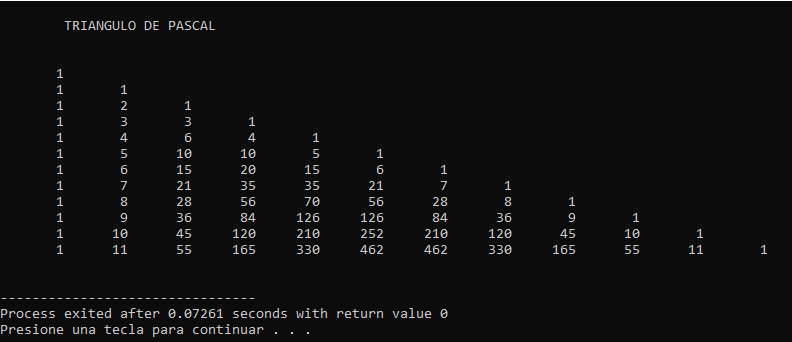
**Tabla de resultados Programa Interactivo Triángulo de Pascal**

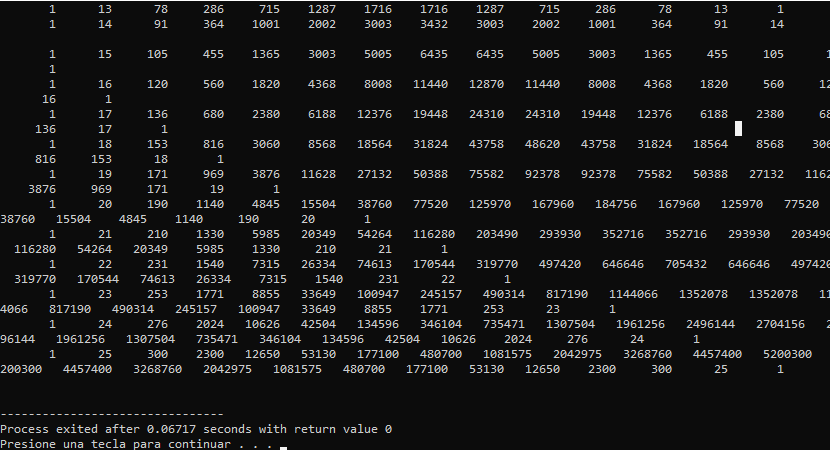
|  |  |
| --- | --- |
| **VALOR DE N** | **TIEMPO DE EJECUCIÓN** |
| 1 | 0.04443 |
| 5 | 0.03043 |
| 12 | 0.07261 |
| 26 | 0.06717 |
| 50 | 0.2138 |
| 100 | 0.6165 |
| 500 | 11.03 |
| 1,000 | 30.99 |
| 20,000 | - |
| 3 | 0.04257 |
| 500,000 | - |
| 600 | 13.35 |
| 15,000 | - |
| 2 | 0.03890 |
| 9,000 | 0.03544 |
| 30 | 0.03447 |
| 2,000 | - |
| 300,000 | - |
| 300 | 4.641 |
| 180,000 | - |
| 300,000,000,000 | - |

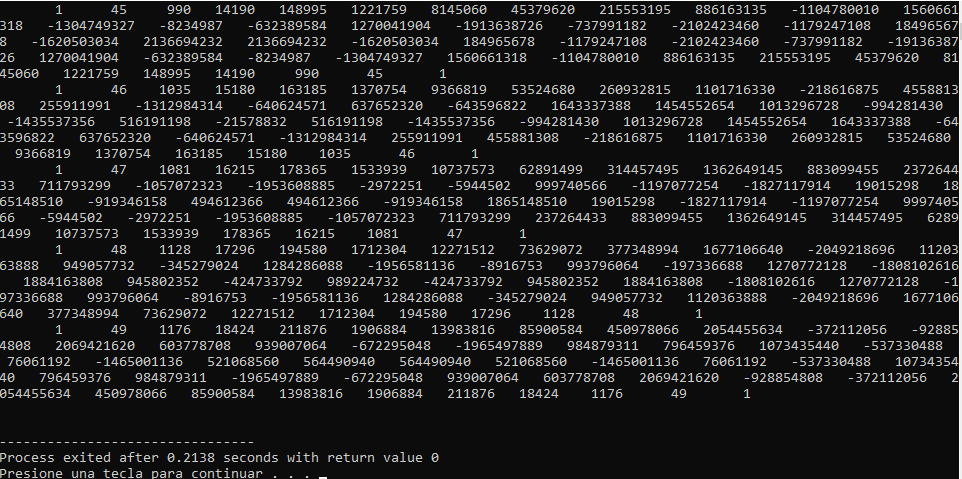
**Capturas de pantalla Interactivo Triángulo de Pascal**

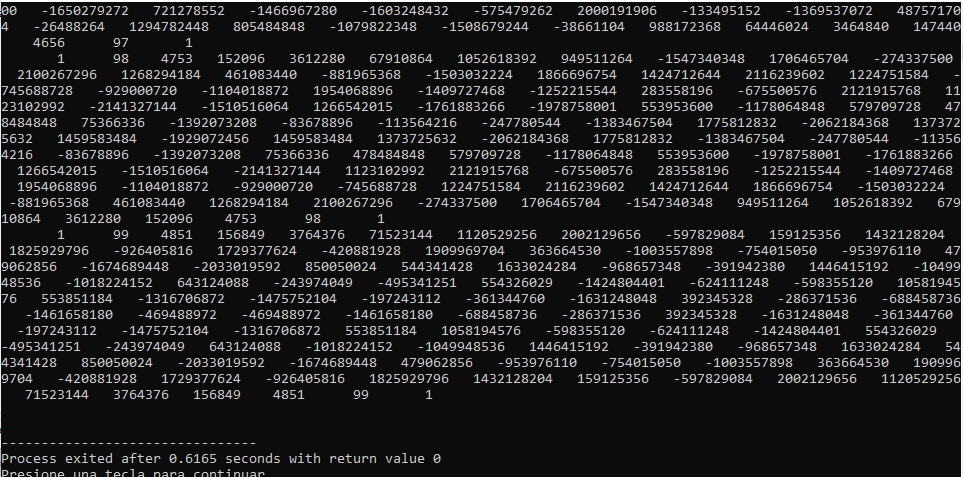


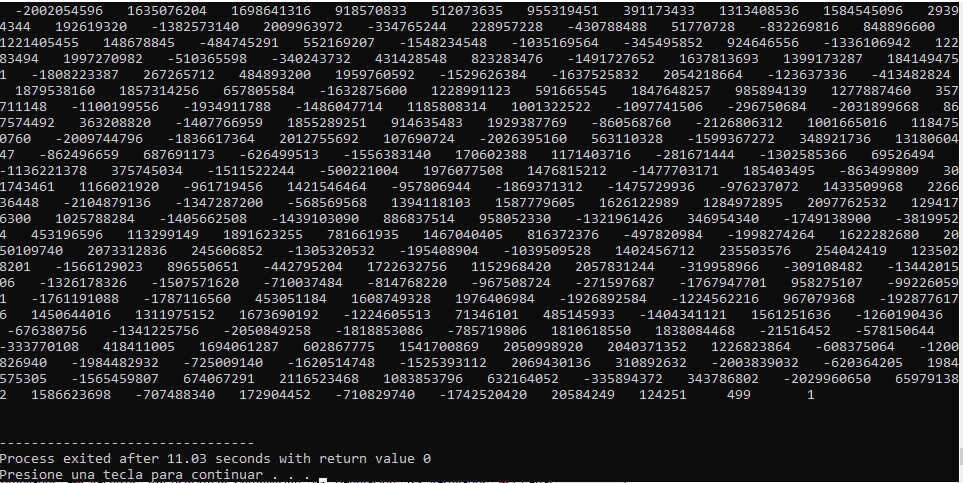


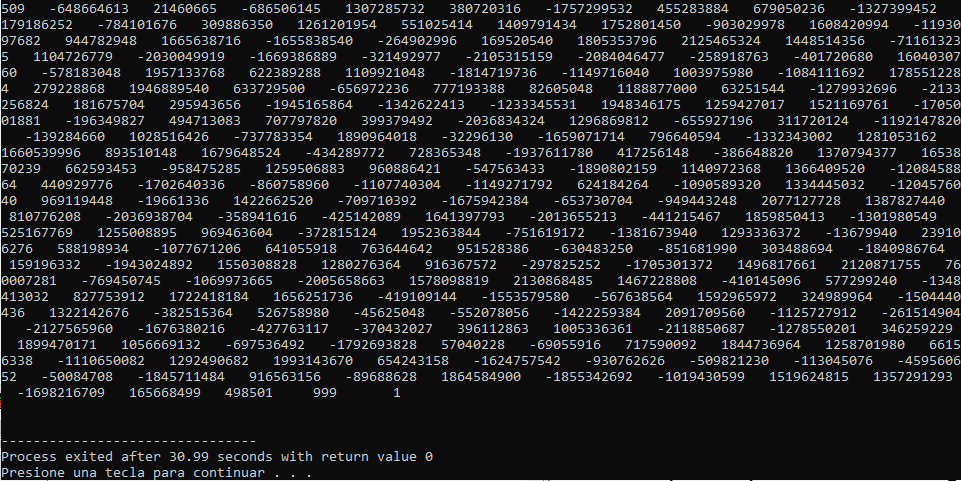


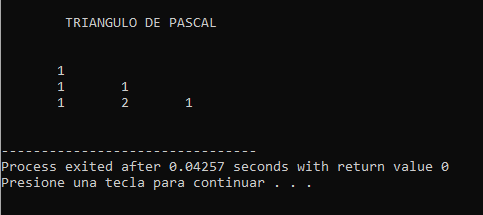


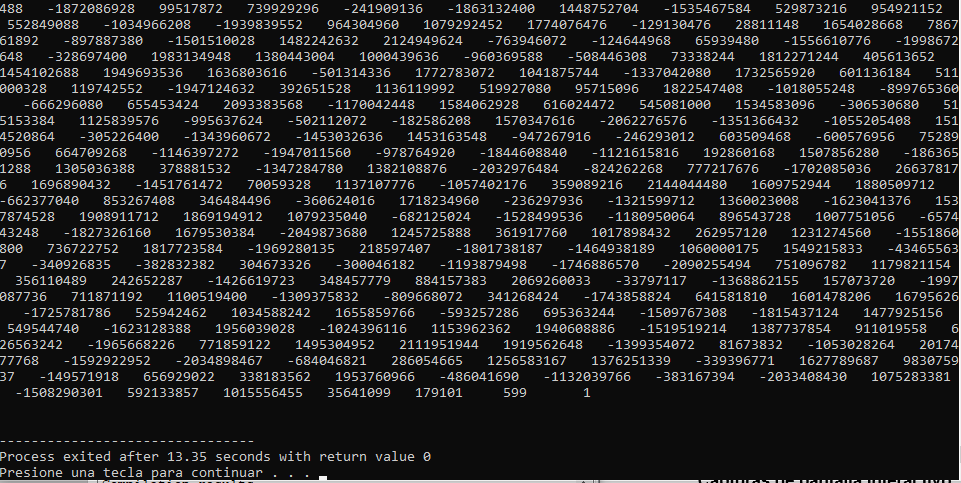


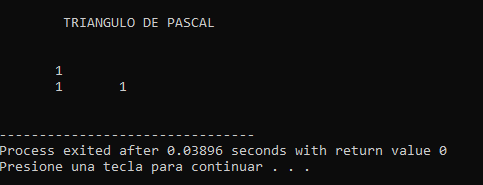


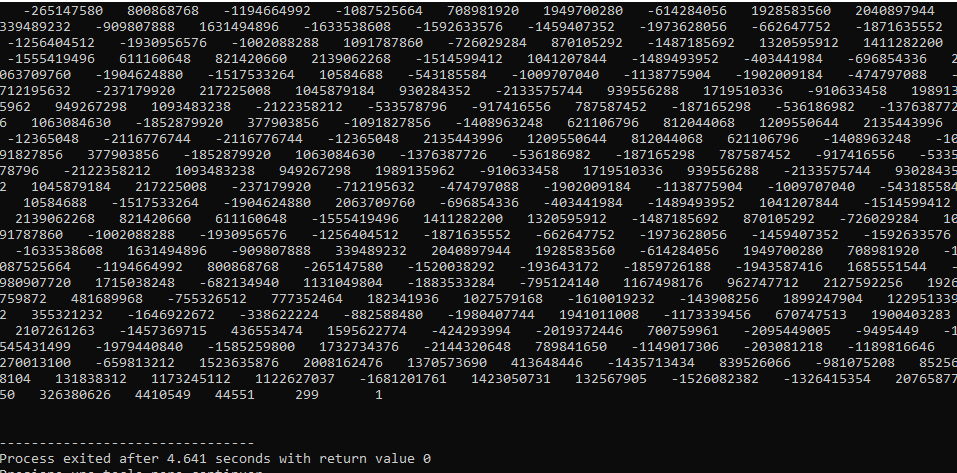




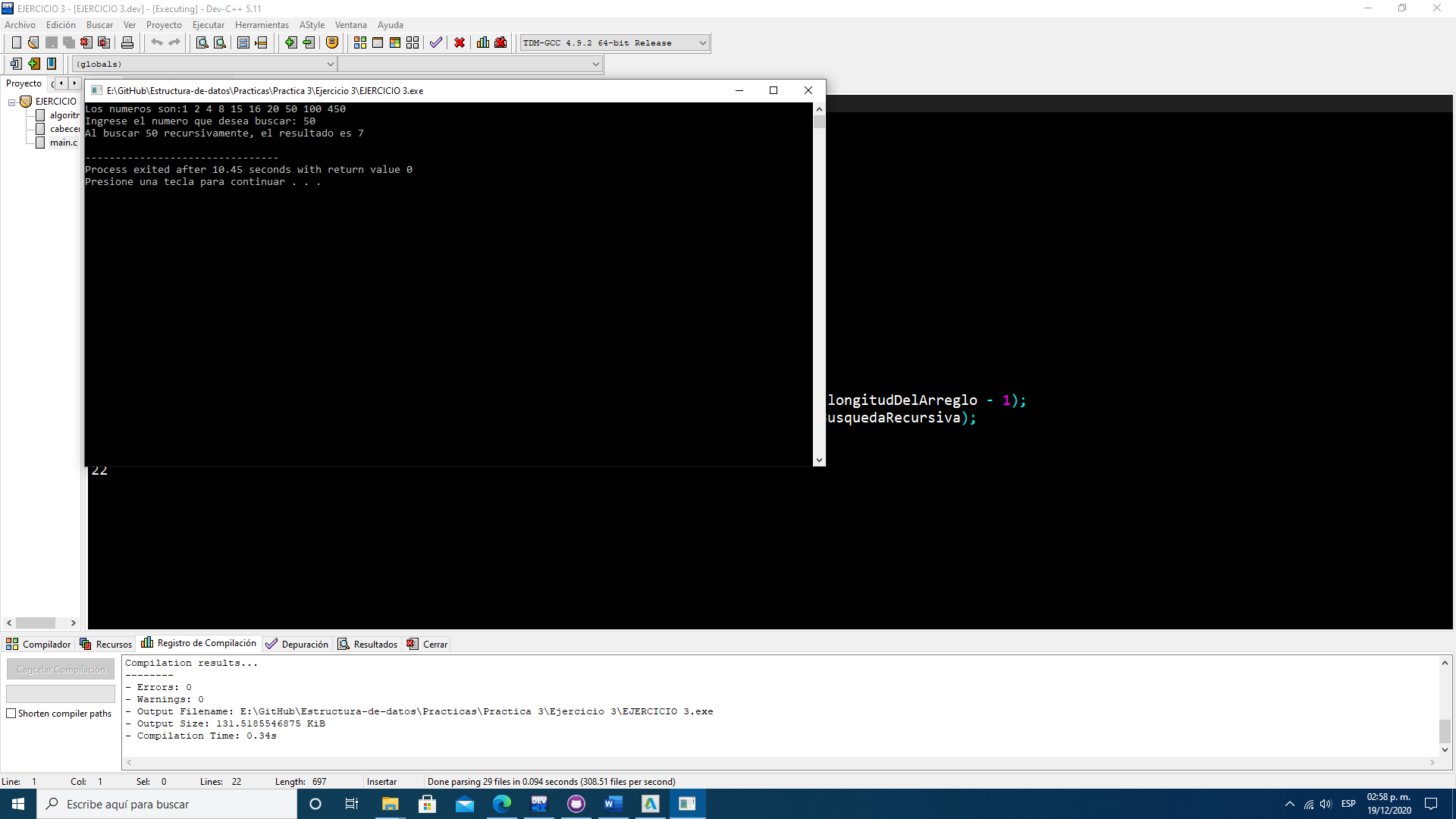






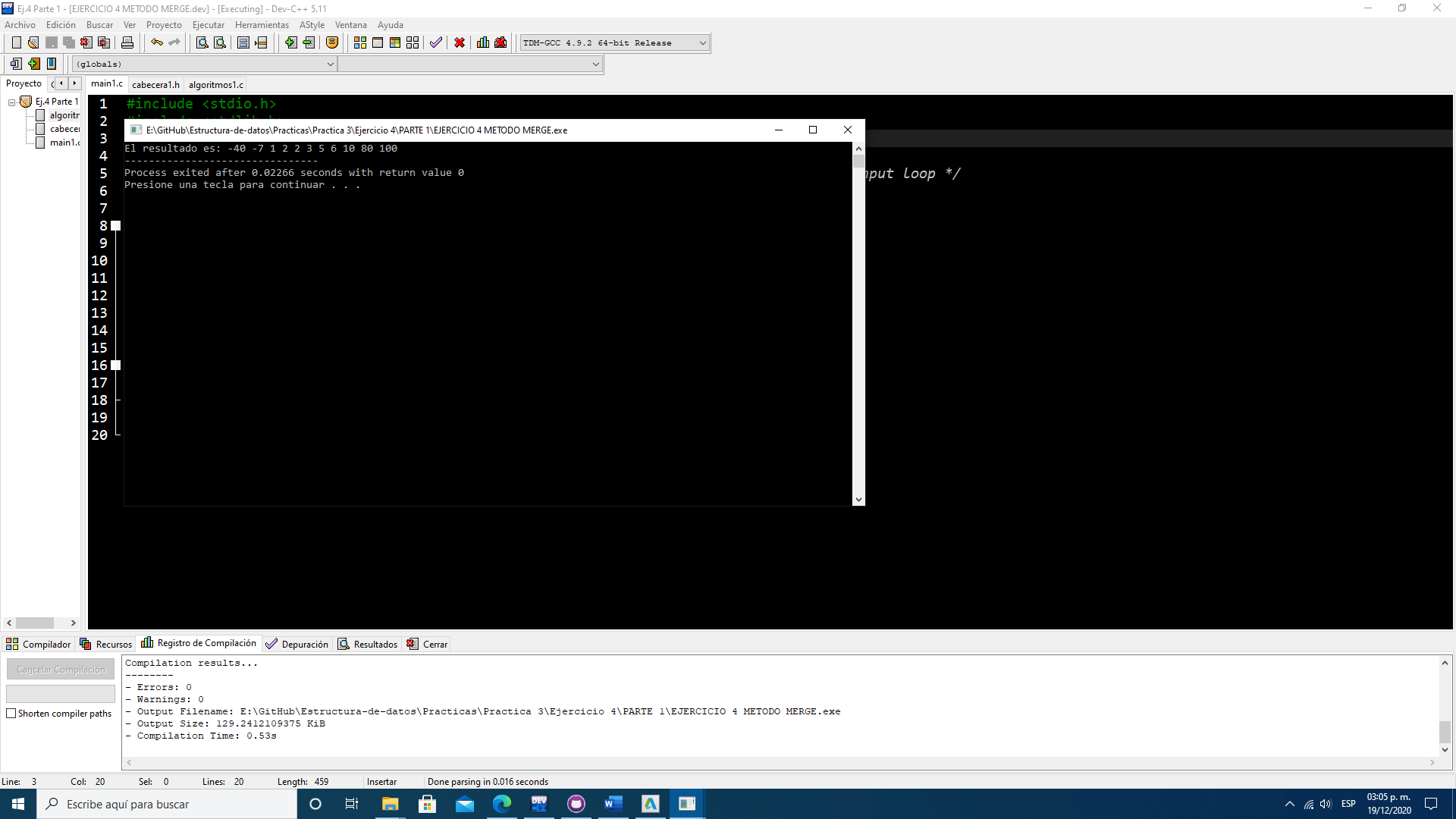


**EJ. 3:**

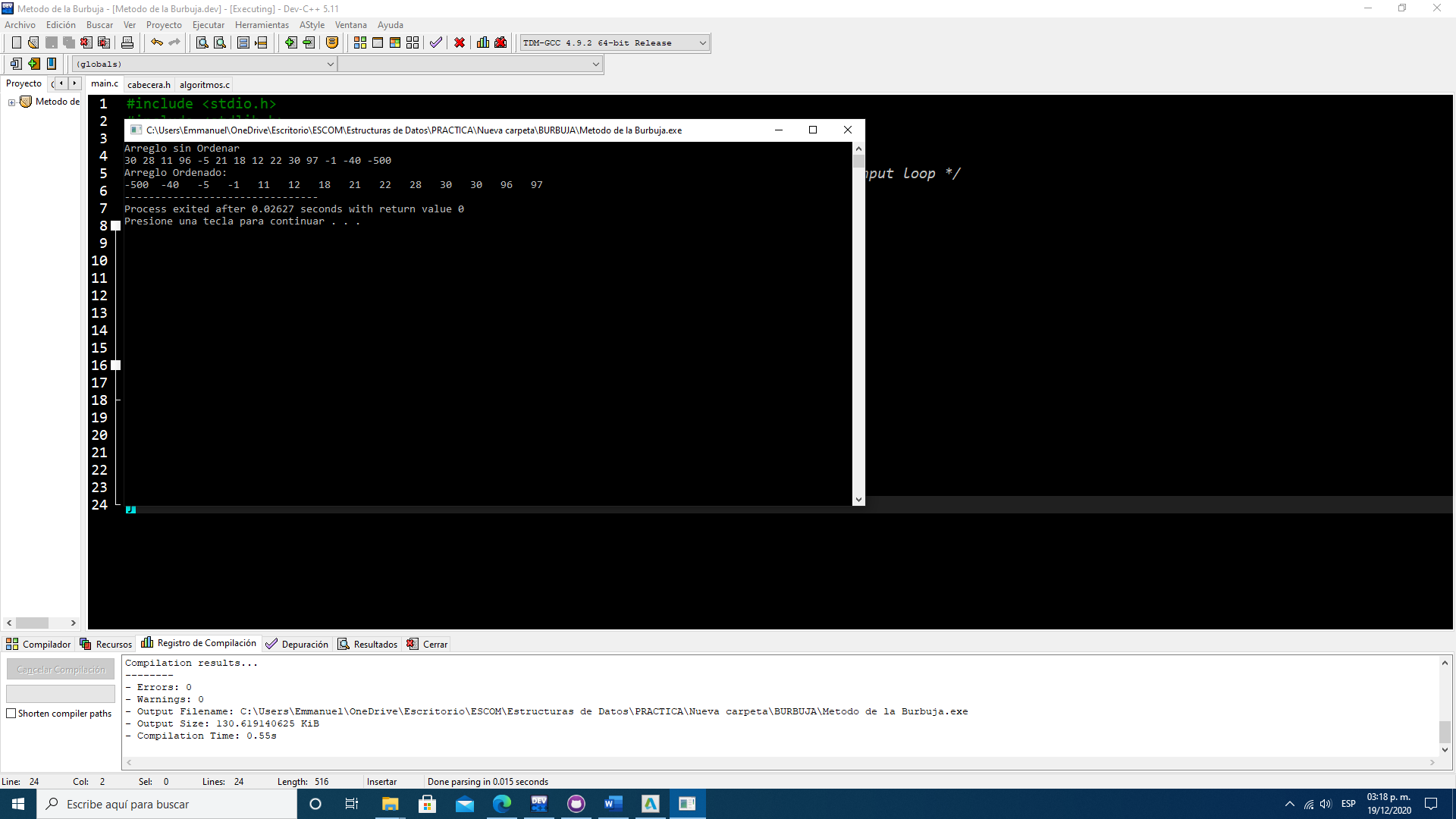


**EJ. 4:**

**Primera Parte “Merge\_Sort”**



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



**EJ. 5:**

Texto

Descripción generada automáticamente

Texto

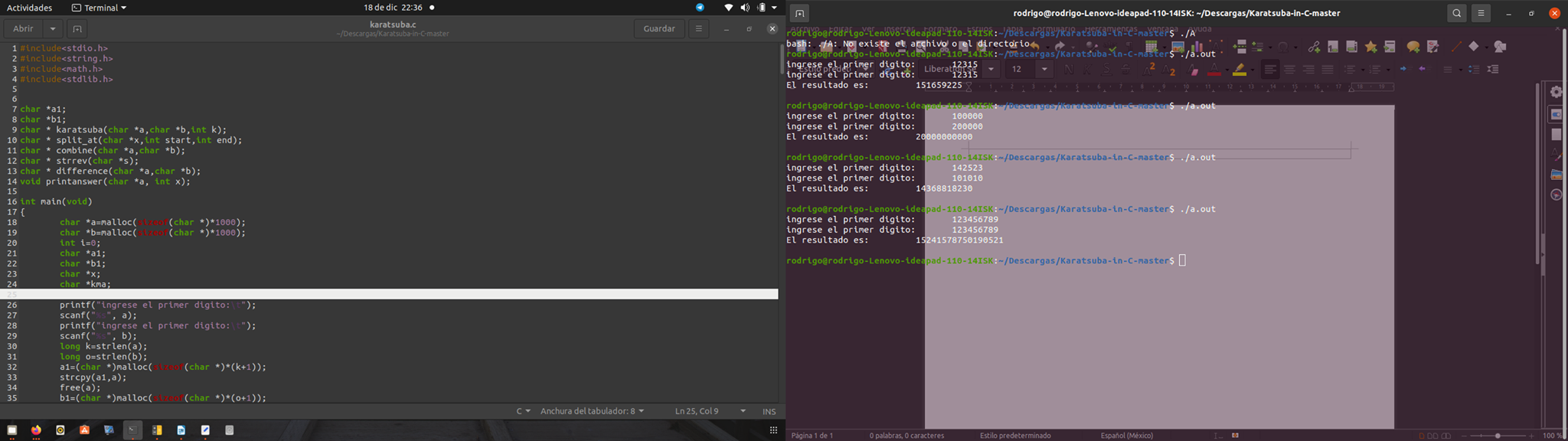
Descripción generada automáticamente

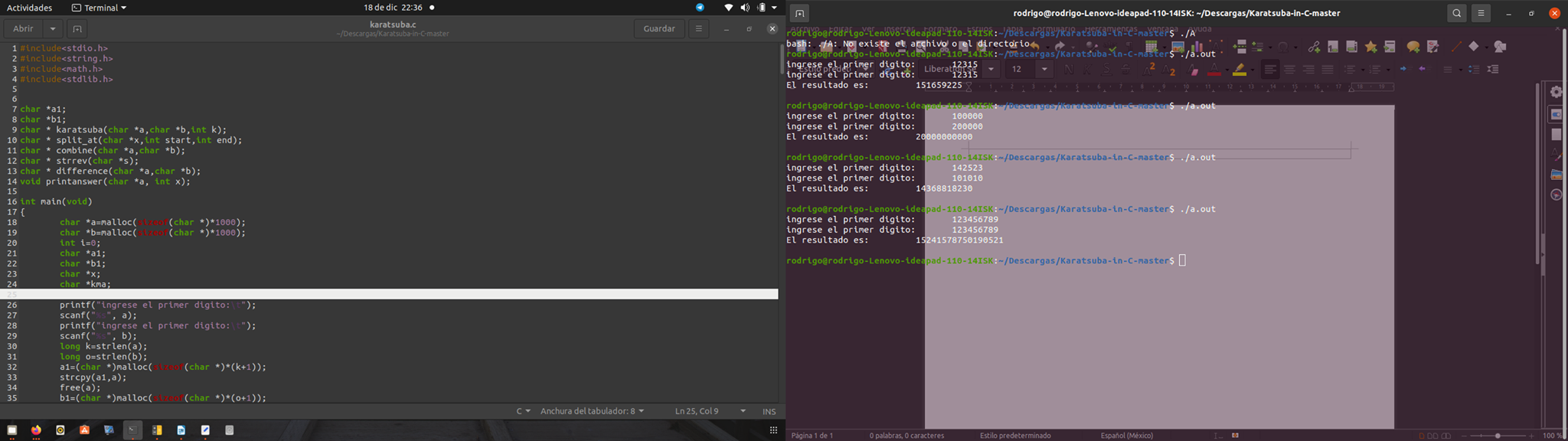
Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**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.

**Rodrigo:** El uso de la recursividad puede ser un poco confusa ya que necesitas una buena idea de como debe de funcionar de forma sólida, si no es fácil su confusión y por ende cometer algún error. El uso de la recursividad en general puede ser de gran ayuda para problemas grandes y tratar de hacerlo en pequeñas partes.

**Bryan:** El uso de la recursividad es un método muy sencillo para programar algoritmo que no sean tan complejos, ya que, si el manejo de los datos es muy complejo, el método recursivo se vuelve improductivo y con un algoritmo más complejo para la máquina. Con esta práctica refuerzo los conocimientos que poseía sobre esta herramienta de programación ya que ya conocía con anterioridad esta herramienta, sin embargo, no la había trabajado más a fondo, por lo que considero que es una práctica con muy buenos resultados. En el algoritmo del triángulo de pascal se complicó el uso de la recursividad, debido a que no tenía una forma clara de utilizar la recursividad para su implementación.

**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:Catalán Recursivo**

#include <stdio.h>

#include <stdlib.h>

int catalan( int n )

{

if ( n == 0 )

return 1;

else

return ( ( 4 \* n - 2 ) \* catalan( n - 1 ) / ( n + 1 ) );

}

int main()

{

int n;

printf( "\n\tNUMEROS DE CATALAN\n\n" );

printf( " Ingrese un numero positivo: " );

scanf( "%d", &n );

if( n < 0 )

printf( "\n\n\tError, el numero ingresado es incorrecto.\n\n\n " );

else

printf( "\n El numero catalan de %d es: %d\n\n ", n, catalan(n) );

system( "pause" );

return 0;

}

**Catalán Iterativo**

#include <stdio.h>

#include <stdlib.h>

int main()

{

int n, i, catalan = 1;

printf( "\n\tNUMEROS DE CATALAN\n\n" );

printf( " Ingrese un numero positivo: " );

scanf( "%d", &n );

if( n < 0 )

printf( "\n\n\tError, el numero ingresado es incorrecto.\n\n\n " );

else

{

for(i = 1;i <= n;i++)

catalan = catalan \* ( 4 \* i - 2 ) / ( i + 1 );

}

printf( "\n El numero catalan de %d es: %d\n\n ", n, catalan );

system( "pause" );

return 0;

}

**EJ. 2: Triangulo de Pascal Iterativo**

#include "stdio.h"

int main()

{

int n, i, j, x;

printf( "\n\tTRIANGULO DE PASCAL\n\n" );

printf( " Ingrese un numero positivo: " );

scanf( "%d", &n );

if( n < 0 )

printf( "\n\n\tError, el numero ingresado es incorrecto.\n\n\n " );

else

{

int pasc[n];

x = 0;

for(i = 1;i <= n;i++)

{

for (j=x; j>=0; j--)

{

if(j==x || j==0)

pasc[j] = 1;

else

pasc[j] = pasc[j] + pasc[j-1];

}

x++;

printf("\n");

//for (j=1; j<=n-i; j++)

printf(" ");

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

printf("%5d ", pasc[j]);

}

}

printf("\n\n ");

system("pause");

return 0;

}

**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:**