	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): MTRO. LUIS RICARDO LOPEZ VILLAFAN

Asignatura: ESTRUCTURA DE DATOS Y ALGORITMOS II

Grupo: 01

No de Práctica(s): Práctica 01. Algoritmos de ordenamiento inserción sort

Integrante(s): Lopez Cruz Jaciel Adrian

Miguel Mata Jared

No. de lista o brigada: NoCompila

Semestre: 2026-1

Fecha de entrega: 24/09/2025

Observaciones:

CALIFICACIÓN: _____

Práctica 1. Algoritmos de ordenamiento.

Ordenación por inserción

La práctica consta de un ejercicio, las instrucciones del cual se presentan a continuación. Fecha de entrega: Viernes 22 de agosto de 2025

Ejercicio 1

Se quiere definir un programa que, dada una matriz cuadrada con valores enteros, se devuelva una permutación de los valores de modo que cada renglón está ordenado de menor a mayor , y de forma que los todos los elementos del segundo renglón (de arriba para abajo) sean mayores que los del primer renglón, los del tercero más grandes que los del segundo, etc. La condición que se debe de satisfacer es la siguiente: Cuando se ordenan los elementos de un renglón, por cada elemento de dicho arreglo lineal, se mueven todos los elementos correspondientes que se encuentran por debajo en la misma columna.

Código:

```
#include <stdio.h>
#include <stdlib.h>

/*
* EJERCICIO:
* Se quiere definir un programa que, dada una matriz cuadrada con valores enteros,
* devuelva una permutación de los valores de modo que:
*
* 1. Cada renglón esté ordenado de menor a mayor.
* 2. Todos los elementos del segundo renglón (de arriba hacia abajo)
*    sean mayores que los del primero, los del tercero mayores que los del segundo, etc.
*
* Regla importante:
* - Cuando se ordenan los elementos de un renglón, por cada elemento de dicho renglón,
*   se mueven también todos los elementos que se encuentran por debajo en la misma columna.
*
* INSTRUCCIONES:
* - Completa la función sort_matrix() que debe modificar la matriz en su lugar.
* - Puedes escribir funciones auxiliares si lo deseas.
* - NO cambies la firma de sort_matrix().
*/

void sort_matrix(int **matriz, int n) {
  for (int filaObjetivo = 0; filaObjetivo < n; filaObjetivo++) {          // Recorre cada fila donde fijaremos valores de izquierda a derecha
     for (int colObjetivo = 0; colObjetivo < n; colObjetivo++) {          // Recorre cada columna destino dentro de la fila actual

        int filaMin = filaObjetivo;                                       // Fila del valor minimo encontrado (inicialmente, la celda destino)
        int colMin  = colObjetivo;                                        // Columna del valor minimo hallado
        int valorMin = matriz[filaObjetivo][colObjetivo];                 // Valor minimo actual (inicialmente, el de la celda destino)

        for (int fila = filaObjetivo; fila < n; fila++) {                 // Busca el valor minimo en la submatriz no fijada
           for (int columna = 0; columna < n; columna++) {                // Recorre todas las columnas
              if (fila == filaObjetivo && columna < colObjetivo) continue;  // No toca el prefijo ya fijado de la fila actual
              if (matriz[fila][columna] < valorMin) {                     // Si encontramos un valor más pequeño
                 valorMin = matriz[fila][columna];                        // Actualizamos el valor minimo
                 filaMin  = fila;                                         // Guardamos su fila
                 colMin   = columna;                                      // Guardamos su columna
```

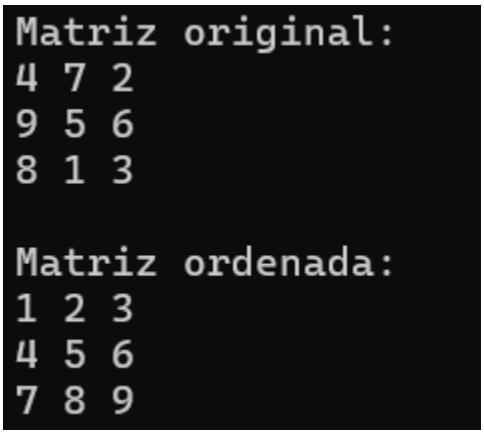
```
            }
        }
    }

    if (filaMin > filaObjetivo) {                   // El valor minimo esta por debajo de la fila objetivo
        while (colMin < colObjetivo) {                  // Lo movemos horizontalmente a la derecha
            for (int f = filaObjetivo + 1; f < n; f++) {    // Intercambiamos solo la cola
                int tmp = matriz[f][colMin];
                matriz[f][colMin] = matriz[f][colMin + 1];
                matriz[f][colMin + 1] = tmp;
            }
            colMin++;                                   // Avanzamos la columna del valor minimo
        }
        while (colMin > colObjetivo) {                  // Lo movemos horizontalmente a la izquierda
            for (int f = filaObjetivo + 1; f < n; f++) {    // De nuevo, solo en la cola (no tocamos las filas superiores)
                int tmp = matriz[f][colMin];
                matriz[f][colMin] = matriz[f][colMin - 1];
                matriz[f][colMin - 1] = tmp;
            }
            colMin--;                                   // Retrocedemos la columna del valor minimo
        }
    } else {                                        // El valor minimo ya está en la misma fila objetivo
        while (colMin < colObjetivo) {                  // Si está a la izquierda, lo movemos a la derecha
            for (int f = filaObjetivo; f < n; f++) {    // Intercambiamos columnas desde filaObjetivo hacia abajo
                int tmp = matriz[f][colMin];
                matriz[f][colMin] = matriz[f][colMin + 1];
                matriz[f][colMin + 1] = tmp;
            }
            colMin++;                                   // Avanzamos la columna del valor minimo
        }
        while (colMin > colObjetivo) {                  // Si está a la derecha, lo movemos a la izquierda
            for (int f = filaObjetivo; f < n; f++) {    // Intercambiamos las columnas desde filaObjetivo hacia abajo
                int tmp = matriz[f][colMin];
                matriz[f][colMin] = matriz[f][colMin - 1];
                matriz[f][colMin - 1] = tmp;
            }
            colMin--;                                   // Retrocedemos la columna del valor minimo
        }
    }

    while (filaMin > filaObjetivo) {                    // Una vez alineada la columna, subimos el valor minimo verticalmente
        int tmp = matriz[filaMin][colObjetivo];
        matriz[filaMin][colObjetivo] = matriz[filaMin - 1][colObjetivo];
        matriz[filaMin - 1][colObjetivo] = tmp;
        filaMin--;                                      // Seguimos subiendo hasta la fila objetivo
    }
  }
 }
}

int main() {
  int n = 3;
  // Reserva dinámica de la matriz
  int **matrix = (int **)malloc(n * sizeof(int *));
  for (int i = 0; i < n; i++) {
    matrix[i] = (int *)malloc(n * sizeof(int));
  }

  // Ejemplo de entrada
  int ejemplo[3][3] = {
    {4, 7, 2},
    {9, 5, 6},
    {8, 1, 3}
  };

  // Copiar ejemplo a la matriz dinámica
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      matrix[i][j] = ejemplo[i][j];
    }
  }

  printf("Matriz original:\n");
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      printf("%d ", matrix[i][j]);
    }
    printf("\n");
  }
```

```
    // Llamada a la función que deben completar
    sort_matrix(matrix, n);

    printf("\nMatriz ordenada:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            printf("%d ", matrix[i][j]);
        }
        printf("\n");
    }

    // Liberar memoria
    for (int i = 0; i < n; i++) {
        free(matrix[i]);
    }
    free(matrix);

    return 0;
}
```

Comentarios

El código funciona construyendo la matriz ordenada de manera gradual, celda por celda, buscando en la submatriz restante el valor mínimo para colocar en la posición actual (filaObjetivo, colObjetivo). Cuando el valor mínimo se encuentra en otra columna, se mueve horizontalmente hacia la posición deseada intercambiando toda la “cola” de la columna desde la fila actual hacia abajo, respetando la regla de mover columnas completas. Una vez alineado horizontalmente, si el mínimo está en una fila inferior, se eleva verticalmente en su columna hasta la fila objetivo. Este proceso se repite para cada celda de la matriz, garantizando que cada renglón quede ordenado de menor a mayor, que los renglones superiores contengan valores menores que los inferiores, y que todos los movimientos se realicen únicamente dentro de la propia matriz sin usar arreglos auxiliares.

Corrida del código:



Conclusiones:

- Lopez Cruz Jaciel Adrian:

La práctica fue un reto interesante, ya que al completar el código tuvimos que analizar con detalle el funcionamiento del algoritmo de inserción. El proceso resultó complejo porque fue necesario razonar cada paso para que el ordenamiento funcionara correctamente en toda la matriz. A pesar de la dificultad, la experiencia permitió reforzar la comprensión del insertion sort y mejorar nuestra capacidad de análisis y resolución de problemas.

- Miguel Mata Jared:

En conclusión, la actividad nos permitió aplicar un algoritmo insertion sort para ordenar una matriz respetando las reglas y restricciones marcadas en el problema, a su vez, se reforzaron los conceptos de ordenamiento, manipulación de matrices y programación en C. Además, de los conceptos tomados en clase sobre complejidad de algoritmos, podemos deducir que nuestro código es de orden O(n2) para el peor de los casos, dado que se usan dos ciclos anidados para su resolución.