	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): M.C. Jorge Alberto Solano Galvez

Asignatura: Estructura de Datos y Algoritmo II

Grupo: 4

No de Práctica(s): 3 - Algoritmos de ordenamiento. Parte 3

Integrante(s): González Barragán Abraham Elienai

No. de lista o brigada:

Semestre: 2026-1

Fecha de entrega: 5 de septiembre de 2025

Observaciones:

CALIFICACIÓN: _____

1. Objetivo

Conocer e identificar la estructura de los algortimos de ordenamiento **Counting sort** y **Radix sort.**

2. Descripción de Actividades

- Implementar en lenguaje Pyhton los algoritmos de ordenamiento **counting sort** y de **radix sort** que permitan ordenar nodos.
- A partir de los algoritmos implementados en Python, obtener los polinomios del mejor, el peor y el caso promedio de complejidad de **counting sort** y de **radix sort**.
- A partir del algoritmo implementado en Python, generar la gráfica del tiempo de ejecución para los algoritmos de **quick sort** y de **merge sort** para diferentes instancias de tiempo (listas de 1 a 1000 elementos) para los siguientes casos:
  - El mejor caso
  - El peor caso
  - El caso promedio

3. Resultados Obtenidos

El código fuente y las gráficas obtenidas se encuentran en el archivo "practica3.ipynb".

4. Conclusiones

Al finalizar esta práctica, quedó claro que la eficiencia de un algoritmo no es un concepto abstracto, sino uno que depende directamente de la naturaleza de los datos, lo que en la teoría parece rápido, en la práctica puede ser sorprendentemente lento.

El aprendizaje más significativo vino de comparar Counting Sort y Radix Sort. Aunque ambos prometen ser veloces, su comportamiento reveló una gran diferencia. Counting Sort es un algoritmo de fortaleza engañosa: su complejidad teórica $O(n+k)$ lo hace parecer ideal, pero esta se convierte en su mayor debilidad cuando el rango de valores (k) es mucho mayor que la cantidad de elementos (n), degradando su rendimiento a un lento $O(n)$.

En contraste, donde Counting Sort falla, Radix Sort brilla, este demostró ser mucho más robusto frente a rangos de datos amplios, su rendimiento $O(d * n)$ no depende del valor máximo, sino del número de dígitos (d), lo que le permite manejar el mismo escenario çuadráticoçon una eficiencia mucho mayor de $O(nlogn)$.

Además, la función para convertir una palabra a número demostró que es posible transformar un problema complejo, como ordenar palabras, en uno simple de ordenar números, mostrando la flexibilidad de estos algoritmos, el proceso de graficar también fue revelador, ya que un pequeño error en la implementación del código puede provocar una gran distorsión en los resultados, algo que no siempre es evidente sin una visualización.

En conclusión, esta práctica me enseñó que no se trata de elegir el algoritmo teóricamente más rápido, sino el más adecuado, la elección correcta es un balance entre la simpleza de la teoría y la complejidad de la realidad de los datos.