	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): 2 - Algoritmos de ordenamiento. Parte 2

Integrante(s): González Barragán Abraham Elienai

No. de lista o brigada:

Semestre: 2026-1

Fecha de entrega: 29 de agosto de 2025

Observaciones:

CALIFICACIÓN: _____

1. Objetivo

Conocer e identificar la estructura de los algortimos de ordenamiento **quick sort y heap sort.**

2. Descripción de Actividades

- Implementar en lenguaje Pyhton los algoritmos de ordenamiento **quick sort** y de **heap sort**.
- A partir de los algoritmos implementados en Python, obtener los polinomios del mejor, el peor y el caso promedio de complejidad de **quick sort** y de **heap 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 "practica2.ipynb".

4. Conclusiones

Se cumplió satisfactoriamente el objetivo de esta práctica, comprendí y analicé heap y quick sort, siendo 2 algoritmos bastante diferentes en cuanto algoritmos de ordenamiento.

Aunque ambos son bastantes parecidos en cuanto complejidad, el más sencillo de a mi parecer fue heap sort, ya que tiene una implementación parecida a merge sort, ambos son $O(nlog_2(n))$, heap sort es bastante eficiente independientemente del caso, o el tamaño de la lista, esto también se ve reflejado en el tiempo que tarda ejecutarse.

Por otro lado quick sort, fue más dificil no tanto de implementar sino el análisis de los casos ya que depende no solo de los valores que están en la lista, sino del pivote que se usa para dividir el arreglo, aprendí que a diferencia de otros algoritmos para el análisis de casos debemos de tener en considerar no solo el tamaño de la lista, sino los valores de la lista y el pivote que tomo para dividir, porque mientras el mejor caso y también el promedio pueden analizarse tomando al pivote como el elemento de en medio, sin embargo, el peor caso teórico es cuando el pivote se toma al primer elemento en un arreglo ordenado ascendentemente, esto cambia el algoritmo en lo mínimo para lo cambia con un if.

Además, el análisis del modelo RAM y los polinomios resultantes confirmaron el resultados de las gráficas reforzaron la comprensión de la complejidad de ambos algoritmos, mientras la complejidad de heap en todos los casos es $O(nlog_2(n))$ porque no importan los elementos de la lista siempre se ejecutan los ciclos, y heap por otro lado dependiendo del caso tiene $O(nlog_2(n))$ ó $O(n^2)$ ya que en el peor caso el funcionamiento es similar a bubble sort, lo cual hizo todo el sentido porque en el peor caso la división que se realiza es solo en un subarreglo, esto es lo que le da la complejidad cuadrática.