Estructura de Datos y Algoritmos 2

Semestre 2023-1

Profesor: Jorge Alberto Solano Galvez

Práctica 02. Algoritmos de ordenamiento. Parte 2.

Grupo 06

Integrantes:
Aguilar Martinez Erick Yair
Casillas Herrera Leonardo Didier

Práctica 2

Objetivo:
El estudiante conocerá e identificará la estructura de los algoritmos de ordenamiento Quick Sort y Heap Sort.

- Implementar el algoritmo quick sort en lenguaje Python para ordenar una secuencia de datos.

- Implementar el algoritmo heap sort en lenguaje Python para ordenar una secuencia de datos.

Instrucciones:
- Implementar en lenguaje Python los algoritmos de ordenamiento quick sort y de heap sort que permitan ordenar números u objetos
- 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 heap sort para diferentes instancias de tiempo (listas de 1 a 1000 elementos) para los siguientes casos:
  – El mejor caso: con una lista ordenada
  – El peor caso: lista ordenada en forma inversa
  – El caso promedio: una lista aleatoria

La práctica se puede realizar de manera individual, en parejas o en equipo de 3 integrantes.
Cada persona debe subir su práctica a Google Classroom porque las conclusiones son individuales.
Durante la revisión de la práctica se le puede cuestionar a cualquier integrante del equipo, la calificación es única para todo el equipo.

Resultados Obtenidos.

QuickSort - Gráficas.

Mejor caso de complejidad.

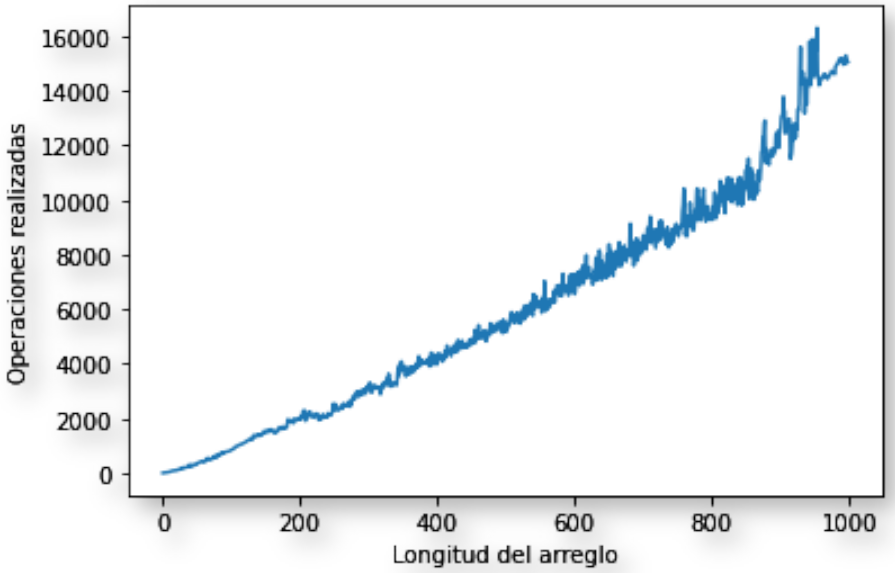

Fig 1. Conjunto de datos distribuidos de manera aleatoria.

Peor caso de complejidad.

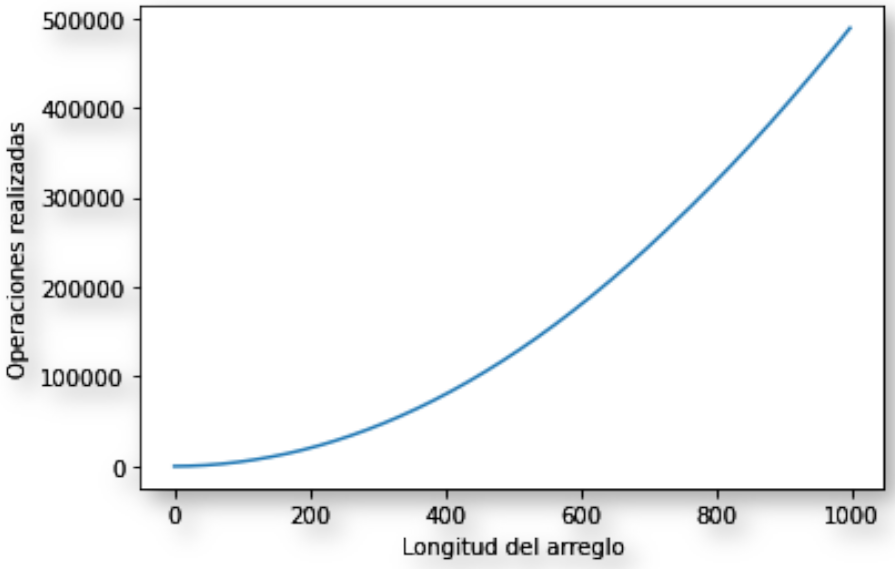

Fig 2. Conjunto de datos ordenado de manera ascendente.

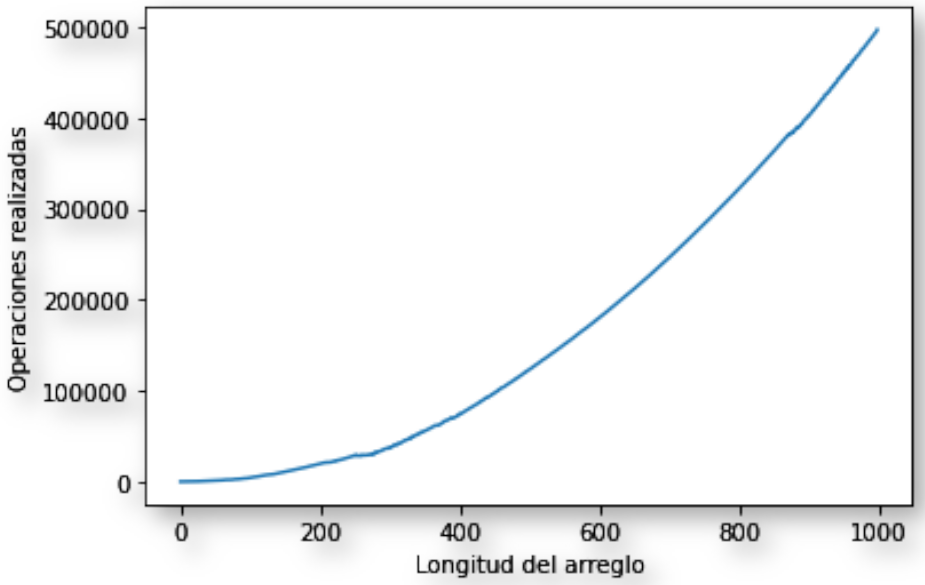

Fig 3. Conjunto de datos ordenados de manera descendente.

HeapSort - Gráficas.
Mejor, Peor y Caso promedio de complejidad.

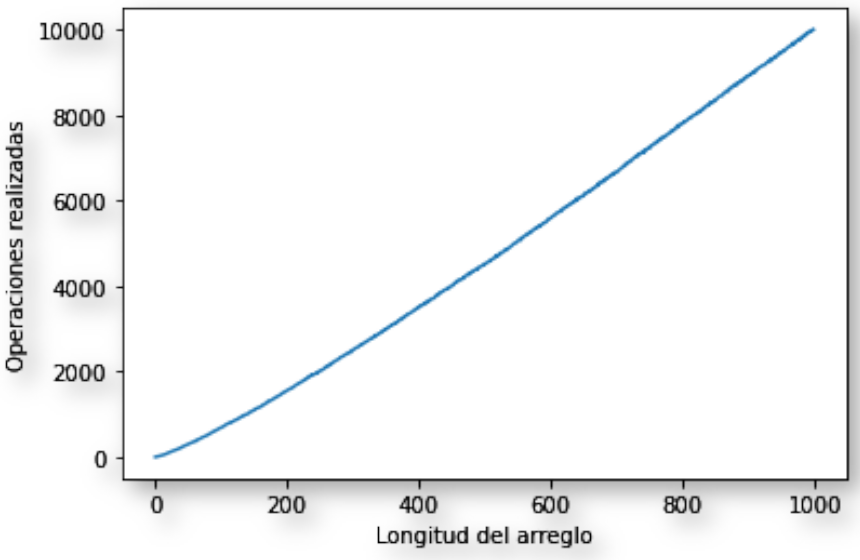

Fig 4. Conjunto de datos distribuidos de manera aleatoria.

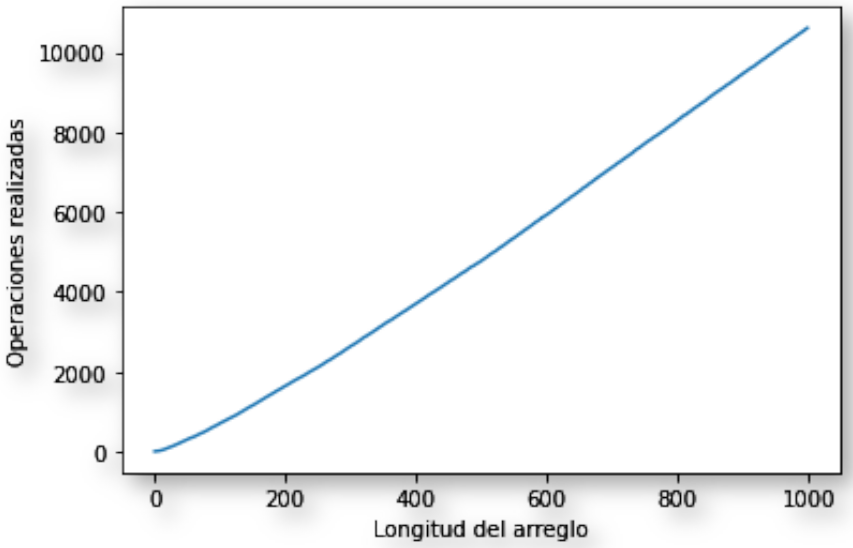

Fig 5. Conjunto de datos distribuidos de manera ascendente.

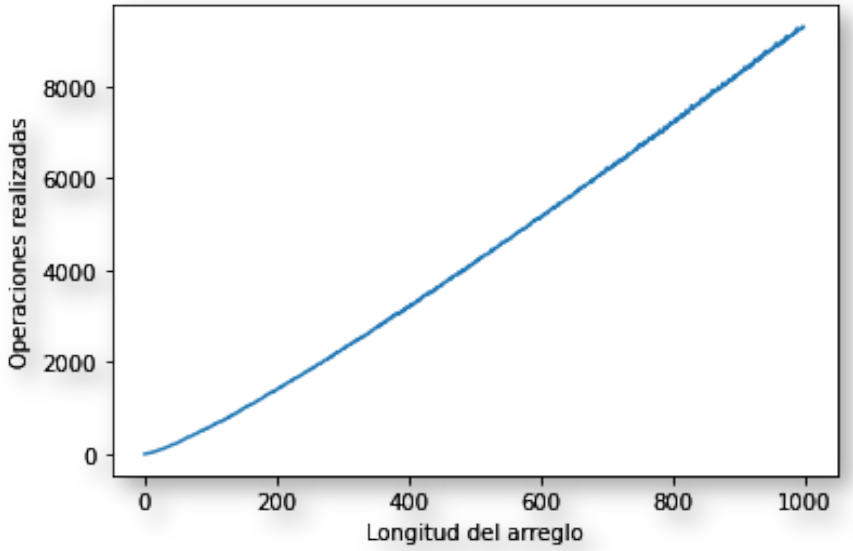

Fig 6. Conjunto de datos distribuidos de manera descendente.

Conclusiones.

- Aguilar Martinez Erick Yair: Analizar un algoritmo puede ser tan extenso como uno desee. Pensar en los casos especiales como el peor, mejor y promedio implica que se domina a la perfección la idea tras la cual el algoritmo es capaz de dar solución a su tarea y contrastar con los resultados arrojados por nuestras gráficas.

- Casillas Herrera Leonardo Didier: Un algoritmo puede tener distintas complejidades dependiendo de cómo está ordenado nuestro conjunto como fue en el caso del algoritmo QuickSort en donde la complejidad para el peor caso cambió al estar ordenado el conjunto de manera ascendente o descendente, por eso es importante realizar los análisis de algoritmos para saber cuál nos conviene utilizar dependiendo de los recursos de los que dispongamos, del conjunto que queremos ordenar y del tiempo que tarda el algoritmo para ordenar.