Universidad Nacional de Entre Ríos

➢*Facultad:* Facultad de Ingeniería.

➢*Carrera:* Licenciatura en Bioinformática y Bioingeniería.

➢*Cátedra:* Algoritmos y estructuras de datos

➢*Docentes:* Javier E. Diaz Zamboni, Jordán F. Insfrán, Juan F. Rizzato

➢*Título del trabajo:* "Aplicaciones de TADs - "*Problema 3 ""*.

➢*Alumnos:*

- Almirón Spahn, María Paz
- Leiva, Giuliana
- Saravia, Lucía Milagros

➢*Fecha de entrega:* 26 de Septiembre del 2025

Problema 3:

En este caso, se nos requería implementar los algoritmos de ordenamiento burbuja, quickSort y radix Sort. De la misma manera, debíamos realizar una prueba para verificar el correcto funcionamiento de cada una, y también, medir los tiempos de ejecución de cada uno de los métodos y realizar sus respectivas gráficas.

Análisis a priori:

- **Ordenamiento burbuja**: se basa en comparaciones e intercambios sucesivos de pares adyacentes. Presenta una complejidad de $O(n^2)$ en el *peor* y el *promedio* de los casos. Este tipo de ordenamiento resulta poco eficiente para listas de gran tamaño.
- **QuickSort**: utiliza la estrategia de "divide y vencerás". El resultado es un ordenamiento $O(n^2)$ en el *peor* caso, mientras que, en el *promedio* de los casos $O(n\ log\ n)$.
- **Radix Sort**: Ordena los números procesando los dígitos de manera sucesiva, sin comparaciones directas. El resultado de este tipo de ordenamiento será **O(n+k)**, donde *n* es el número de elementos y *k* es el número de dígitos del número más grande en la lista a ordenar.

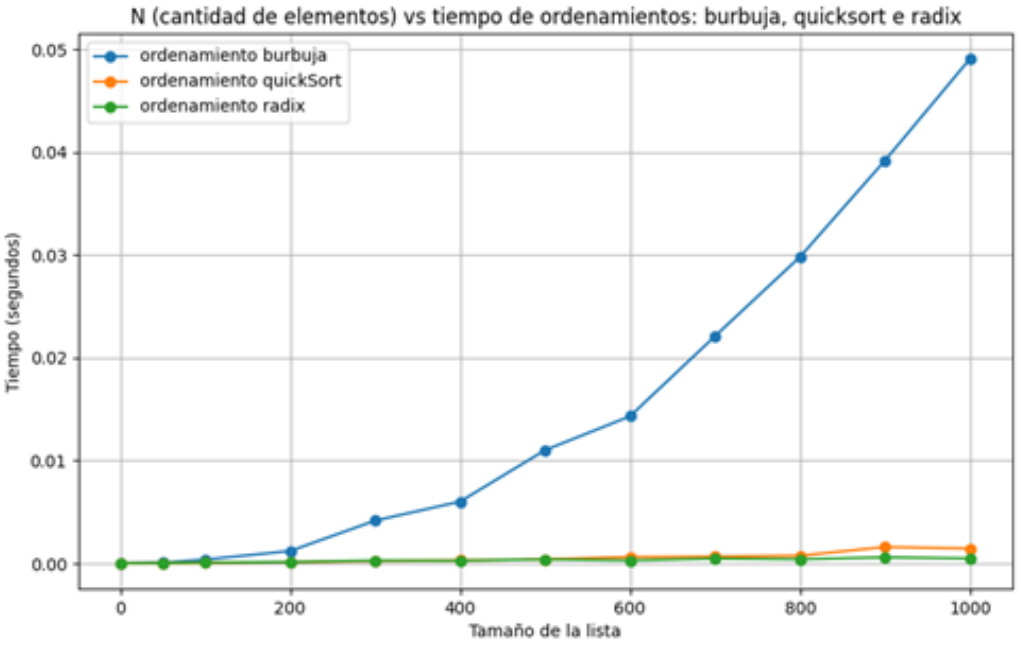

Figura 3: N (cantidad de elementos) vs tiempo de ordenamiento: burbuja, quicksort y radix

Debido a que el ordenamiento burbuja requiere más tiempo que los demás ordenamientos, para poder examinar mejor los ordenamientos de quickSort y radixSort, creamos otra gráfica (sin el ordenamiento burbuja), la cual nos permite visualizar correctamente el orden de complejidad de cada uno de los distintos métodos.

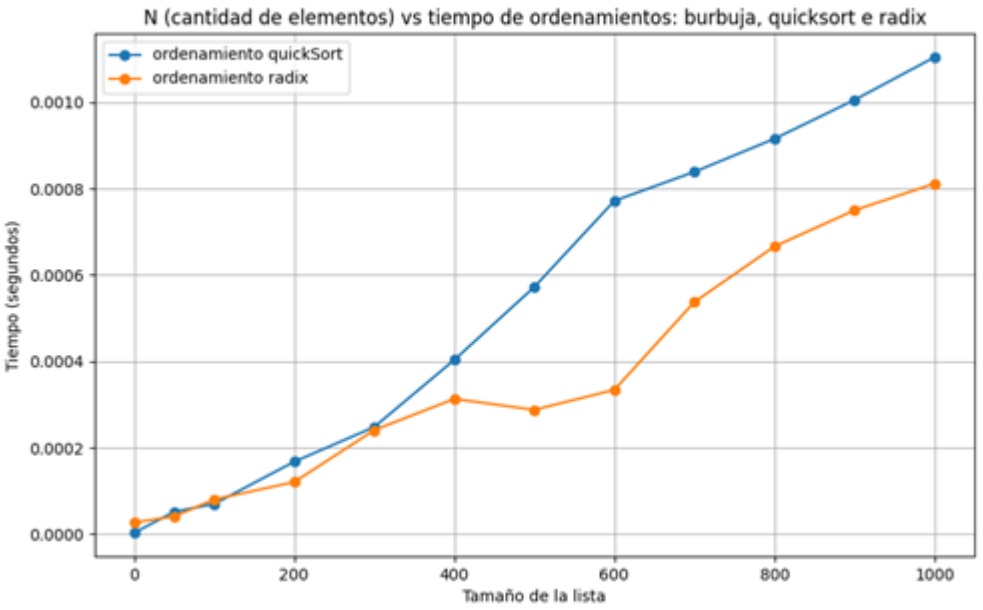

Figura 4: N (cantidad de elementos) vs tiempo de ordenamiento: quicksort y radix

Sorted:

Por otra parte, se nos solicitó investigar sobre la función built-in de python ***sorted,*** como es su funcionamiento y compararla con nuestro código. Esta función se utiliza para ordenar cualquier elemento iterable de forma ascendente (como listas, tuplas, diccionarios) y devuelve una nueva lista con los elementos ordenados, sin modificar el objeto original.