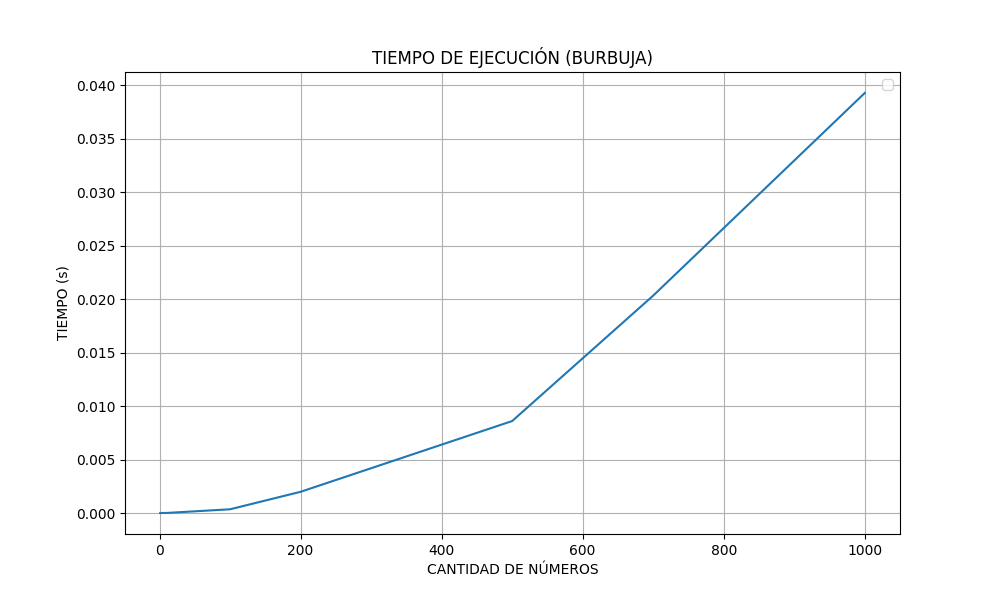
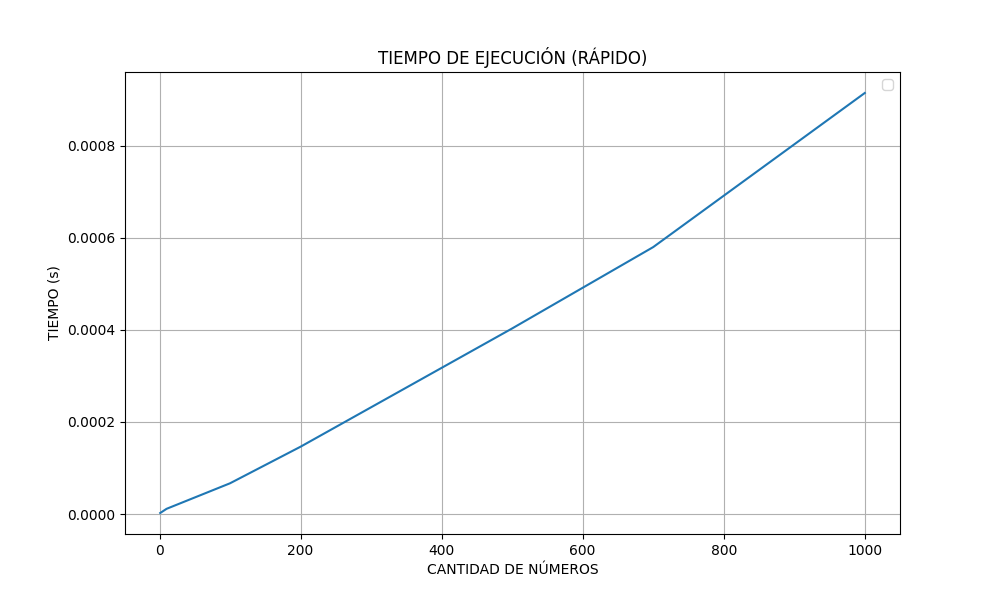
**ANALISIS A PRIORI**

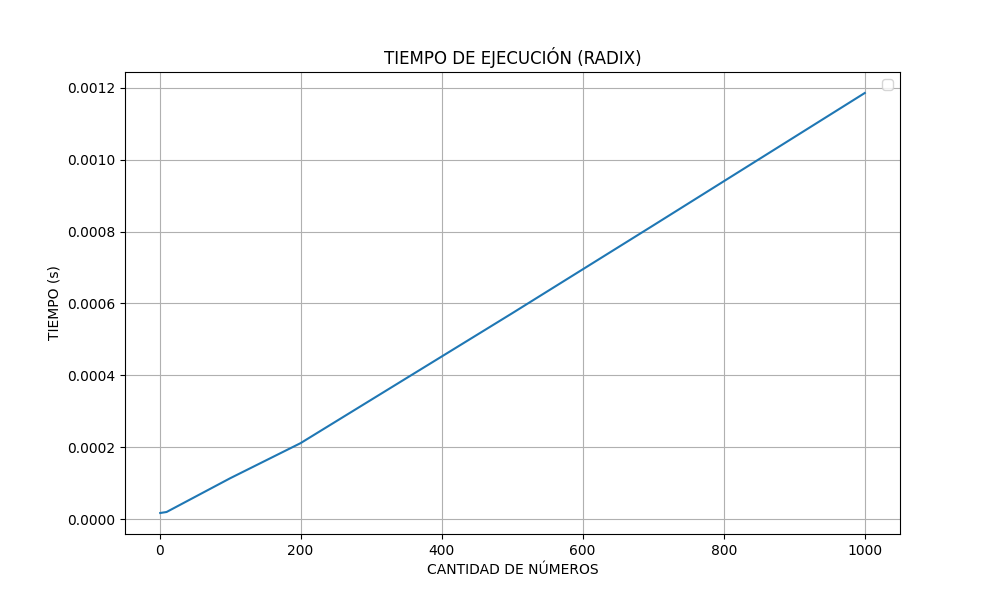
a) Ordenamiento Burbuja: dado que código tiene dos ciclos for anidados, concluimos que el orden de complejidad es . La gráfica realizada mediante código nos confirma que el orden de complejidad es



b) Ordenamiento Quicksort (rápido): realizando el mismo análisis que con el ordenamiento burbuja, podemos concluir que el orden de complejidad es , dado que tiene dos ciclos while anidados. La gráfica también nos confirma el análisis a priori.



c) Ordenamiento Radix (residuos): observando el código podemos determinar que el orden de complejidad depende de la cantidad de dígitos de los números de la lista y de la cantidad de números en la lista, por lo tanto, podemos concluir que el orden de complejidad es . La gráfica muestra un orden lineal de complejidad.

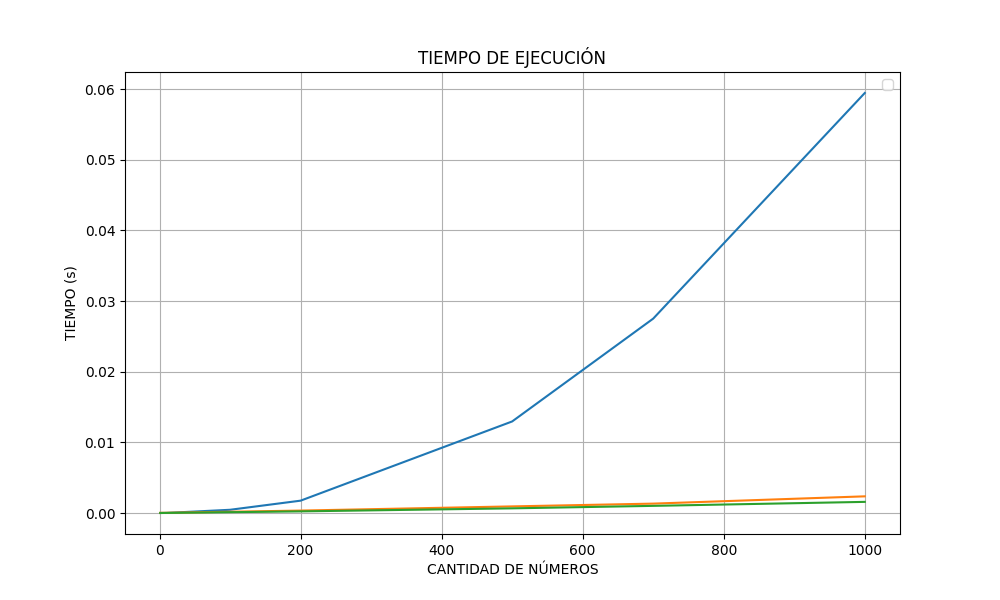


**GRÁFICA COMPARATIVA ENTRE LOS TRES TIPOS DE ORDENAMIENTOS**

Ordenamiento Burbuja.

Ordenamiento Quicksort.

Ordenamiento Radix.



**FUNCIÓN BUILT-IN SORTED**

La función sorted funciona creando una lista nueva en donde muestra los elementos de manera ordenada, no opera sobre la lista original. Teniendo las gráficas de los cuatro algoritmos de ordenamiento, podemos observar que sorted es mucho más rápido que los demás. Creemos que se debe a que es una función built-in dentro del lenguaje Python, lo que hace menor el tiempo de ejecución.

|  |  |
| --- | --- |
| ALGORITMOS | COMPLEJIDAD |
| sorted | O(n) |
| Burbuja | O(n2) |
| Radix | O(kn) |
| Quicksort | O(n2) |

