Introducción del Factor 1

En este apartado se analiza cómo influye la potencia del hardware en el rendimiento de un programa. Para ello, se ha ejecutado exactamente el mismo código fuente (PythonA1.py) en dos ordenadores distintos (PC 1 y PC 2) sometiéndolos a las mismas cargas de trabajo (mismos valores de n).

El objetivo es comparar los tiempos de ejecución resultantes y demostrar cómo las características técnicas de cada equipo (principalmente la CPU) afectan a la velocidad de resolución del problema.

Especificaciones de los equipos:

Ordenador 1 (PC_1 - Referencia):

- **Procesador:** 12th Gen Intel(R) Core(TM) i5-12400 @ 2.50 GHz
- **Memoria RAM:** 16,0 GB

Ordenador 2 (PC_2 - Alto Rendimiento):

- **Procesador:** AMD Ryzen 7 7800X3D 8-Core Processor @ 4.20 GHz
- **Memoria RAM:** 32,0 GB

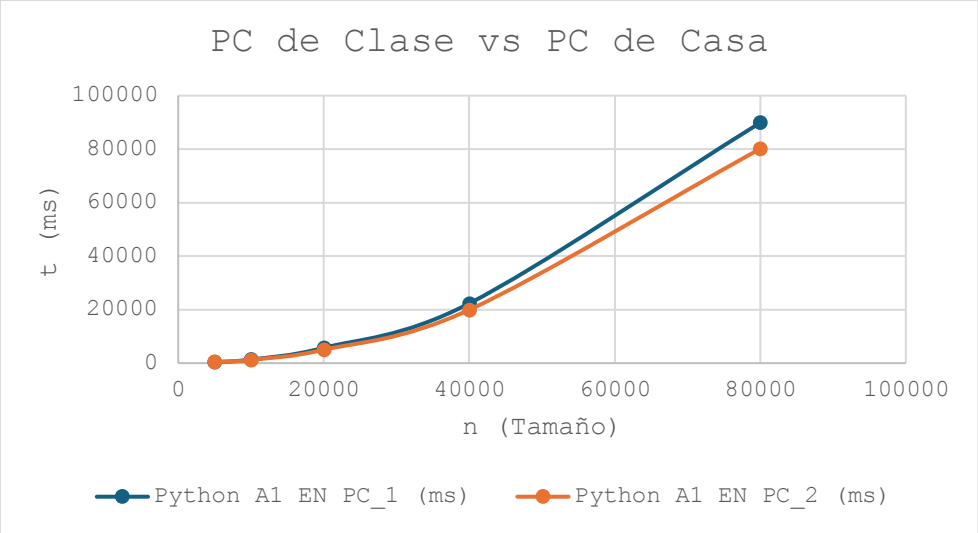
A priori, se espera que el PC_2 obtenga mejores tiempos debido a su mayor frecuencia base (4.20 GHz frente a 2.50 GHz) y a que se trata de un procesador de gama entusiasta frente a la gama media del i5.

Resultados Experimentales

A continuación, se presentan los tiempos obtenidos en milisegundos para ambos equipos:

n (Tamaño)	Python A1 EN PC_1 (ms)	Python A1 EN PC_2 (ms)
5000	335	334
10000	1347	1242
20000	5738	4999
40000	22318	19928
80000	89896	80168

A continuación, se adjunta también una gráfica comparativa:



Conclusión del Factor 1

Se demuestra que un hardware más potente (mayor frecuencia de CPU y mejor arquitectura) reduce el tiempo de ejecución de manera constante. Sin embargo, dado que el Algoritmo A1 es ineficiente por naturaleza (complejidad cuadrática).

Introducción del Factor 2

En este apartado se aborda el factor más determinante en el rendimiento: el diseño del algoritmo. Mientras que el hardware o el lenguaje pueden mejorar el tiempo en un factor constante (x2, x10), un buen algoritmo puede reducir la complejidad teórica del problema, logrando mejoras de varios órdenes de magnitud.

Se han implementado y medido tres evoluciones sobre el algoritmo base:

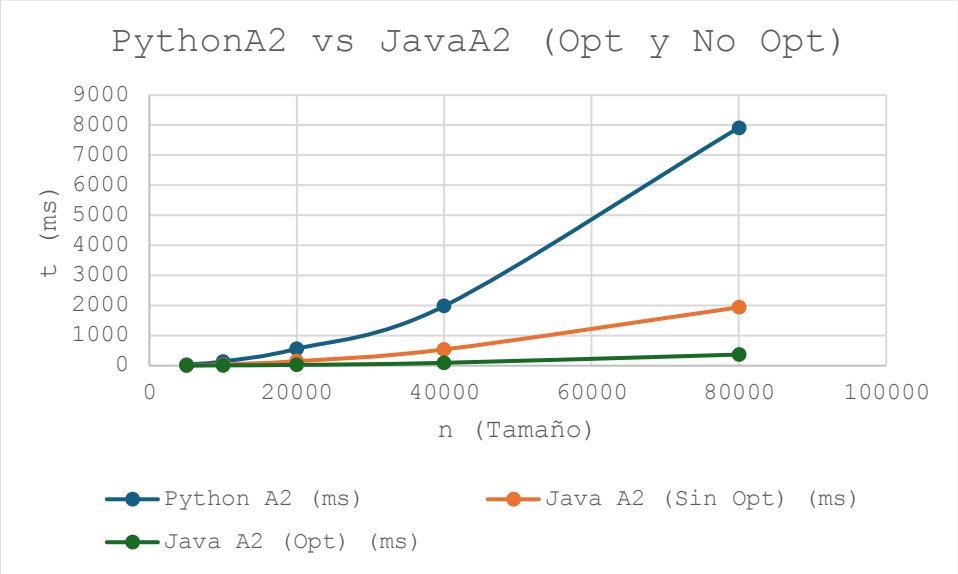
- **Algoritmo A2 (Rotura de bucle):** Mejora respecto a A1 deteniendo la búsqueda en cuanto se encuentra el primer divisor.
- **Algoritmo A3 (Raíz Cuadrada):** Optimización matemática que reduce el espacio de búsqueda. Solo se comprueban divisores hasta $\sqrt{n}$ y se omiten los números pares.
- **Algoritmo A4 (Criba de Eratóstenes):** Cambio de paradigma. En lugar de divisiones por tentativa, utiliza un array de booleanos para "tachar" múltiplos. Cambia tiempo de CPU por consumo de memoria.

Resultados Experimentales

La siguiente tabla muestra los tiempos obtenidos para diferentes tamaños de n tanto en Python como en Java (con y sin optimización JIT):

n (Tamaño)	Python A2 (ms)	Java A2 (Sin Opt) (ms)	Java A2 (Opt) (ms)
5000	37	10	3
10000	141	40	10
20000	561	150	25
40000	1983	540	95
80000	7920	1940	370

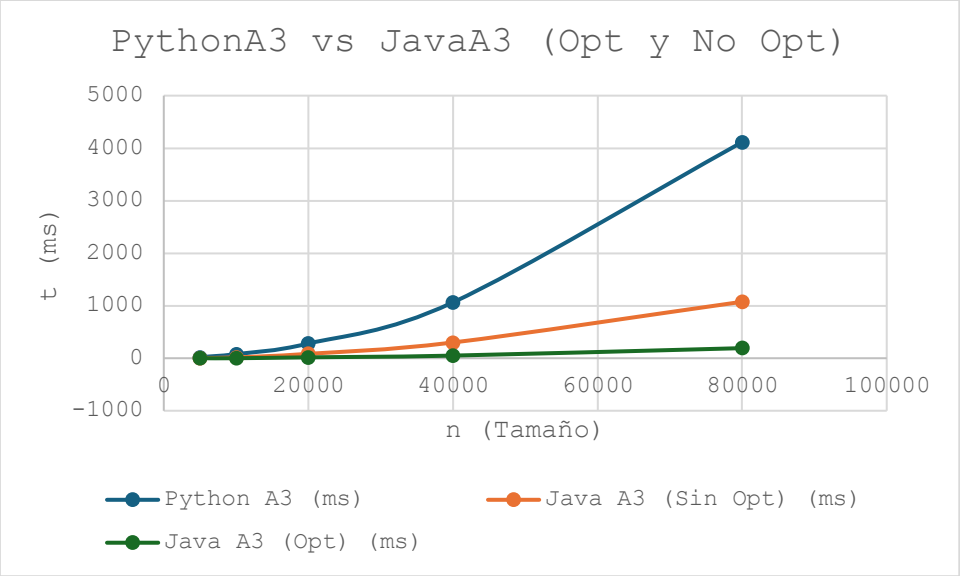
Con su respectiva gráfica:



Ahora el algoritmo A3:

Python A3 (ms)	Java A3 (Sin Opt) (ms)	Java A3 (Opt) (ms)
20	6	0
79	16	0
280	84	17
1064	300	50
4115	1075	192

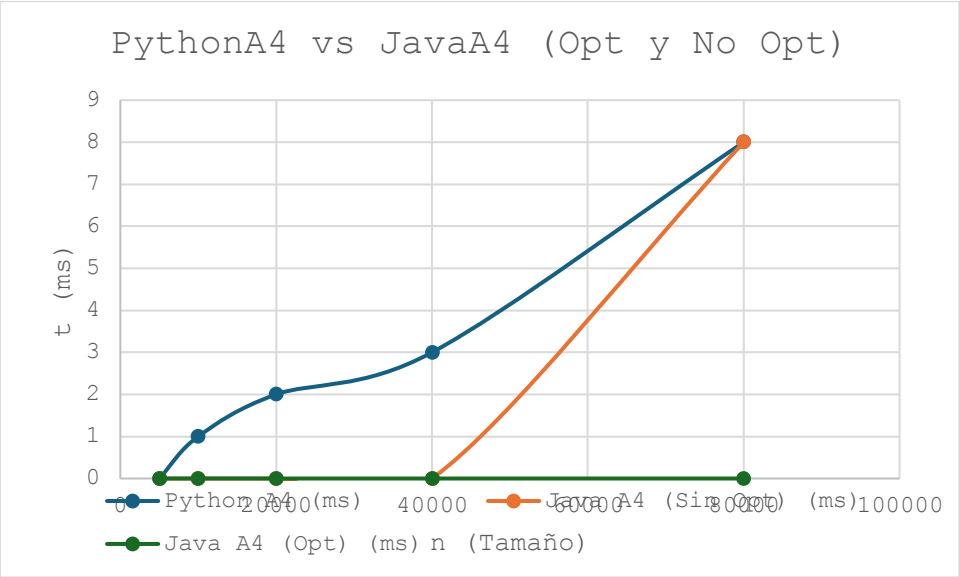
Con su gráfica:



Y, por último, el algoritmo A4 y el más eficiente:

Python A4 (ms)	Java A4 (Sin Opt) (ms)	Java A4 (Opt) (ms)
0	0	0
1	0	0
2	0	0
3	0	0
8	8	0

Con su gráfica:



Diferencia entre Java y Python

Los resultados demuestran claramente que Java es mucho más rápido que Python para este tipo de cálculos. La razón es sencilla:

- **Python (Interpretado):** Funciona como un traductor simultáneo. Lee el código línea por línea y lo traduce al ordenador mientras se ejecuta. Esto hace que pierda tiempo en cada vuelta del bucle "pensando" qué tiene que hacer.
- **Java (Compilado/JIT):** Funciona como si leyera un libro ya traducido. Gracias a su compilador (JIT), transforma el código a un lenguaje que el procesador entiende directamente. Una vez que "aprende" lo que hace el bucle, lo ejecuta a toda velocidad sin detenerse a traducir.

Conclusión Final

Este experimento valida la premisa fundamental de la asignatura: **El diseño del algoritmo es el factor dominante.**

Mientras que cambiar del PC 1 al PC 2 (Factor 2) mejoraba el tiempo en un 10%, cambiar del Algoritmo A1 al Algoritmo A4 reduce el tiempo en un factor de 10.000 (de casi 90 segundos a 8 milisegundos en Python). Ninguna mejora de hardware puede competir con una reducción en la complejidad algorítmica.