

# Algoritmia

## Práctica 0

8/2/2026

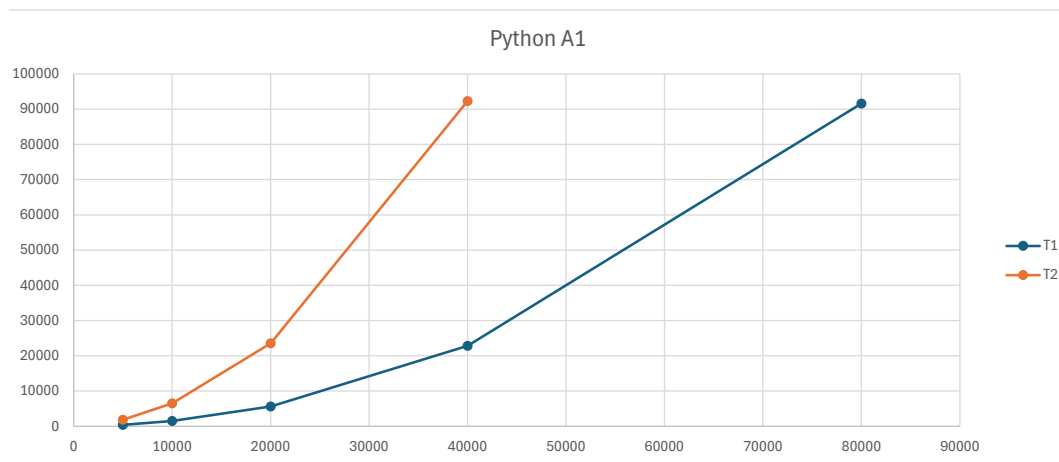
### Contenidos

1.	Toma de tiempos Python A1 en dos ordenadores distintos . . . . .	2
2.	Tiempos en Java con y sin optimización para A2 . . . . .	3
3.	Tiempos en Java con y sin optimización para A3 . . . . .	4
4.	Tiempos en Java con y sin optimización para A4 . . . . .	5
5.	Comparación de tiempos con optimización . . . . .	6
6.	Comparación entre algoritmos usando -Xint . . . . .	7
7.	Comparativa Java Python . . . . .	8
8.	Algoritmo A5: Criba de Atkin . . . . .	10

## 1. Toma de tiempos Python A1 en dos ordenadores distintos

$n$	$T_1$ (ms)	$T_2$ (ms)	Cociente $T_1/T_2$
5.000	387	1.838	0,210554951
10.000	1.464	6.496	0,225369458
20.000	5.619	23.515	0,238953859
40.000	22.840	92.309	0,247429828
80.000	91.608	FdT	-
160.000	FdT	FdT	-

El cociente de los tiempos se mantiene aun habiendo incrementado  $n$  mucho, en línea con el principio de invarianza visto en teoría.



### Comparativa gráfica entre ambos ordenadores

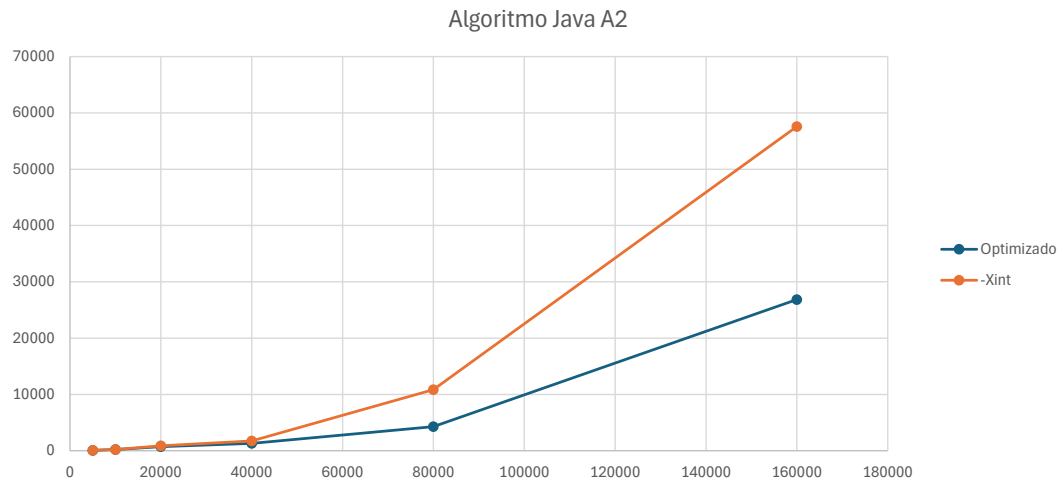
Los tiempos mostrados se corresponden con las siguientes configuraciones de hardware:

- $T_1$ : CPU 12th Gen Intel(R) Core(TM) i5-12400 @ 2,5GHz + 16GB RAM
- $T_2$ : CPU: Intel Core i7-6700HQ CPU @ 2,60GHz + 16GB RAM

El resto de tomas de tiempos de esta memoria se realizarán con la configuración “ $T_2$ ” de este apartado.

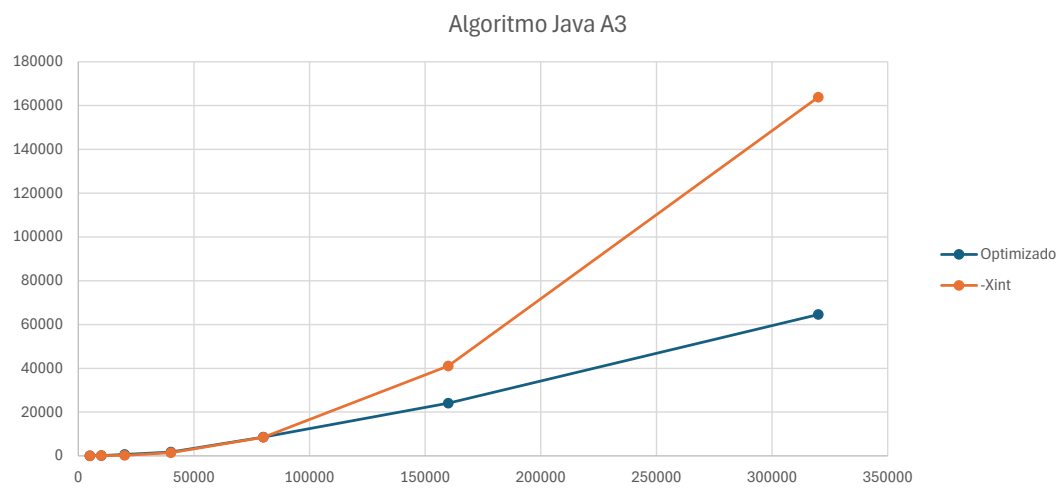
## 2. Tiempos en Java con y sin optimización para A2

$n$	$T_{Optim}(ms)$	$T_{Xint}(ms)$
5.000	66	68
10.000	219	221
20.000	698	886
40.000	1.319	1.719
80.000	4.267	10.860
160.000	26.851	57.587
320.000	FdT	FdT



### 3. Tiempos en Java con y sin optimización para A3

$n$	$T_{Optim}(ms)$	$T_{Xint}(ms)$
5.000	48	50
10.000	113	169
20.000	663	212
40.000	1.734	1.406
80.000	8.588	8.489
160.000	24.114	41.085
320.000	64.550	163.845
640.000	FdT	FdT



#### 4. Tiempos en Java con y sin optimización para A4

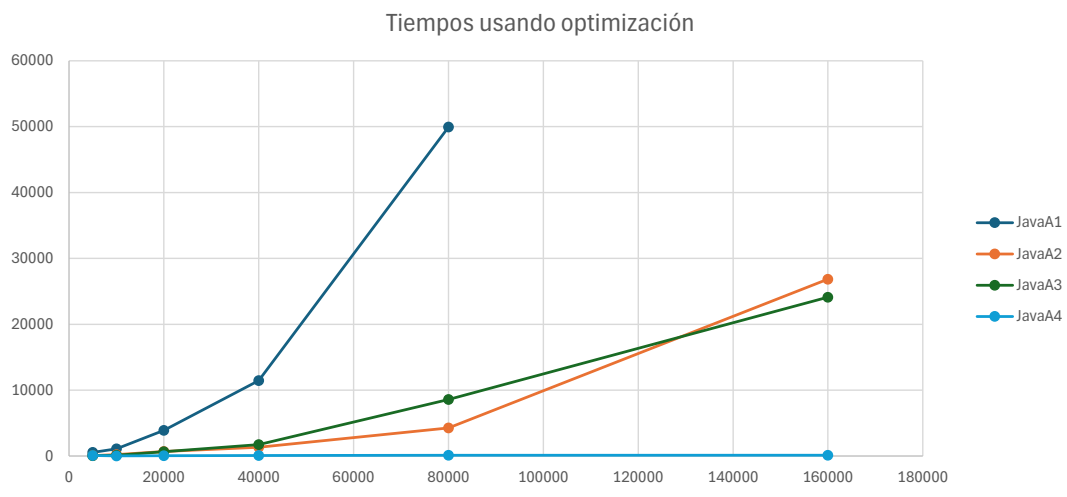
$n$	$T_{Optim}(ms)$	$T_{Xint}(ms)$
5.000	43	109
10.000	1	1
20.000	1	1
40.000	4	4
80.000	12	11
160.000	38	28
320.000	75	55
640.000	147	128
1.280.000	292	146
2.560.000	448	321
5.120.000	1.064	903
10.240.000	2.216	1.914
20.480.000	3.113	3.934
40.960.000	8.503	6.835
81.920.000	14.329	15.591
163.840.000	31.754	30.892

La optimización no aporta cambios tan grandes o incluso empeora el rendimiento



## 5. Comparación de tiempos con optimización

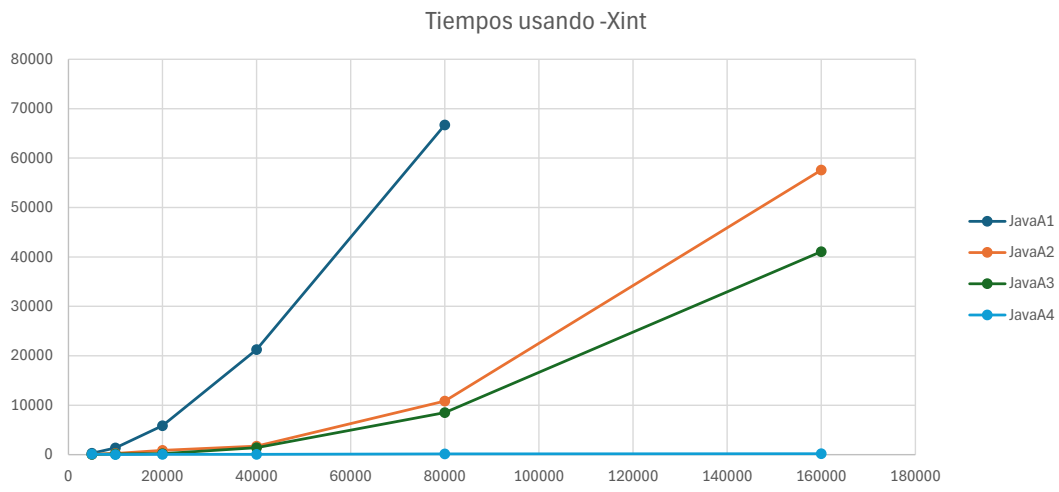
$n$	$JavaA1$ (ms)	$JavaA2$ (ms)	$JavaA3$ (ms)	$JavaA4$ (ms)
5.000	543	66	48	43
10.000	1.083	219	113	1
20.000	3.895	698	663	1
40.000	11.454	1.319	1.734	4
80.000	49.932	4.267	8.588	12
160.000	FdT	26.851	24.114	38
320.000	FdT	FdT	64.550	75
640.000	FdT	FdT	FdT	147
1.280.000	FdT	FdT	FdT	292
2.560.000	FdT	FdT	FdT	448
5.120.000	FdT	FdT	FdT	1.064
10.240.000	FdT	FdT	FdT	2.216
20.480.000	FdT	FdT	FdT	3.113
40.960.000	FdT	FdT	FdT	8.503
81.920.000	FdT	FdT	FdT	14.329
163.840.000	FdT	FdT	FdT	31.754



Se han omitido datos de JavaA4 para evitar la distorsión de la gráfica

## 6. Comparación entre algoritmos usando -Xint

$n$	<i>JavaA1</i> (ms)	<i>JavaA2</i> (ms)	<i>JavaA3</i> (ms)	<i>JavaA4</i> (ms)
5.000	278	68	50	109
10.000	1.364	221	169	1
20.000	5.860	886	212	1
40.000	21.258	1.719	1.406	4
80.000	66.728	10.860	8.489	11
160.000	FdT	57.587	41.085	28
320.000	FdT	FdT	FdT	55
640.000	FdT	FdT	FdT	128
1.280.000	FdT	FdT	FdT	146
2.560.000	FdT	FdT	FdT	321
5.120.000	FdT	FdT	FdT	903
10.240.000	FdT	FdT	FdT	1.914
20.480.000	FdT	FdT	FdT	3.934
40.960.000	FdT	FdT	FdT	6.835
81.920.000	FdT	FdT	FdT	15.591
163.840.000	FdT	FdT	FdT	30.892



Se han omitido datos de JavaA4 para evitar la distorsión de la gráfica

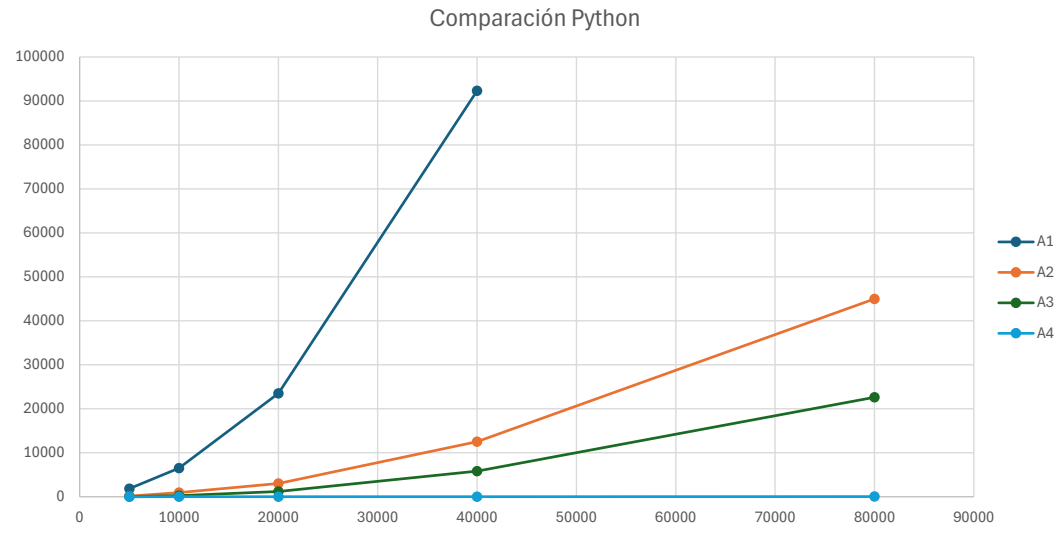
## 7. Comparativa Java Python

$n$	<i>JavaA1 (ms)</i>	<i>PythonA1 (ms)</i>
5.000	278	1.838
10.000	1.364	6.496
20.000	5.860	23.515
40.000	21.258	92.309
80.000	66.728	FdT
160.000	FdT	FdT

$n$	<i>JavaA2 (ms)</i>	<i>PythonA2 (ms)</i>
5.000	68	148
10.000	221	939
20.000	886	2.994
40.000	1.719	12.533
80.000	10.860	44.993
160.000	57.587	FdT

$n$	<i>JavaA3 (ms)</i>	<i>PythonA3 (ms)</i>
5.000	50	68
10.000	169	281
20.000	212	1.188
40.000	1.406	5.828
80.000	8.489	22.621
160.000	41.085	80.348
320.000	FdT	FdT

$n$	$JavaA_4$ (ms)	$PythonA_4$ (ms)
5.000	109	1
10.000	1	8
20.000	1	5
40.000	4	24
80.000	11	39
160.000	28	72
320.000	55	182
640.000	128	413
1.280.000	146	645
2.560.000	321	1.416
5.120.000	903	3.468
10.240.000	1.914	6.657
20.480.000	3.934	13.347
40.960.000	6.835	30.003
81.920.000	15.591	58.415
163.840.000	30.892	FdT



Se han omitido datos de A3 y A4 para no distorsionar el eje de abscisas

## 8. Algoritmo A5: Criba de Atkin

$n$	<i>Python (ms)</i>	<i>-Xint (ms)</i>	<i>Cociente Py/J</i>
5.000	3	99	0,0303
10.000	6	5	1,2000
20.000	13	10	1,3000
40.000	34	18	1,8889
80.000	82	36	2,2778
160.000	153	72	2,1250
320.000	442	140	3,1571
640.000	799	193	4,1399
1.280.000	1.552	403	3,8511
2.560.000	3.544	894	3,9642
5.120.000	5.938	1.891	3,1401
10.240.000	13.359	3.736	3,5757
20.480.000	27.652	7.588	3,6442
40.960.000	53.071	15.416	3,4426
81.920.000	FdT	27.186	-
163.840.000	FdT	53.810	-
327.680.000	FdT	FdT	-

Comparativa de tiempos entre Python y Java con *-Xint*. Podemos observar que los cocientes tienden a una constante no nula, en línea con el principio de invarianza visto en teoría.

Puesto que el algoritmo A4 ya produce tiempos mucho menores que el resto, se ha optado por no mostrar A5 con los demás algoritmos, pues su grafo coincidiría estéticamente con el de A4, provocando ruido en la figura.

Aun teniendo Atkin una complejidad teórica menor, el tamaño de carga  $n$  no es lo suficientemente grande como para superar en rendimiento a la criba de Eratóstenes en la práctica.