Algorítmica: práctica 1

Análisis de la eficiencia de algoritmos

Sofía Almeida Bruno
Antonio Coín Castro
María Victoria Granados Pozo
Miguel Lentisco Ballesteros
José María Martín Luque

23 de marzo de 2017

Objetivo

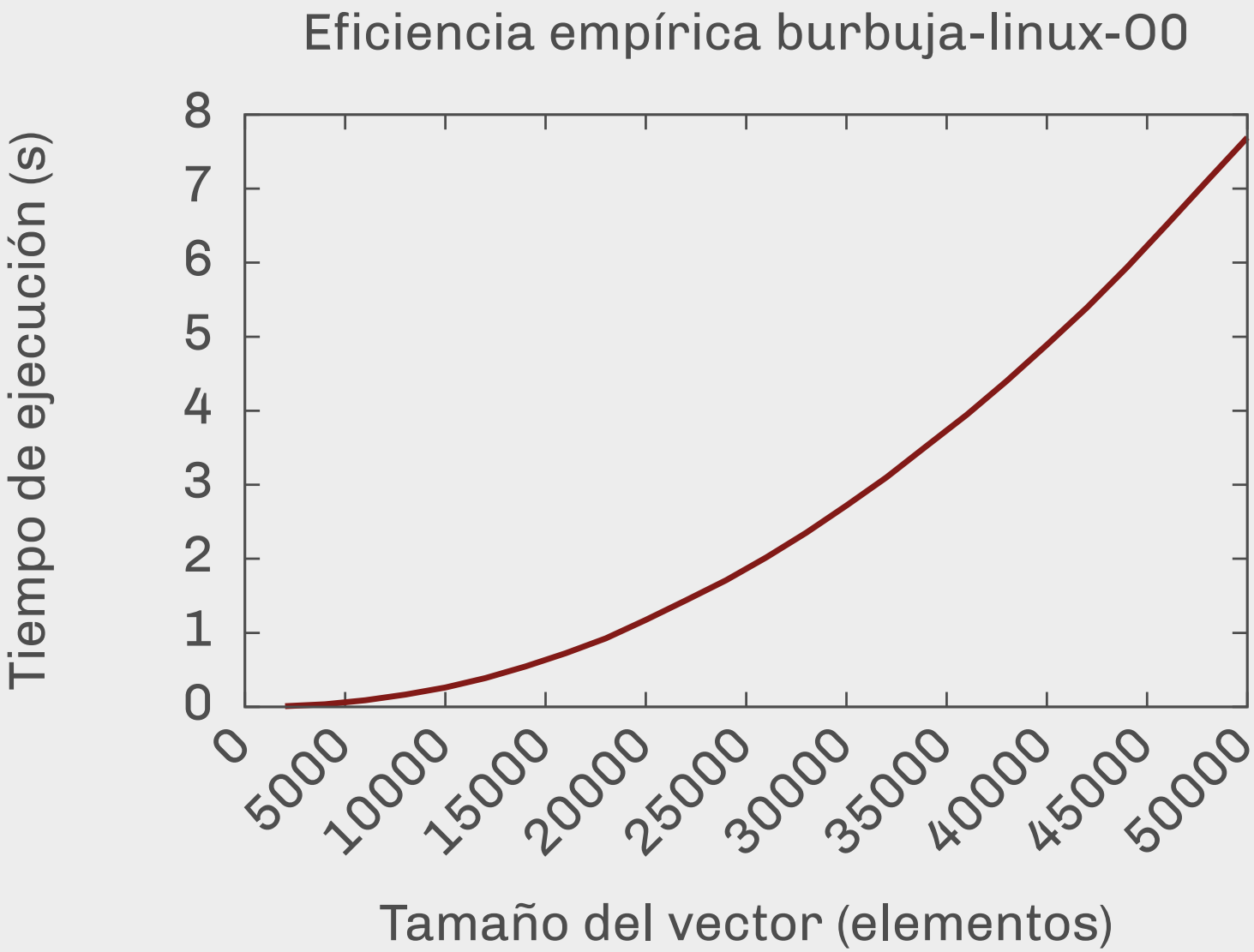
Estudiar la eficiencia teórica, empírica e híbrida de 8 algoritmos. También, realizar comparaciones entre sus tiempos de ejecución, así como la influencia de otros factores, como S.O. utilizado o prestaciones del PC.

- Burbuja, Inserción y Selección
- Mergesort, Quicksort y Heapsort
- Floyd
- Hanoi

Burbuja

Revisa cada elemento de la lista con el siguiente, intercambiándose de posición si no están en el orden correcto.

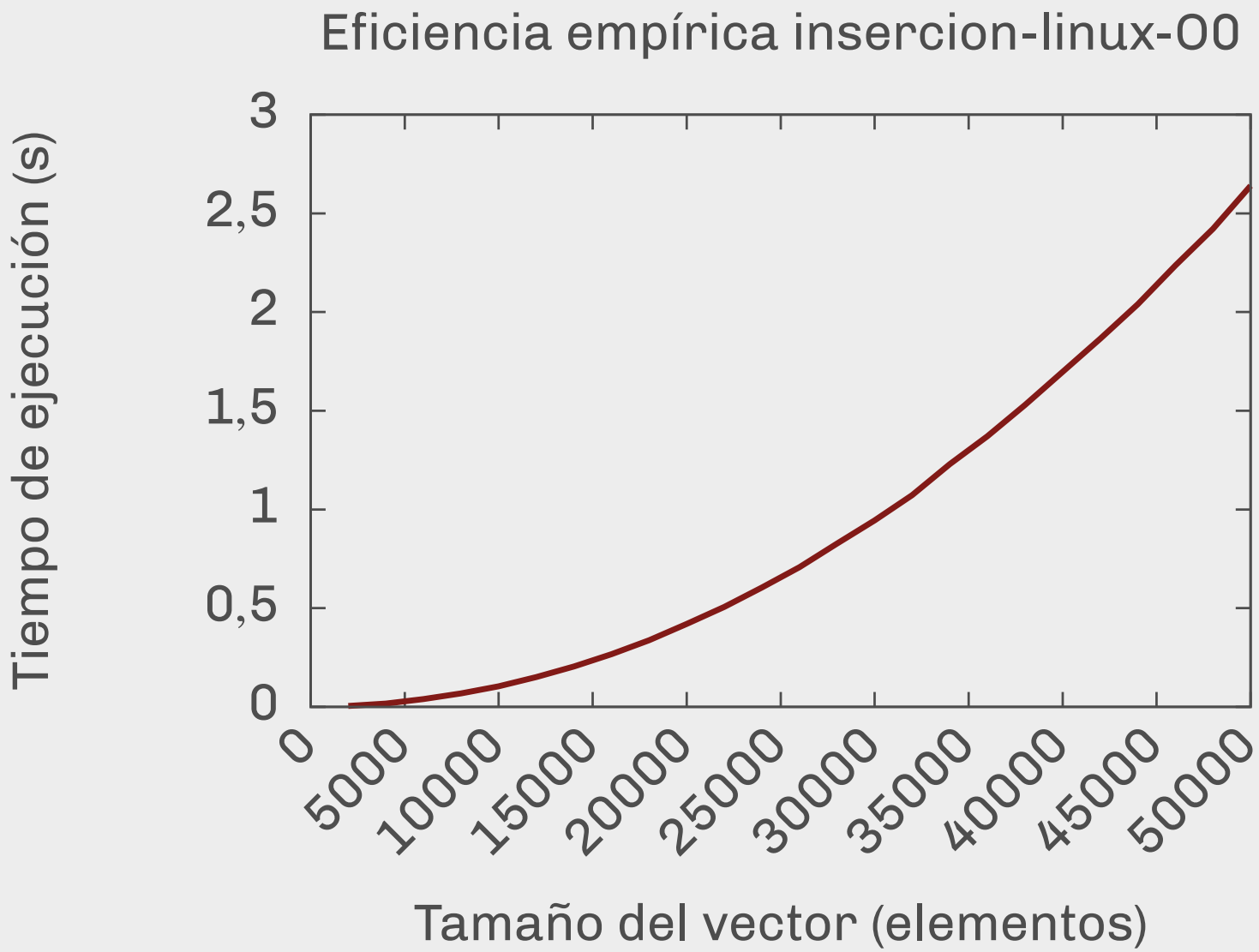
Es $O(n^2)$.


Eficiencia empírica burbuja-linux-O0
Tiempo de ejecución (s)
0
1
2
3
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
Tamaño del vector (elementos)

Inserción

Consideramos el elemento N-ésimo de la lista y lo ordenamos respecto de los elementos desde el primero hasta el N-1-ésimo.

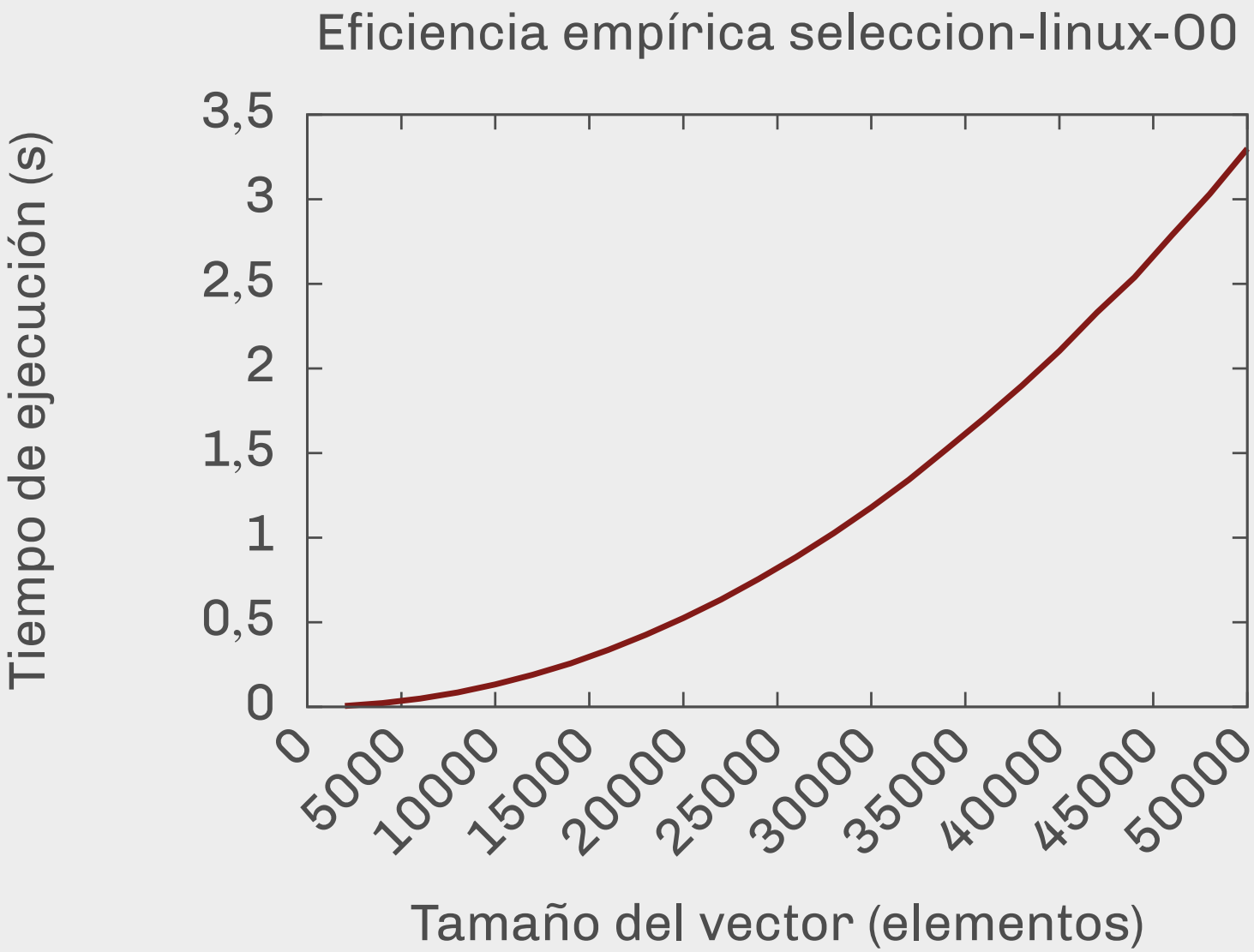
Es $O(n^2)$.


Eficiencia empírica insercion-linux-O0
Tiempo de ejecución (s)
3
2,5
2
1,5
1
0,5
0
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
Tamaño del vector (elementos)

Selección

Consiste en encontrar el menor de todos los elementos de la lista e intercambiarlo con el de la primera posición. Luego con el segundo, y así sucesivamente hasta ordenarlo todo.

Es $O(n^2)$.

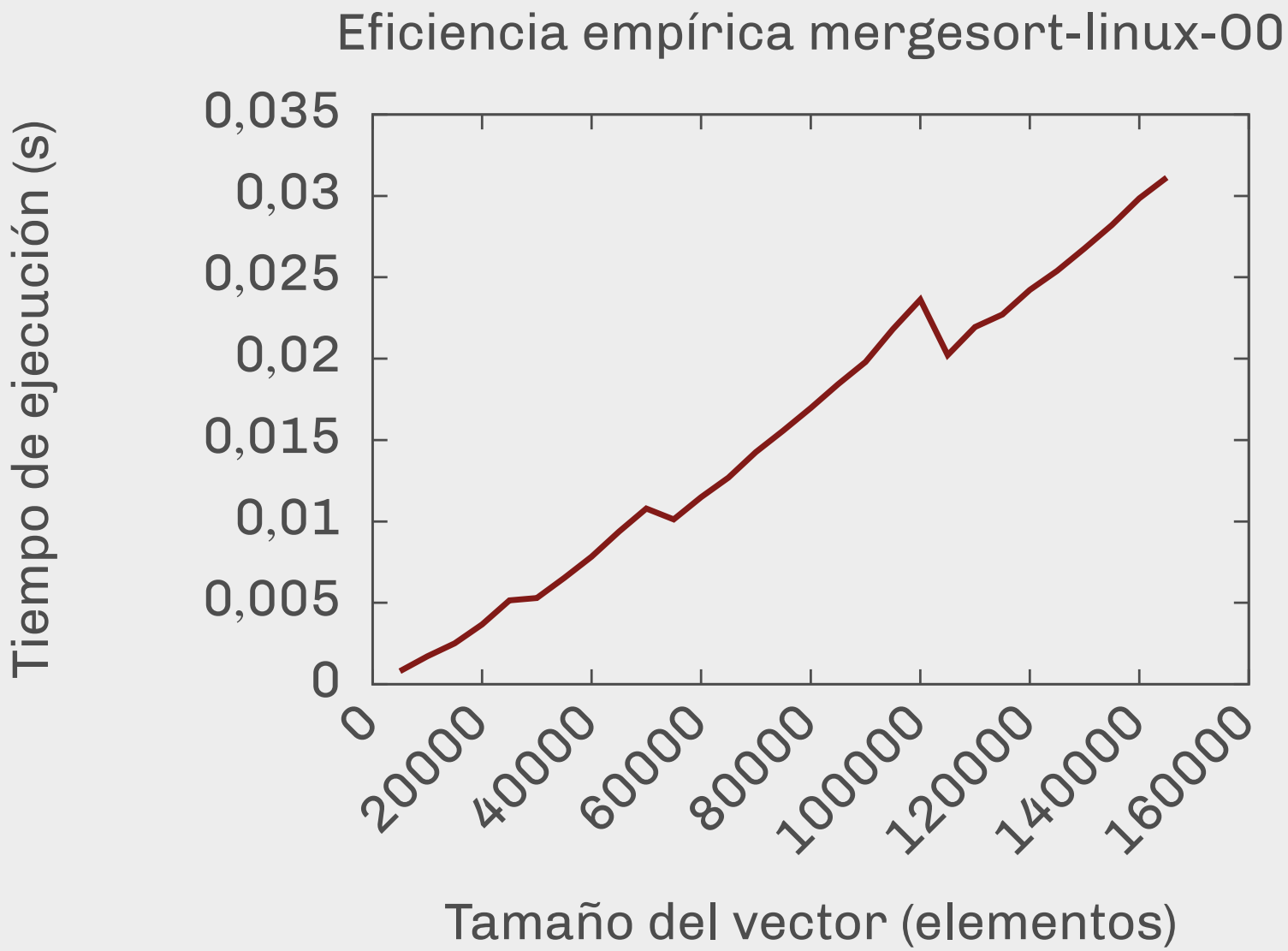

Eficiencia empírica seleccion-linux-O0
Tiempo de ejecución (s)
3,5
3
2,5
2
1,5
1
0,5
0
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000
Tamaño del vector (elementos)

Mergesort

Se basa en la técnica de *divide y vencerás*.

- Se divide la lista a ordenar en dos sublistas de la mitad de tamaño.
- Se ordena cada sublista de forma recursiva.
- Si el tamaño de una sublista es 0 o 1 entonces ya está ordenada.
- Se unen todas las sublistas en una sola.

Es $O(nlogn)$.

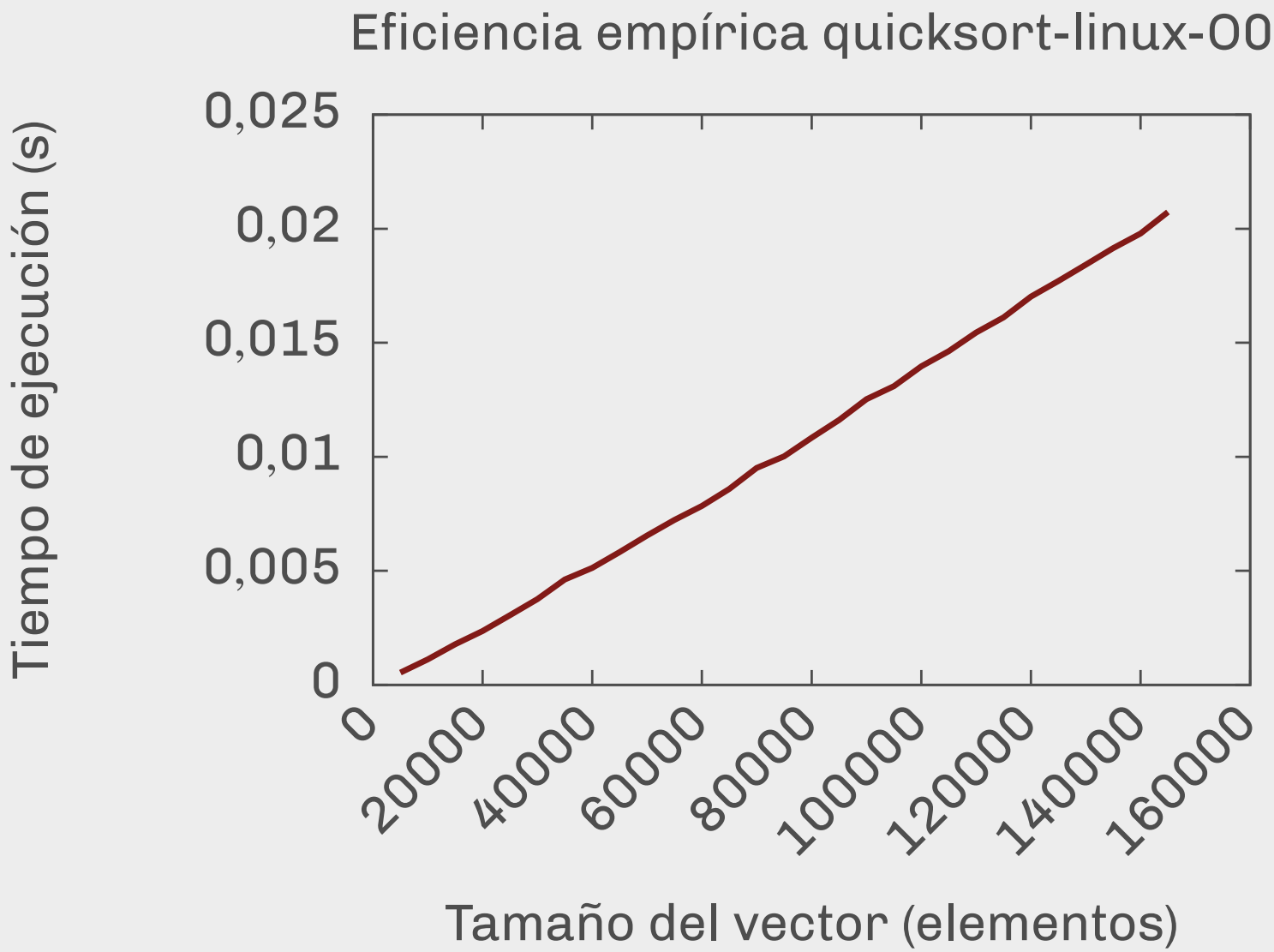

Eficiencia empírica mergesort-linux-O0
Tiempo de ejecución (s)
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
0
20000
40000
60000
80000
100000
120000
140000
160000
Tamaño del vector (elementos)

Quicksort

Se basa en la técnica de *divide y vencerás*.

- Elegimos un elemento de la lista, el *pivote*.
- Se ordena la lista, dejando los elementos mayores a la derecha del pivote y los menores a la izquierda.
- Realizamos el proceso recursivamente en las dos sublistas que nos quedan (derecha e izquierda) hasta que tengan 0 o 1 elemento.

Es $O(nlogn)$.

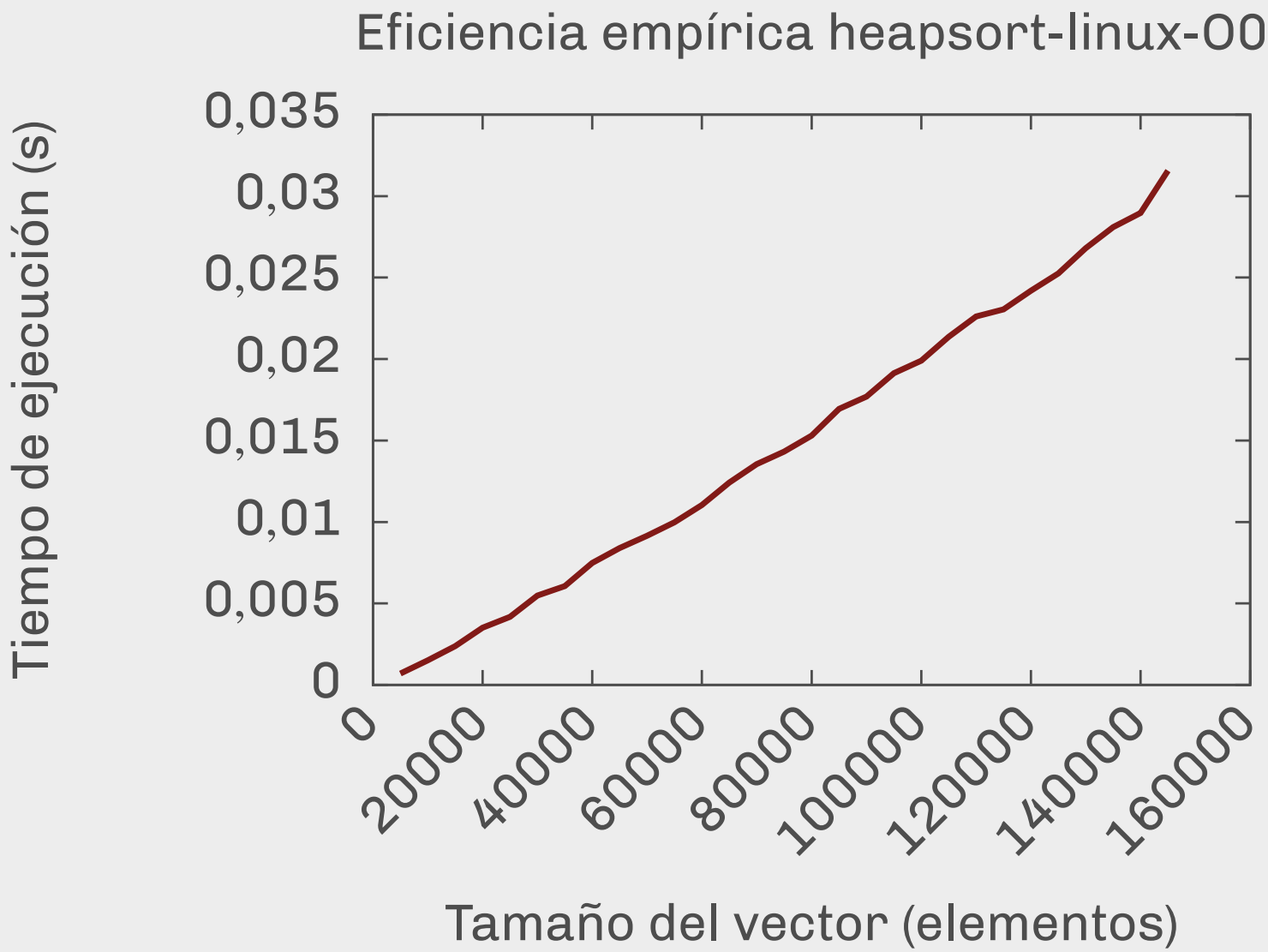

Eficiencia empírica quicksort-linux-O0
Tiempo de ejecución (s)
0,025
0,02
0,015
0,01
0,005
0
0
20000
40000
60000
80000
100000
120000
140000
160000
Tamaño del vector (elementos)

Heapsort

Es un método de ordenación por selección.

- El *heap* es un árbol binario de altura mínima, en el que los nodos del nivel más bajo están lo más a la izquierda posible.
- Los hijos de cada nodo son siempre menores que el padre.
- No es necesario recorrer el árbol de forma desordenada para encontrar los elementos máximos.

Es $O(nlogn)$.

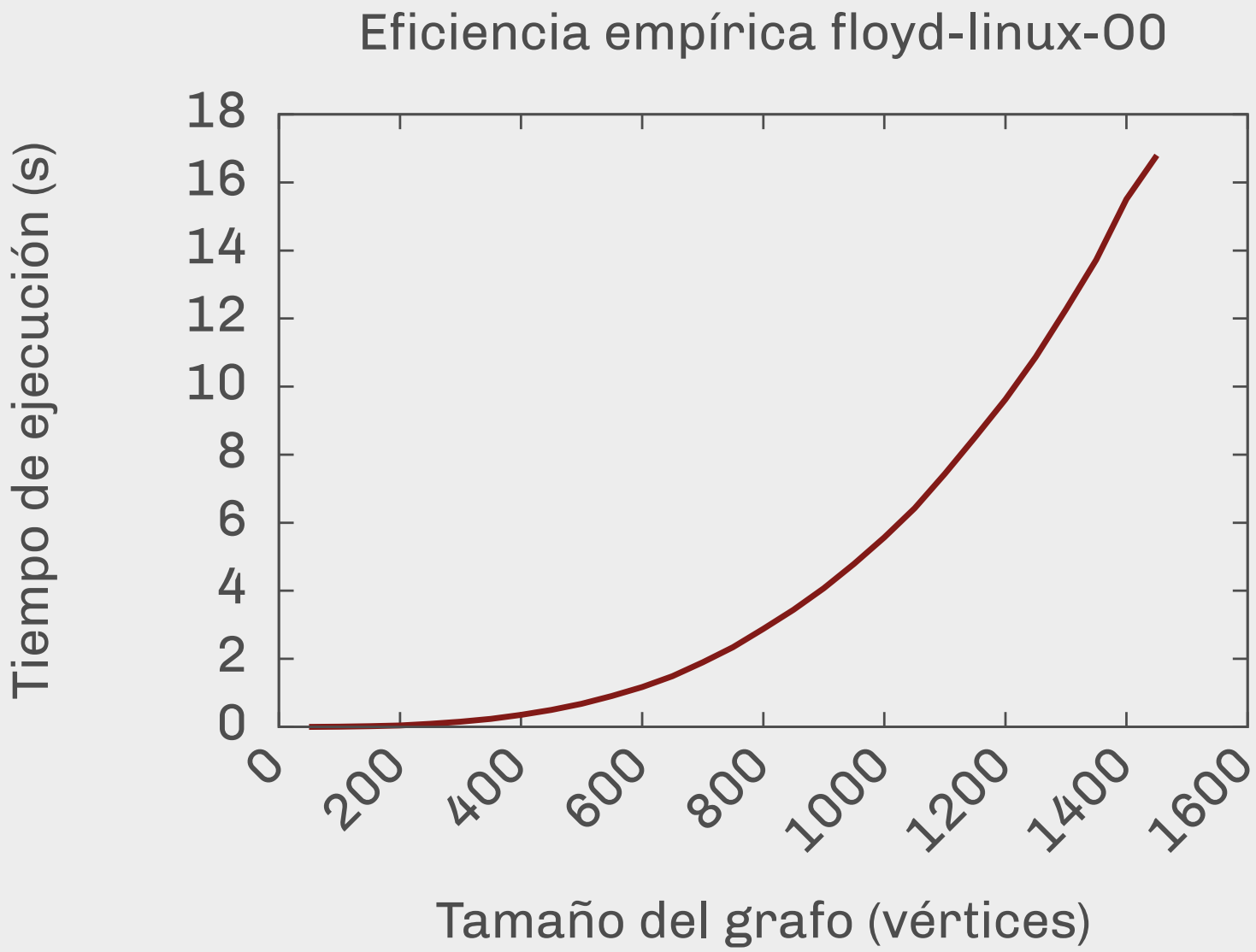

Eficiencia empírica heapsort-linux-O0
Tiempo de ejecución (s)
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
0
20000
40000
60000
80000
100000
120000
140000
160000
Tamaño del vector (elementos)

Floyd

Algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos ponderados.

- El algoritmo compara todos los posibles caminos a través del grafo entre cada par de vértices.

Es $O(n^3)$.

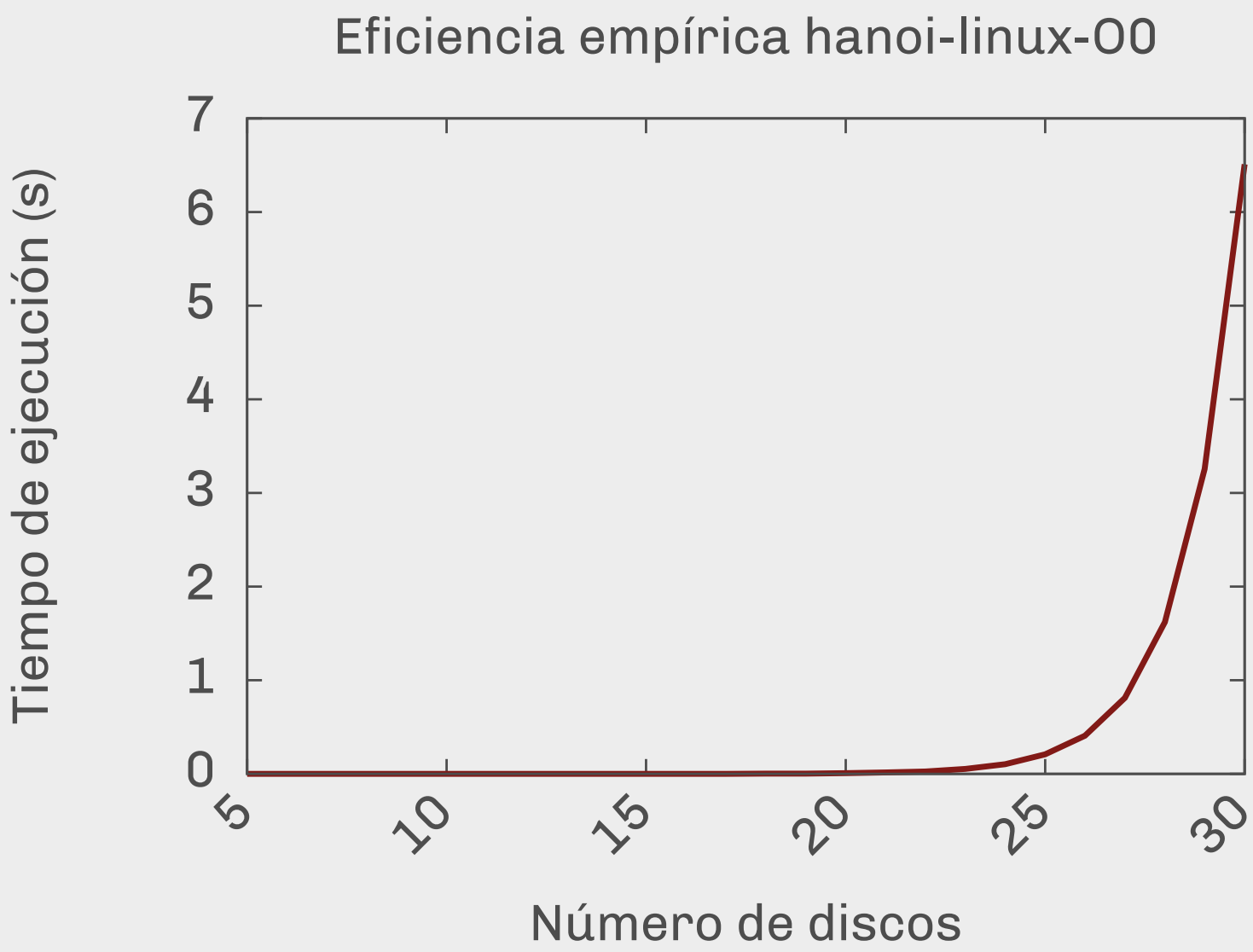

Eficiencia empírica floyd-linux-O0
Tiempo de ejecución (s)
0
2
4
6
8
10
12
14
16
18
0
200
400
600
800
1000
1200
1400
1600
Tamaño del grafo (vértices)

Hanoi

Las torres de Hanoi son un puzzle matemático.

- Tenemos 3 pilas: origen, auxiliar y destino.
- Origen está ordenada por tamaño creciente de discos.
- Se mueve un disco de la pila origen a la de destino si hay un único disco en la pila origen.
- Si no, se mueven todos los discos a la auxiliar, excepto el más grande.
- Por último, movemos el disco mayor al destino, y los $n - 1$ restantes encima del mayor.

Es $O(2^n)$.


Eficiencia empírica hanoi-linux-O0
Tiempo de ejecución (s)
0
1
2
3
4
5
6
7
5
10
15
20
25
30
Número de discos

Datos de la eficiencia empírica

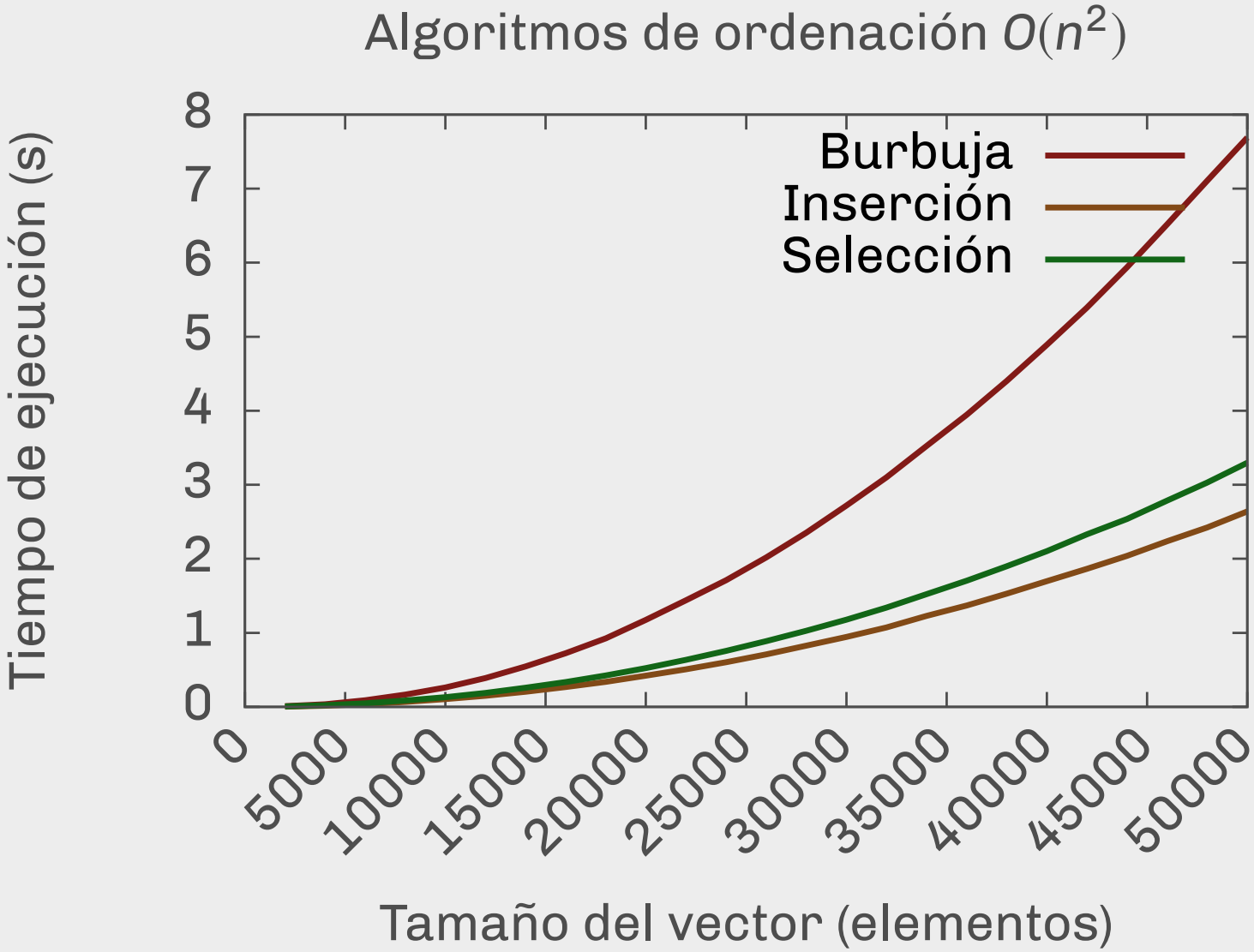
Hemos recopilado los datos de la eficiencia empírica de la ejecución de los distintos algoritmos en varias tablas comparativas. Los datos han sido obtenidos ejecutandose en la misma máquina con Linux y sin optimización.

Hemos creado tablas para los distintos órdenes de eficiencia de los algoritmos y hemos puesto juntos aquellos que tienen el mismo.

Finalmente, para cada tabla comparativa hemos creado una gráfica.

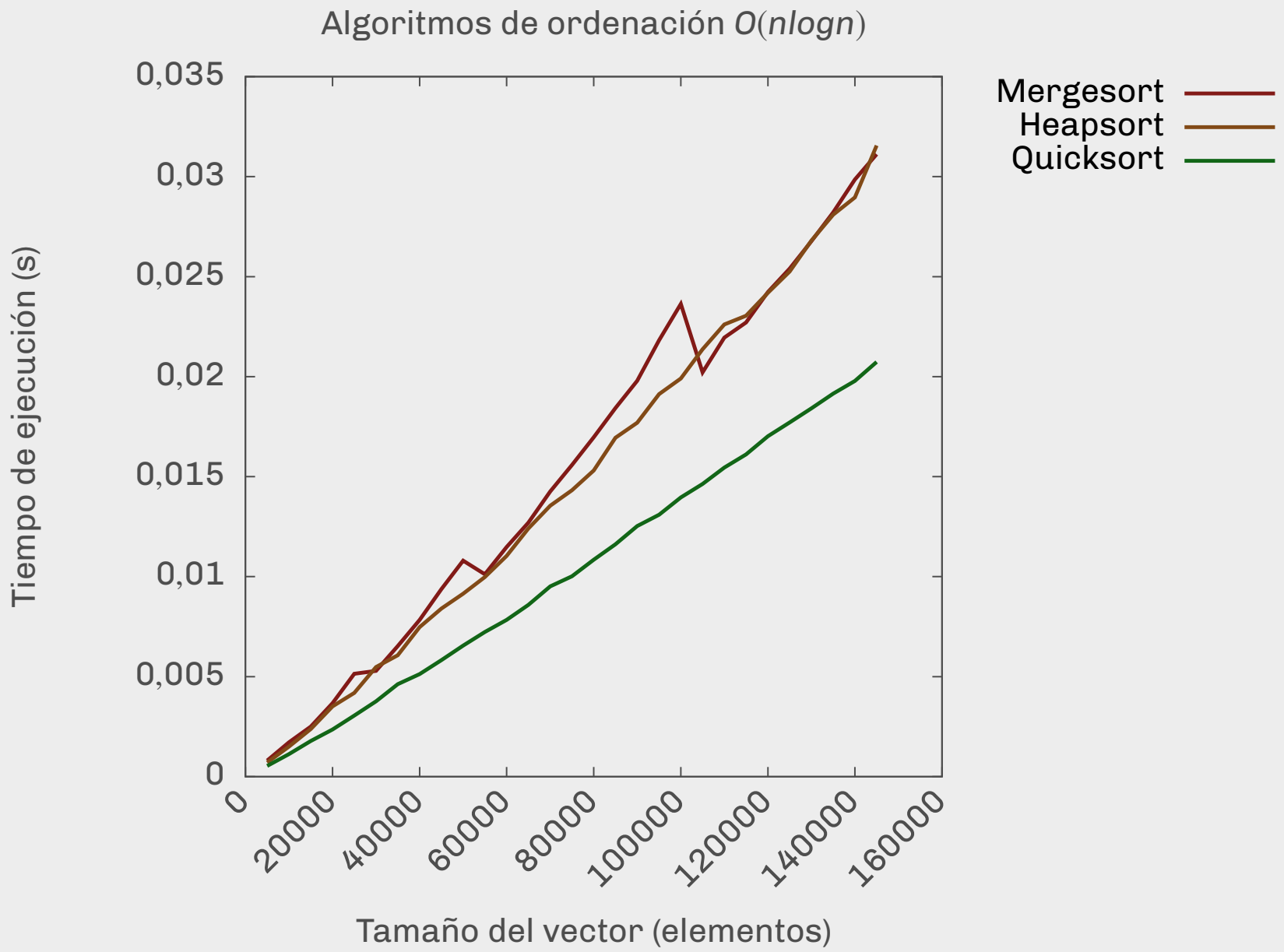
Elementos	Burbuja	Inserción	Selección
2,000	$8{,}02 \cdot 10^{-3}$	$4{,}41 \cdot 10^{-3}$	$5{,}4 \cdot 10^{-3}$
4,000	$3{,}5 \cdot 10^{-2}$	$1{,}73 \cdot 10^{-2}$	$2{,}17 \cdot 10^{-2}$
6,000	$8{,}93 \cdot 10^{-2}$	$3{,}95 \cdot 10^{-2}$	$4{,}84 \cdot 10^{-2}$
8,000	0,16	$6{,}77 \cdot 10^{-2}$	$8{,}52 \cdot 10^{-2}$
10,000	0,26	0,1	0,13
12,000	0,39	0,15	0,19
14,000	0,55	0,2	0,26
16,000	0,73	0,27	0,34
18,000	0,93	0,34	0,43
20,000	1,18	0,42	0,52
22,000	1,44	0,51	0,63
24,000	1,71	0,6	0,76
26,000	2,02	0,71	0,89
28,000	2,35	0,83	1,03
30,000	2,72	0,94	1,18
32,000	3,1	1,07	1,34
34,000	3,53	1,23	1,52
36,000	3,95	1,37	1,71
38,000	4,4	1,53	1,9
40,000	4,89	1,7	2,1
42,000	5,39	1,87	2,33
44,000	5,94	2,04	2,54
46,000	6,52	2,24	2,79
48,000	7,11	2,42	3,03
50,000	7,69	2,64	3,3

Algoritmos que son $O(n^2)$


Algoritmos de ordenación $O(n^2)$
Tiempo de ejecución (s)
Tamaño del vector (elementos)
Burbuja
Inserción
Selección
0
1
2
3
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000

Elementos	Mergesort	Quicksort	Heapsort
5,000	$8{,}02 \cdot 10^{-4}$	$5{,}34 \cdot 10^{-4}$	$7{,}01 \cdot 10^{-4}$
10,000	$1{,}72 \cdot 10^{-3}$	$1{,}12 \cdot 10^{-3}$	$1{,}5 \cdot 10^{-3}$
15,000	$2{,}5 \cdot 10^{-3}$	$1{,}79 \cdot 10^{-3}$	$2{,}38 \cdot 10^{-3}$
20,000	$3{,}68 \cdot 10^{-3}$	$2{,}36 \cdot 10^{-3}$	$3{,}51 \cdot 10^{-3}$
25,000	$5{,}15 \cdot 10^{-3}$	$3{,}05 \cdot 10^{-3}$	$4{,}18 \cdot 10^{-3}$
30,000	$5{,}29 \cdot 10^{-3}$	$3{,}76 \cdot 10^{-3}$	$5{,}48 \cdot 10^{-3}$
35,000	$6{,}53 \cdot 10^{-3}$	$4{,}63 \cdot 10^{-3}$	$6{,}07 \cdot 10^{-3}$
40,000	$7{,}83 \cdot 10^{-3}$	$5{,}13 \cdot 10^{-3}$	$7{,}48 \cdot 10^{-3}$
45,000	$9{,}37 \cdot 10^{-3}$	$5{,}82 \cdot 10^{-3}$	$8{,}4 \cdot 10^{-3}$
50,000	$1{,}08 \cdot 10^{-2}$	$6{,}55 \cdot 10^{-3}$	$9{,}15 \cdot 10^{-3}$
55,000	$1{,}01 \cdot 10^{-2}$	$7{,}23 \cdot 10^{-3}$	$9{,}98 \cdot 10^{-3}$
60,000	$1{,}15 \cdot 10^{-2}$	$7{,}84 \cdot 10^{-3}$	$1{,}1 \cdot 10^{-2}$
65,000	$1{,}27 \cdot 10^{-2}$	$8{,}58 \cdot 10^{-3}$	$1{,}24 \cdot 10^{-2}$
70,000	$1{,}43 \cdot 10^{-2}$	$9{,}52 \cdot 10^{-3}$	$1{,}35 \cdot 10^{-2}$
75,000	$1{,}56 \cdot 10^{-2}$	$1 \cdot 10^{-2}$	$1{,}43 \cdot 10^{-2}$
80,000	$1{,}7 \cdot 10^{-2}$	$1{,}08 \cdot 10^{-2}$	$1{,}53 \cdot 10^{-2}$
85,000	$1{,}84 \cdot 10^{-2}$	$1{,}16 \cdot 10^{-2}$	$1{,}69 \cdot 10^{-2}$
90,000	$1{,}98 \cdot 10^{-2}$	$1{,}25 \cdot 10^{-2}$	$1{,}77 \cdot 10^{-2}$
95,000	$2{,}18 \cdot 10^{-2}$	$1{,}31 \cdot 10^{-2}$	$1{,}91 \cdot 10^{-2}$
$1 \cdot 10^{5}$	$2{,}36 \cdot 10^{-2}$	$1{,}4 \cdot 10^{-2}$	$1{,}99 \cdot 10^{-2}$
$1{,}05 \cdot 10^{5}$	$2{,}02 \cdot 10^{-2}$	$1{,}46 \cdot 10^{-2}$	$2{,}14 \cdot 10^{-2}$
$1{,}1 \cdot 10^{5}$	$2{,}2 \cdot 10^{-2}$	$1{,}54 \cdot 10^{-2}$	$2{,}26 \cdot 10^{-2}$
$1{,}15 \cdot 10^{5}$	$2{,}27 \cdot 10^{-2}$	$1{,}61 \cdot 10^{-2}$	$2{,}3 \cdot 10^{-2}$
$1{,}2 \cdot 10^{5}$	$2{,}42 \cdot 10^{-2}$	$1{,}7 \cdot 10^{-2}$	$2{,}42 \cdot 10^{-2}$
$1{,}25 \cdot 10^{5}$	$2{,}54 \cdot 10^{-2}$	$1{,}77 \cdot 10^{-2}$	$2{,}53 \cdot 10^{-2}$
$1{,}3 \cdot 10^{5}$	$2{,}68 \cdot 10^{-2}$	$1{,}84 \cdot 10^{-2}$	$2{,}68 \cdot 10^{-2}$
$1{,}35 \cdot 10^{5}$	$2{,}82 \cdot 10^{-2}$	$1{,}92 \cdot 10^{-2}$	$2{,}81 \cdot 10^{-2}$
$1{,}4 \cdot 10^{5}$	$2{,}99 \cdot 10^{-2}$	$1{,}98 \cdot 10^{-2}$	$2{,}9 \cdot 10^{-2}$
$1{,}45 \cdot 10^{5}$	$3{,}11 \cdot 10^{-2}$	$2{,}07 \cdot 10^{-2}$	$3{,}16 \cdot 10^{-2}$

Algoritmos que son $O(nlogn)$


Algoritmos de ordenación O(nlogn)
Tiempo de ejecución (s)
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
0
20000
40000
60000
80000
100000
120000
140000
160000
Tamaño del vector (elementos)
Mergesort
Heapsort
Quicksort

Elementos	Tiempo en segundos
50	$7{,}41 \cdot 10^{-4}$
100	$6{,}13 \cdot 10^{-3}$
150	$2{,}05 \cdot 10^{-2}$
200	$4{,}5 \cdot 10^{-2}$
250	$8{,}92 \cdot 10^{-2}$
300	0,15
350	0,24
400	0,35
450	0,5
500	0,68
550	0,91
600	1,17
650	1,49
700	1,9
750	2,34
800	2,88
850	3,44
900	4,07
950	4,79
1,000	5,57
1,050	6,43
1,100	7,44
1,150	8,51
1,200	9,63
1,250	10,87
1,300	12,27
1,350	13,73
1,400	15,51
1,450	16,79

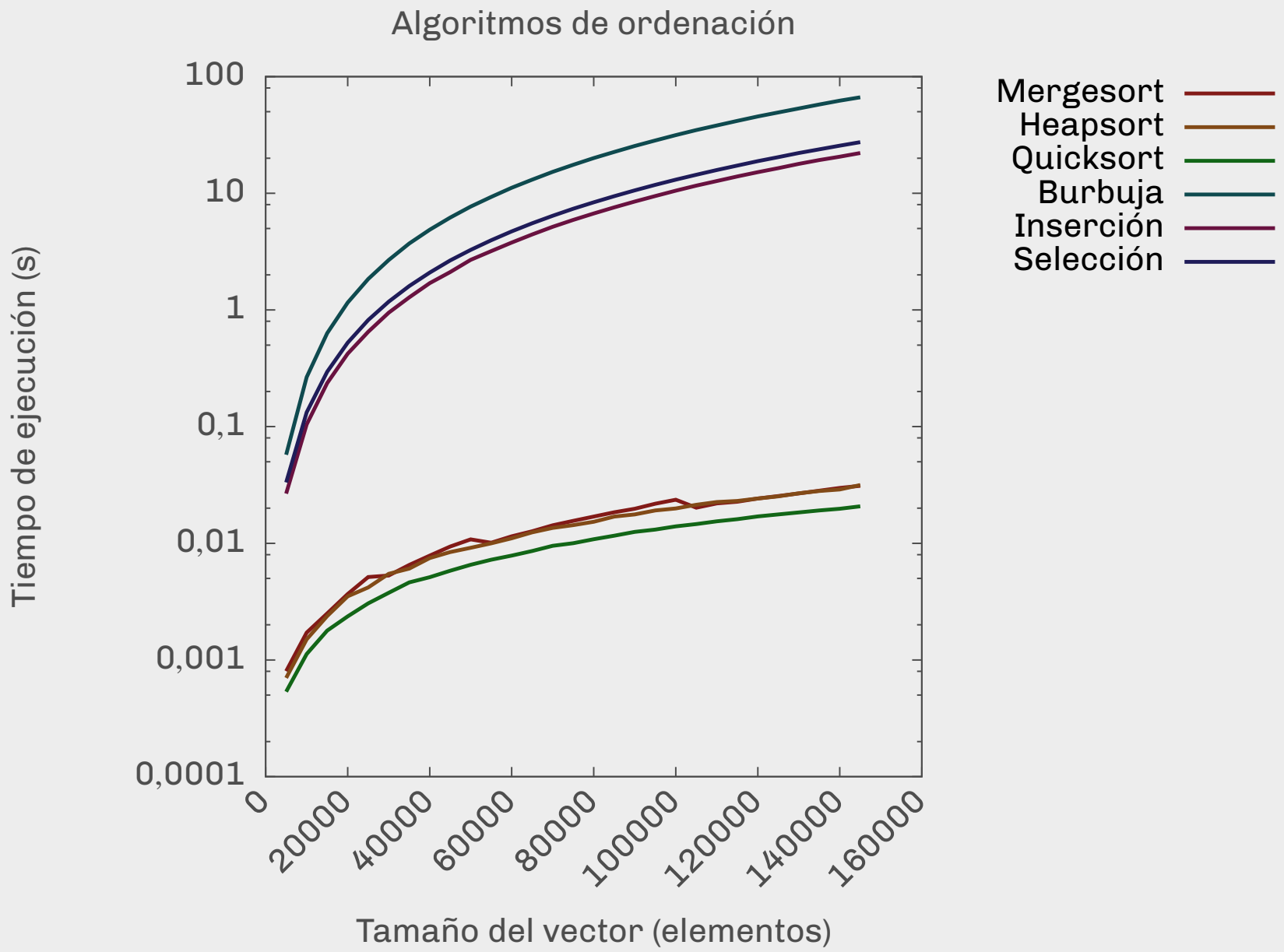
Floyd

Elementos	Tiempo en segundos
5	$1 \cdot 10^{-6}$
6	$1 \cdot 10^{-6}$
7	$1 \cdot 10^{-6}$
8	$2 \cdot 10^{-6}$
9	$4 \cdot 10^{-6}$
10	$7 \cdot 10^{-6}$
11	$1{,}3 \cdot 10^{-5}$
12	$2{,}7 \cdot 10^{-5}$
13	$5{,}1 \cdot 10^{-5}$
14	$1 \cdot 10^{-4}$
15	$2 \cdot 10^{-4}$
16	$4{,}37 \cdot 10^{-4}$
17	$8{,}23 \cdot 10^{-4}$
18	$1{,}58 \cdot 10^{-3}$
19	$3{,}17 \cdot 10^{-3}$
20	$6{,}45 \cdot 10^{-3}$
21	$1{,}41 \cdot 10^{-2}$
22	$2{,}58 \cdot 10^{-2}$
23	$5{,}24 \cdot 10^{-2}$
24	0,1
25	0,21
26	0,41
27	0,81
28	1,62
29	3,26
30	6,51

Hanoi

Comparativa de los algoritmos de ordenación

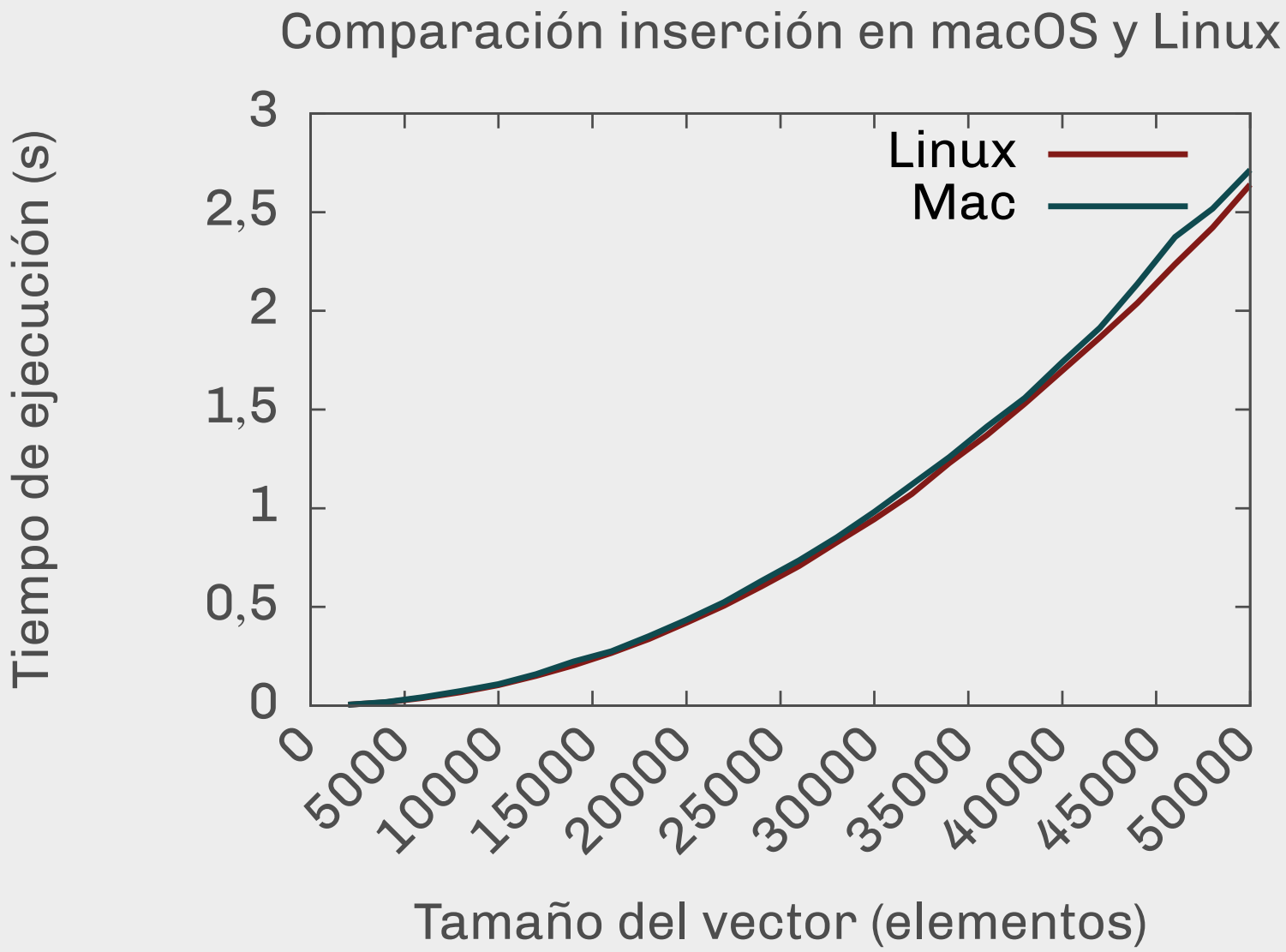
Recopilando los datos de todos los algoritmos de ordenación hemos realizado una tabla comparativa en la que se muestra qué algoritmo es el más eficiente.


Algoritmos de ordenación
Tiempo de ejecución (s)
100
10
1
0,1
0,01
0,001
0,0001
0
20000
40000
60000
80000
100000
120000
140000
160000
Tamaño del vector (elementos)
Mergesort
Heapsort
Quicksort
Burbuja
Inserción
Selección

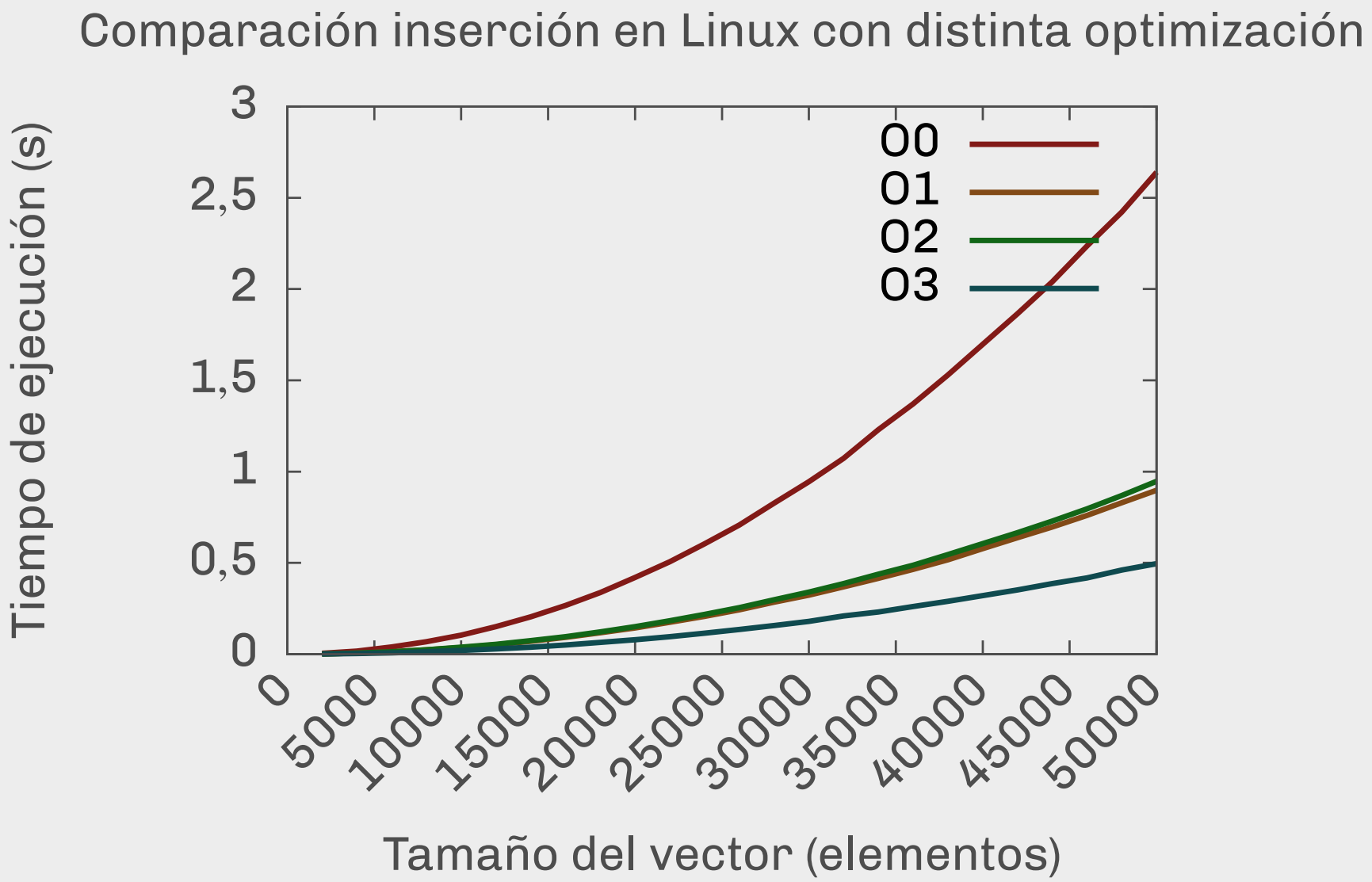
Comparativa según optimización y sistema operativo

Hemos elegido un representante de cada orden de eficiencia, y hemos realizado una comparativa para analizar cómo varían los tiempos de ejecución según el sistema operativo y el nivel de optimización.

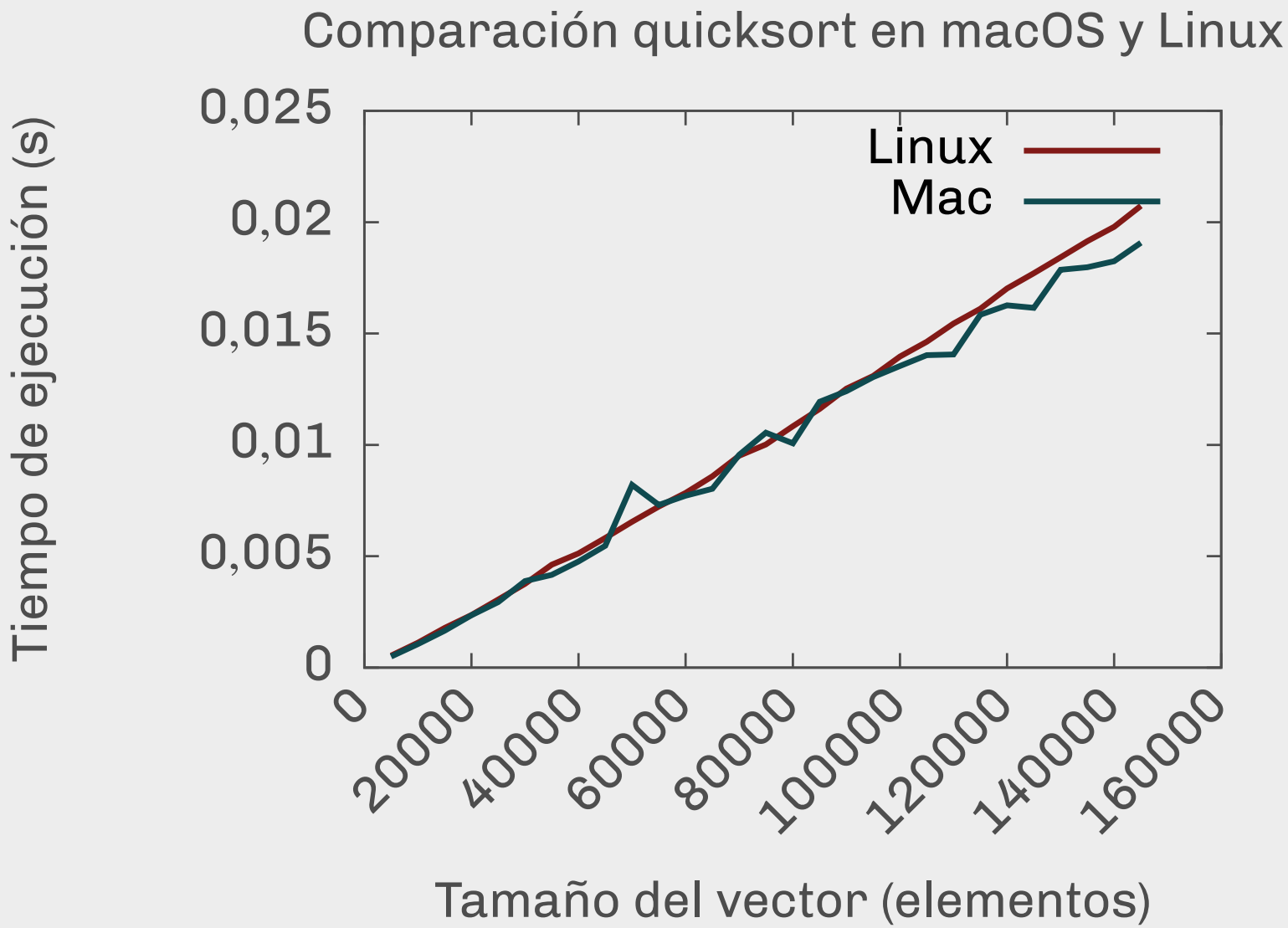
Representante de $O(n^2)$



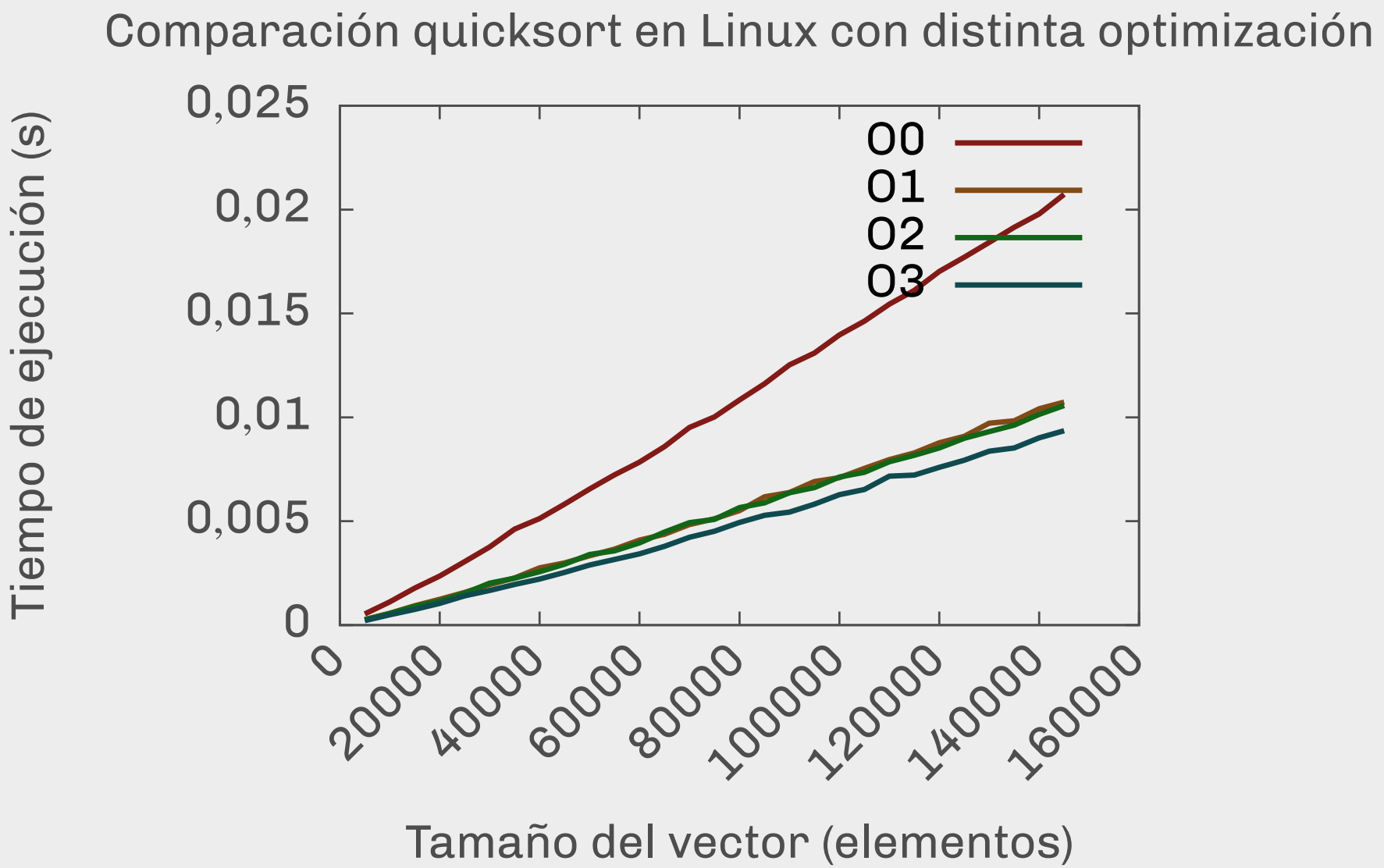
Representante de $O(n^2)$



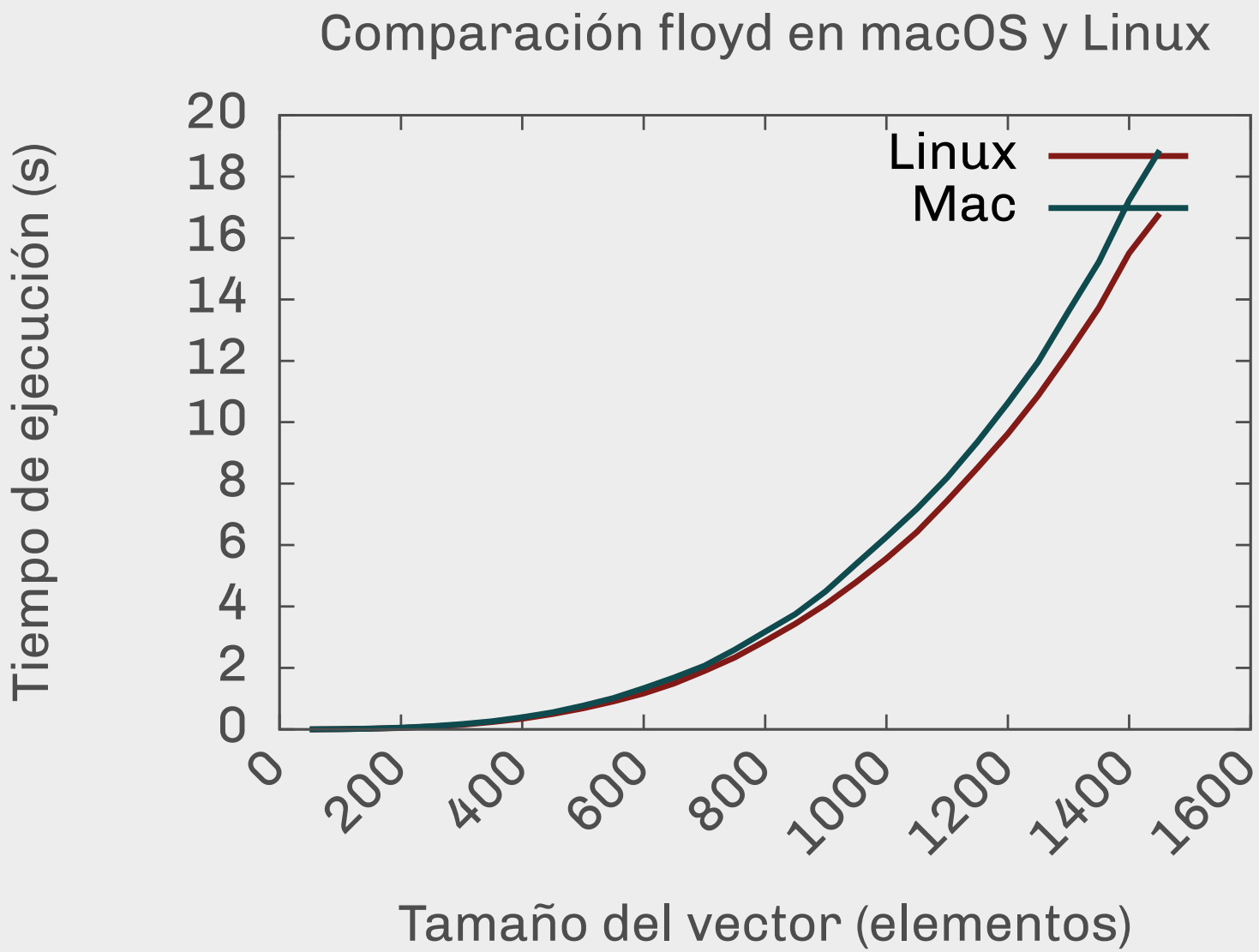
Representante de $O(n \log n)$



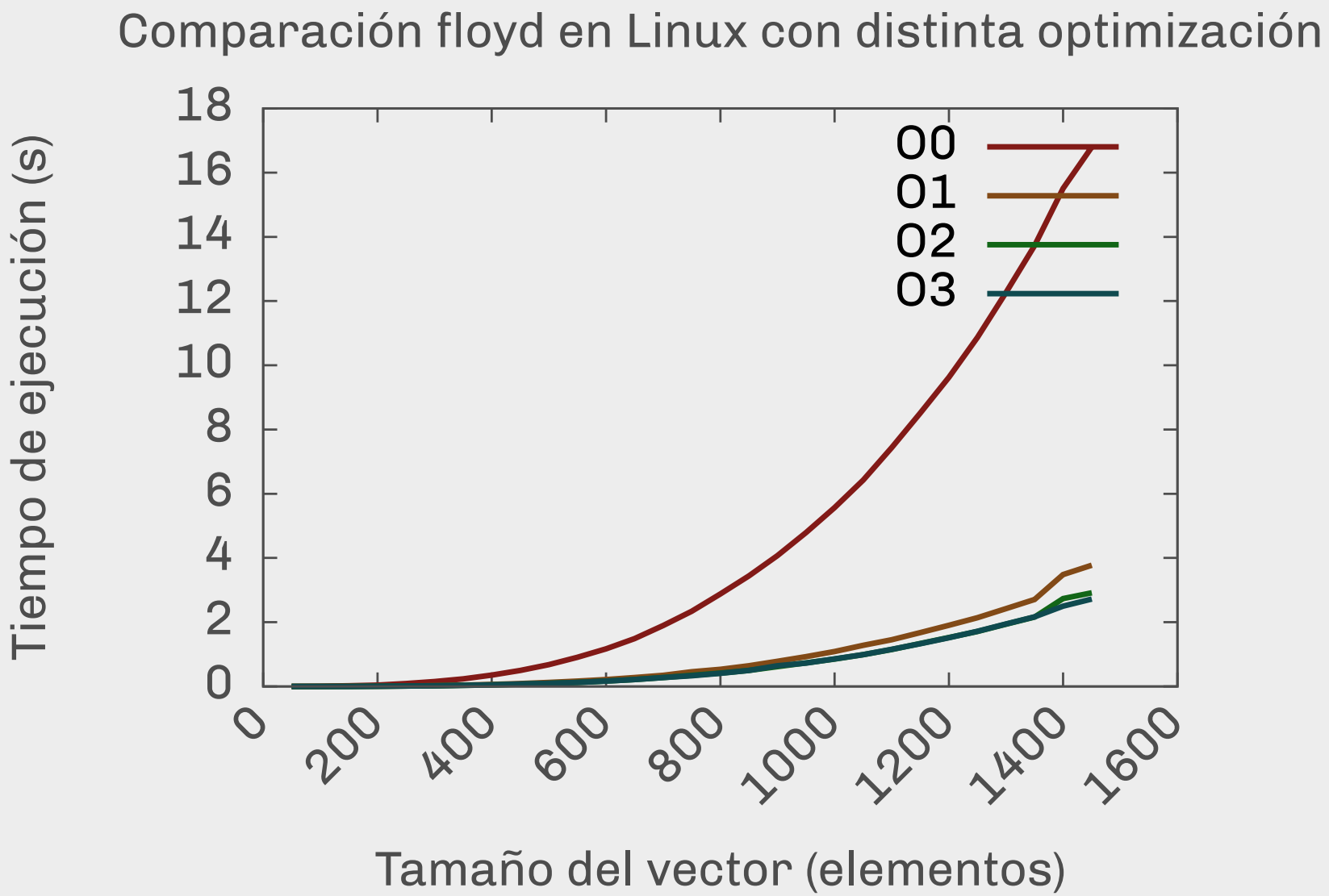
Representante de $O(n \log n)$



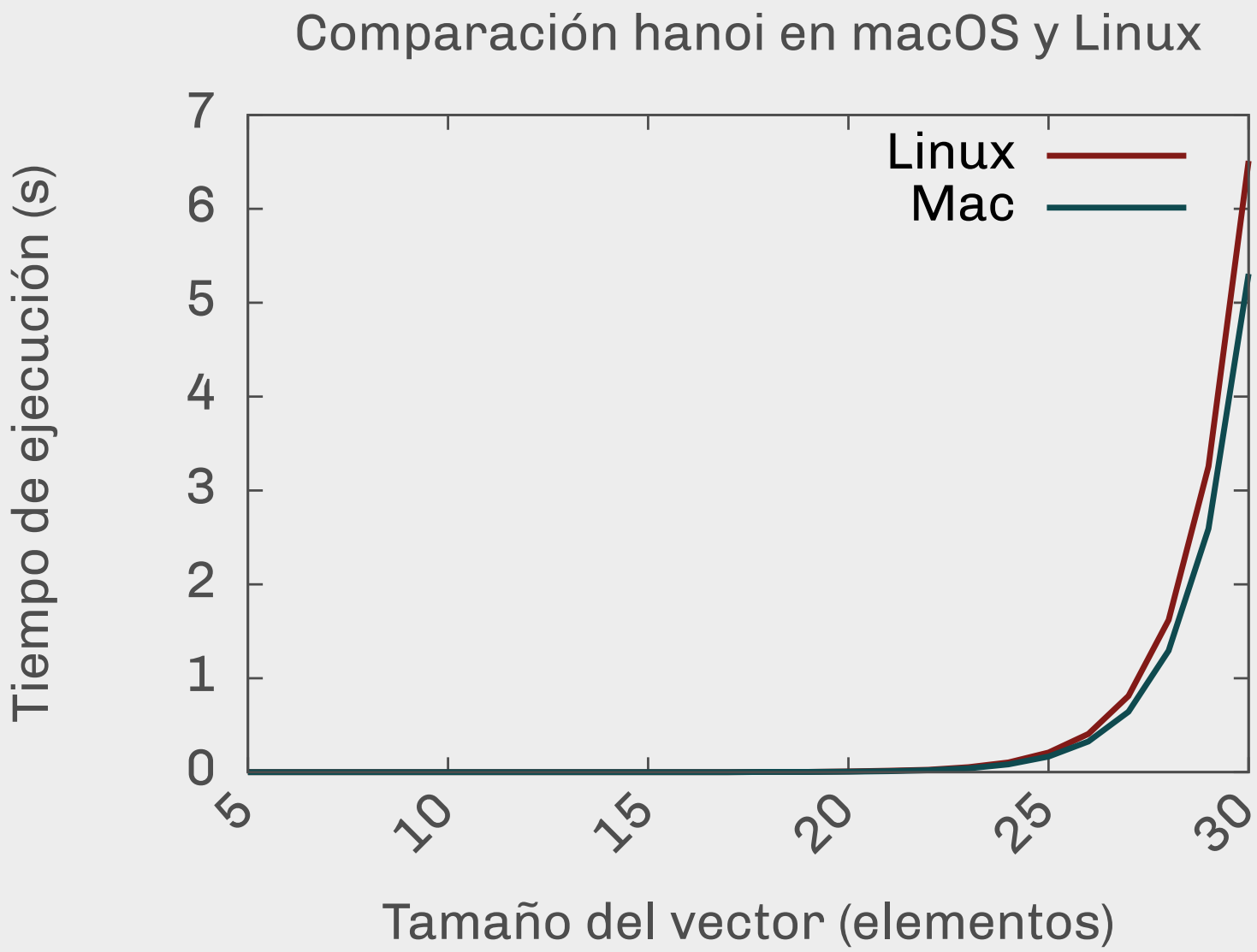
Representante de $O(n^3)$



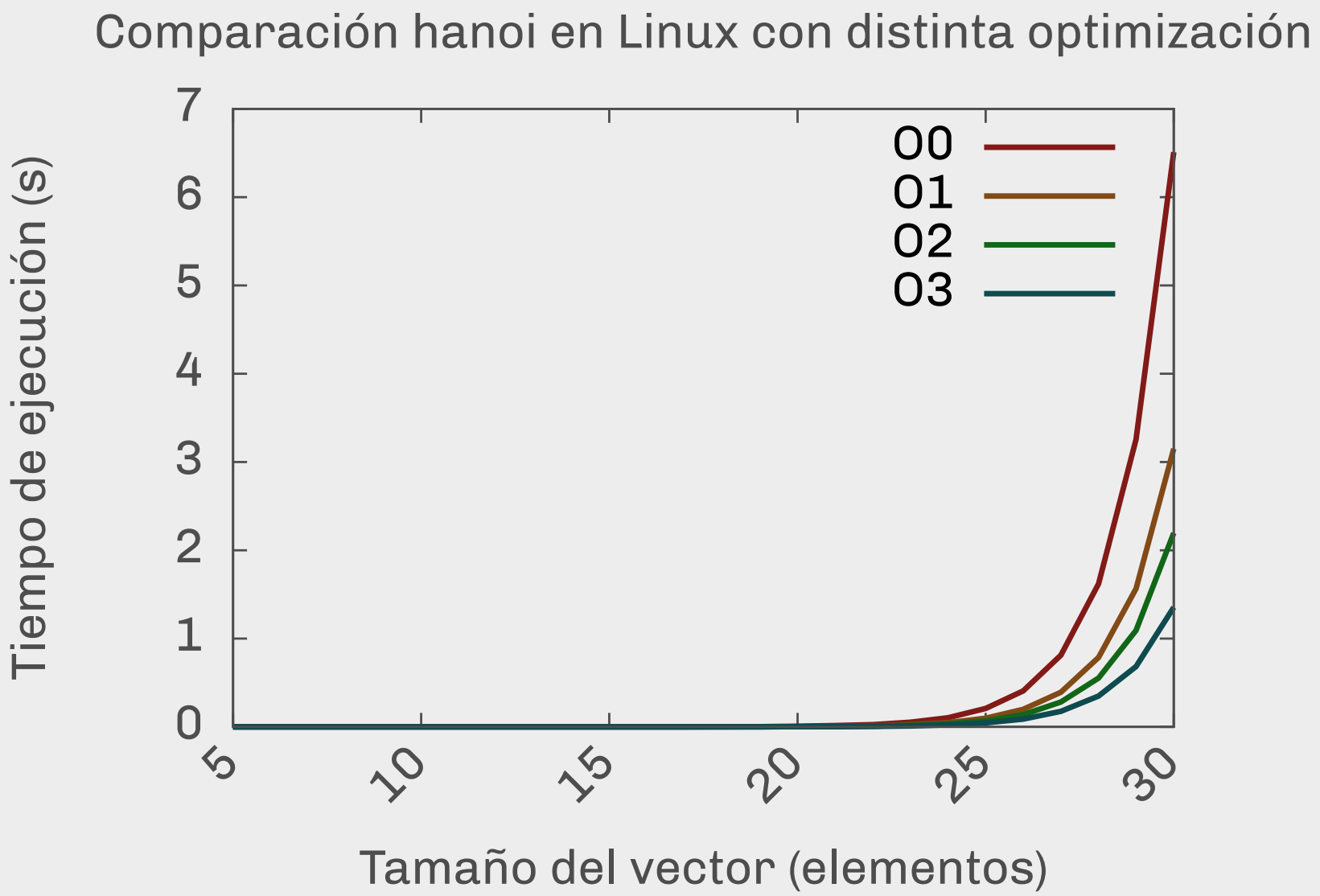
Representante de $O(n^3)$



Representante de $O(2^n)$



Representante de $O(2^n)$

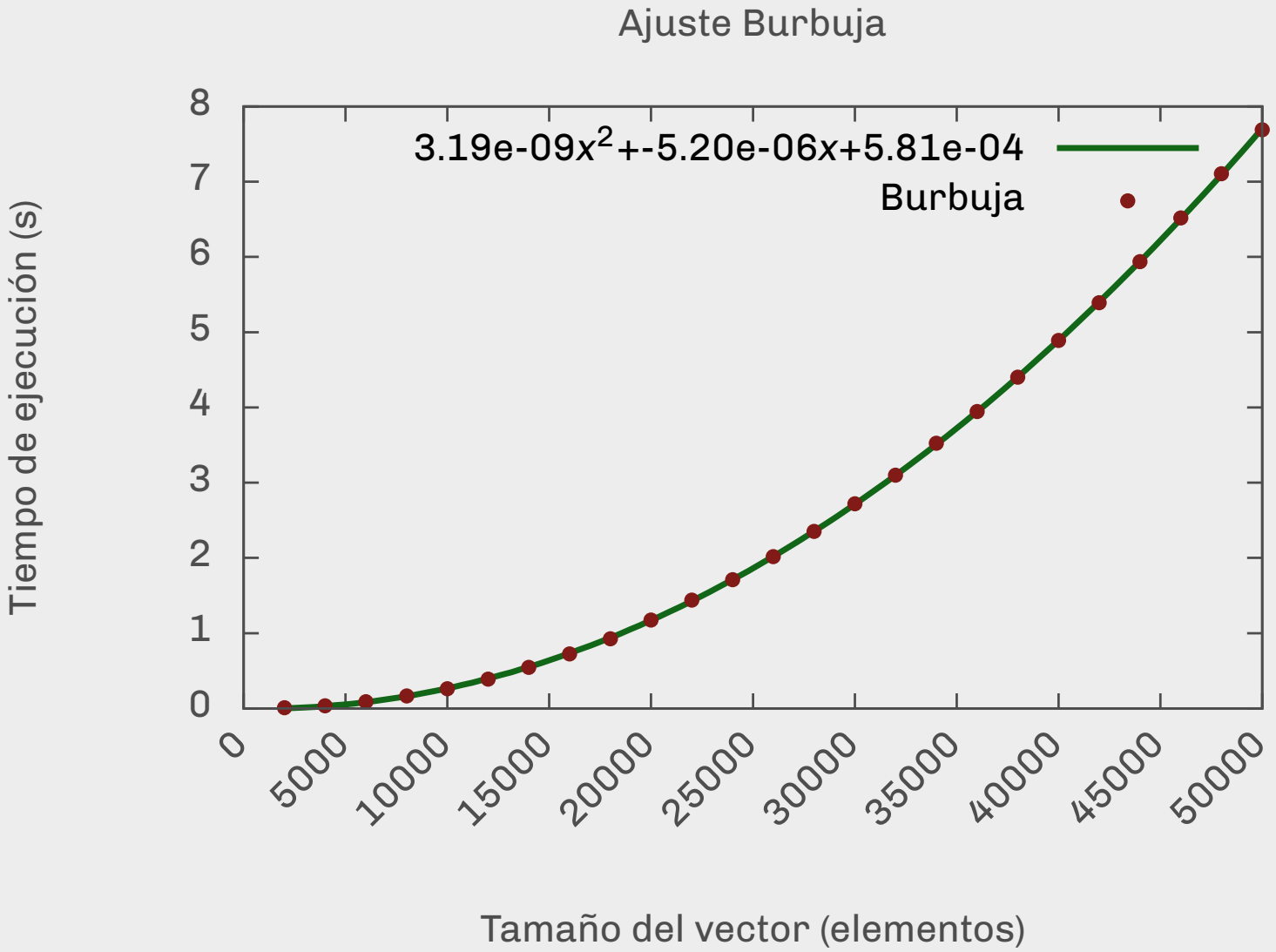


Eficiencia híbrida

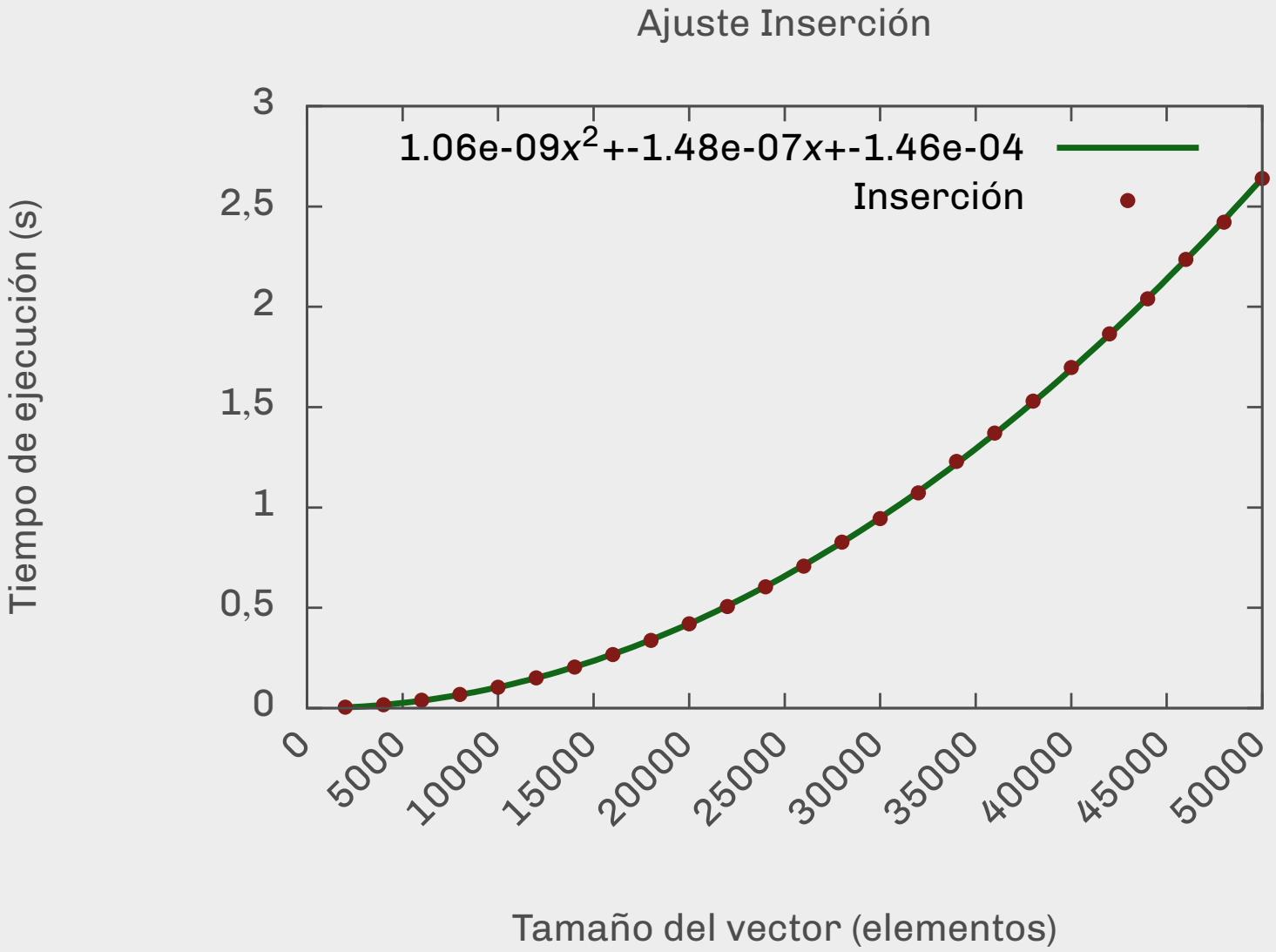
A continuación se recogen los gráficos que muestran tanto la eficiencia empírica como la función ajustada o *eficiencia híbrida* de cada algoritmo.

También se muestran, para cada algoritmo, las constantes ocultas en la expresión de la eficiencia teórica.

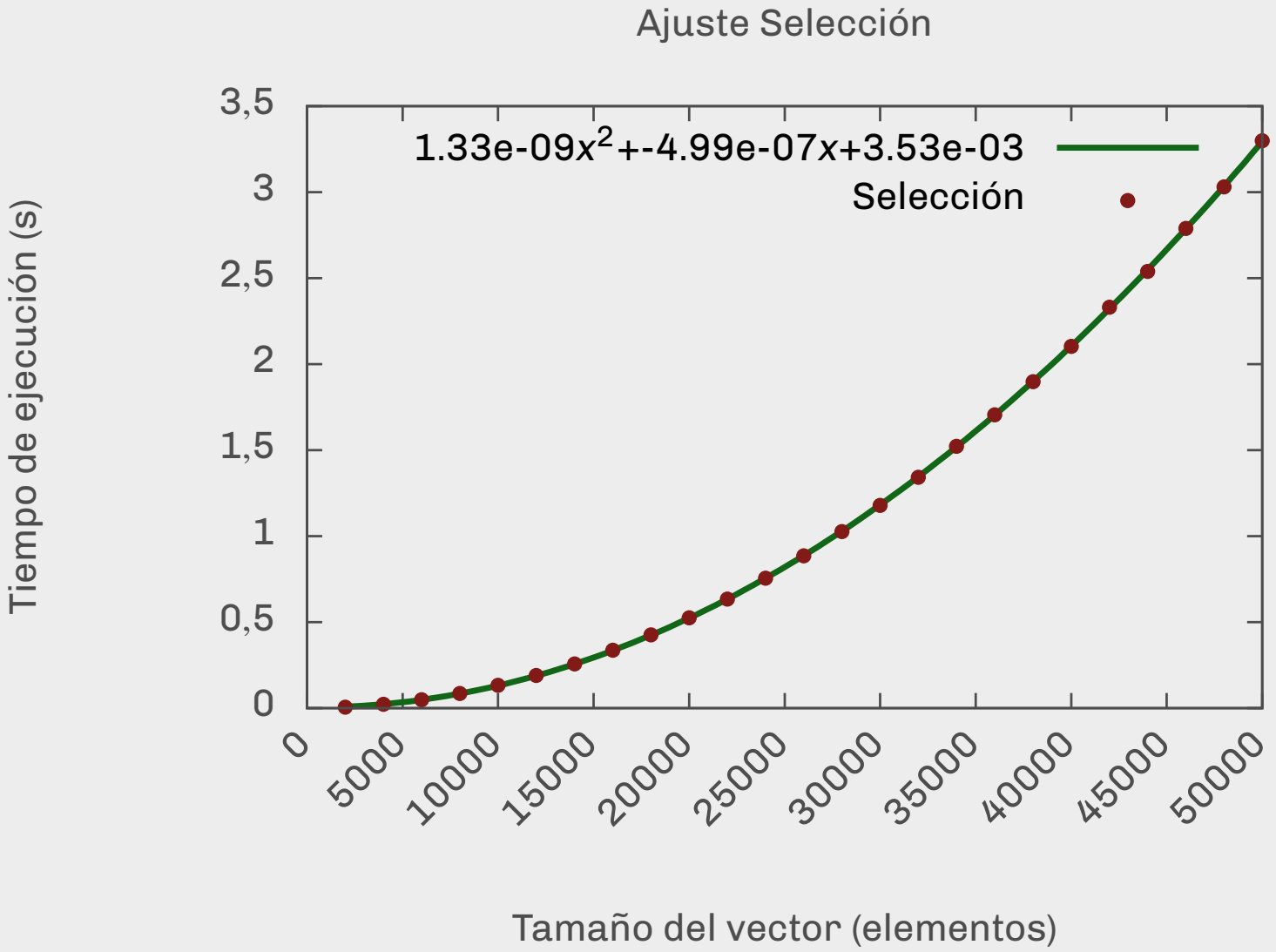
Burbuja


Ajuste Burbuja
3.19e-09x^2+-5.20e-06x+5.81e-04
Burbuja
Tiempo de ejecución (s)
Tamaño del vector (elementos)
0
1
2
3
4
5
6
7
8
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000

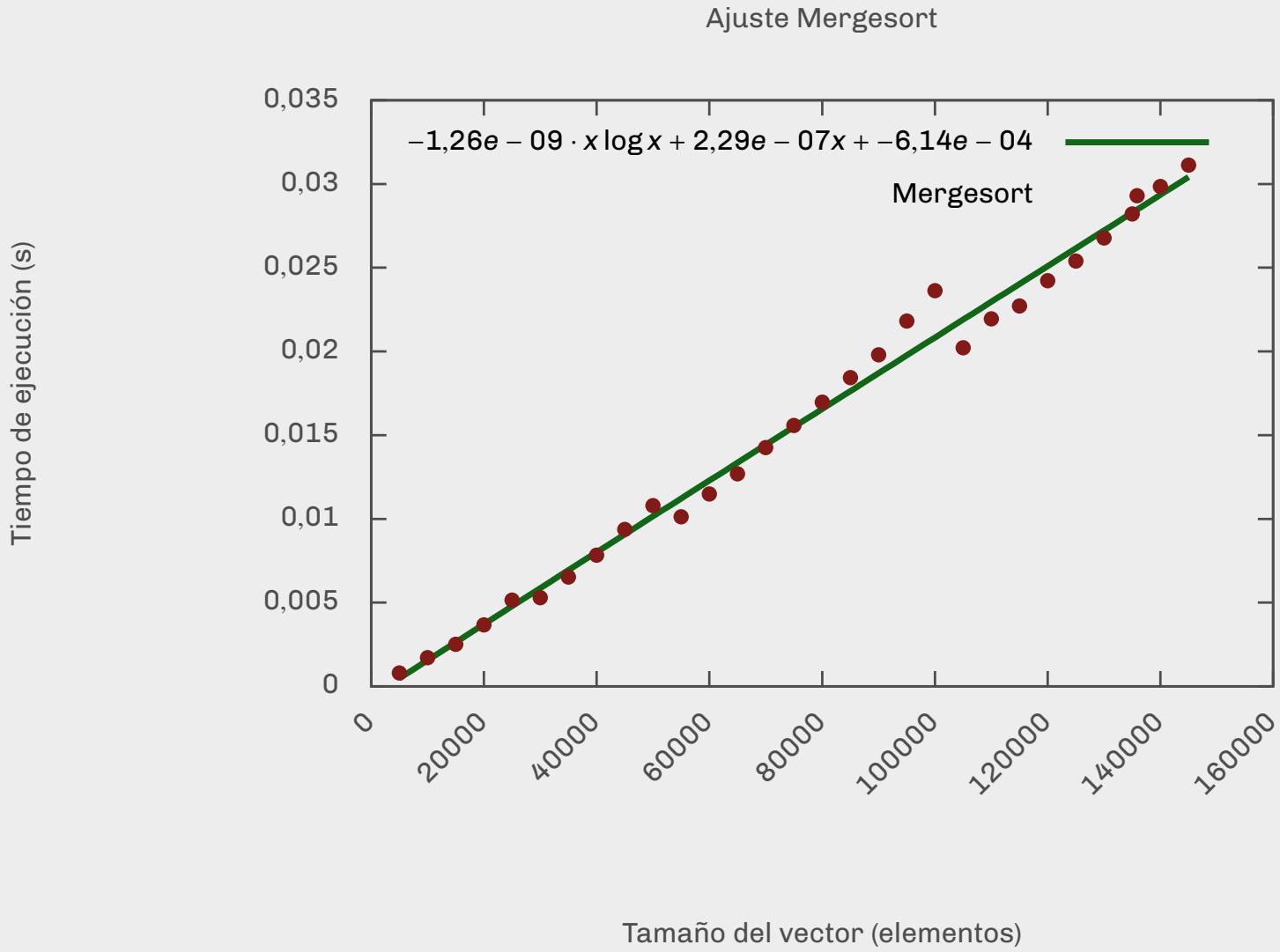
Inserción


Ajuste Inserción
1.06e-09x^2+-1.48e-07x+-1.46e-04
Inserción
Tiempo de ejecución (s)
Tamaño del vector (elementos)
0
0,5
1
1,5
2
2,5
3
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000

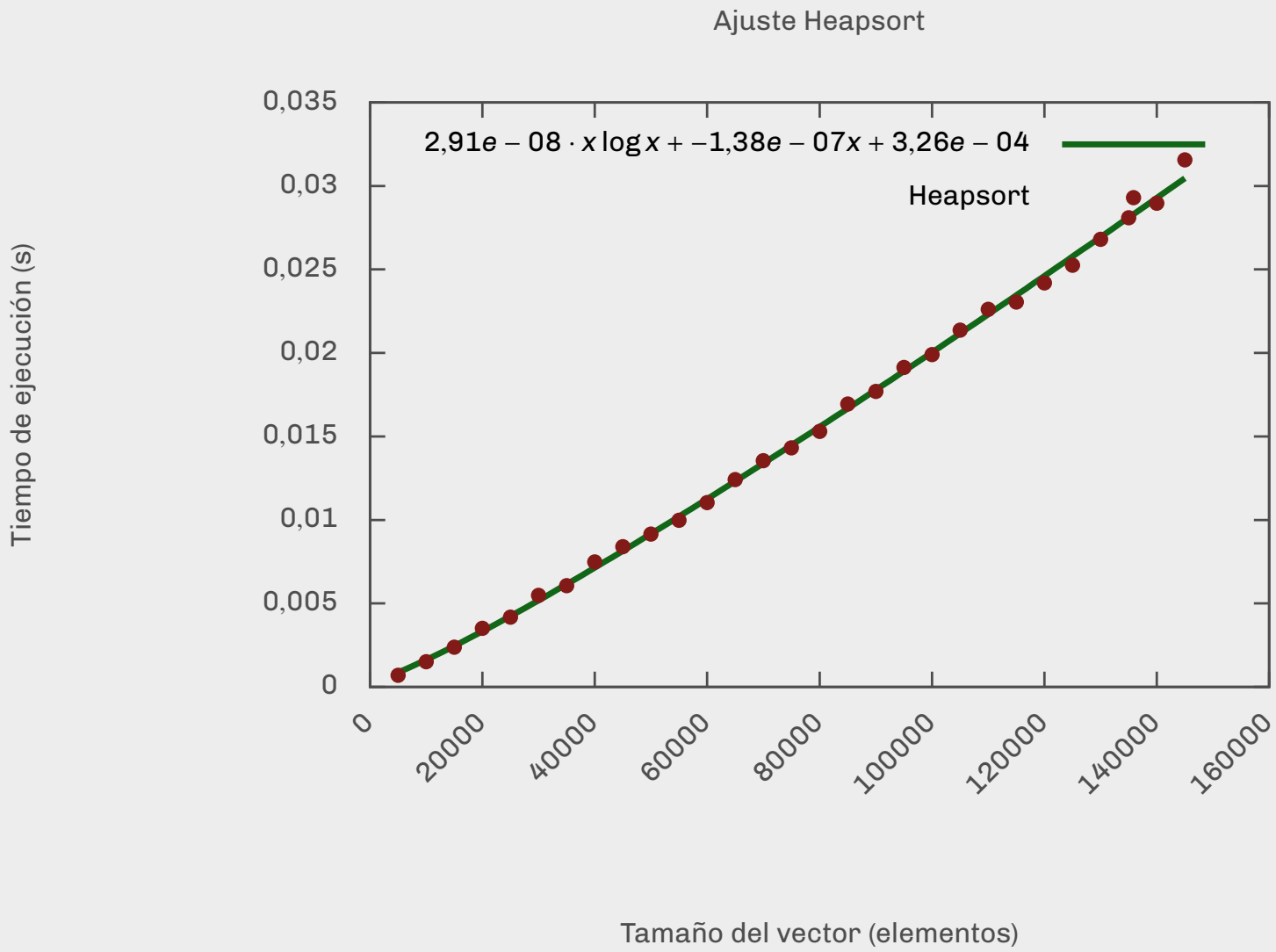
Selección


Ajuste Selección
1.33e-09x^2+-4.99e-07x+3.53e-03
Selección
Tiempo de ejecución (s)
Tamaño del vector (elementos)
0
0,5
1
1,5
2
2,5
3
3,5
0
5000
10000
15000
20000
25000
30000
35000
40000
45000
50000

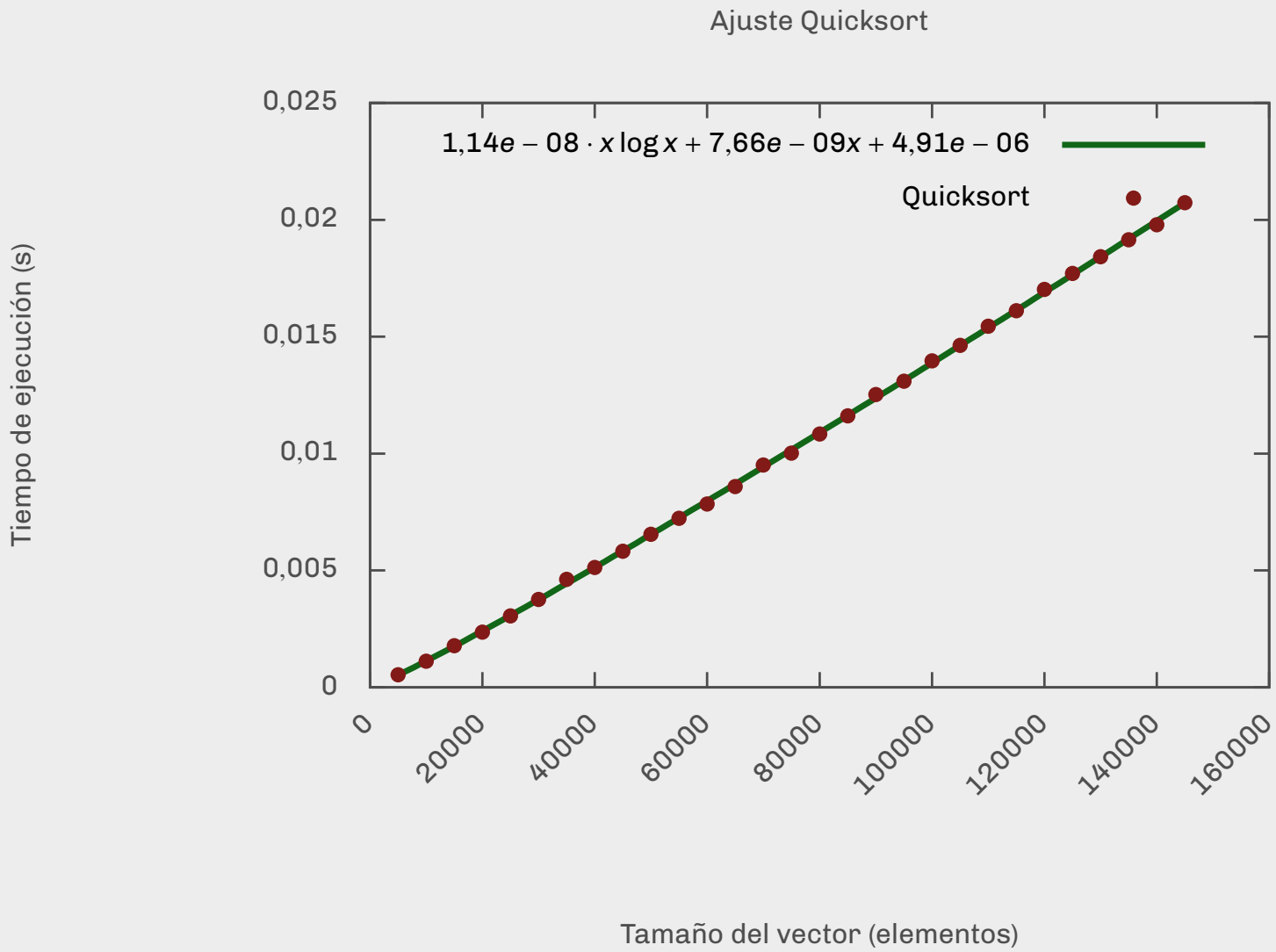
Mergesort


Ajuste Mergesort
−1,26e − 09 · x log x + 2,29e − 07x + −6,14e − 04
Mergesort
Tiempo de ejecución (s)
Tamaño del vector (elementos)
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
0
20000
40000
60000
80000
100000
120000
140000
160000

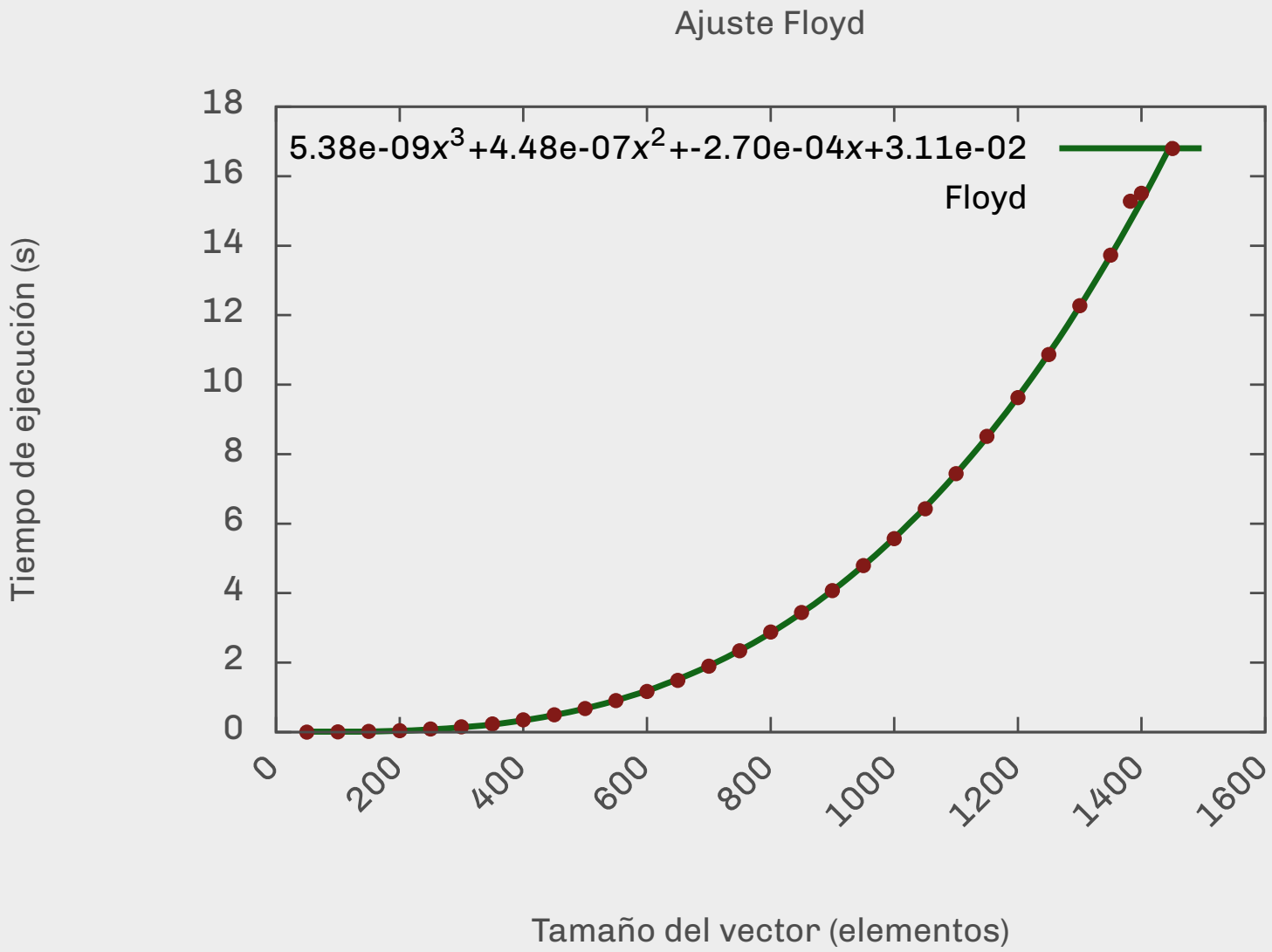
Heapsort


Ajuste Heapsort
2,91e − 08 · x log x + −1,38e − 07x + 3,26e − 04
Heapsort
Tiempo de ejecución (s)
Tamaño del vector (elementos)
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
0
20000
40000
60000
80000
100000
120000
140000
160000

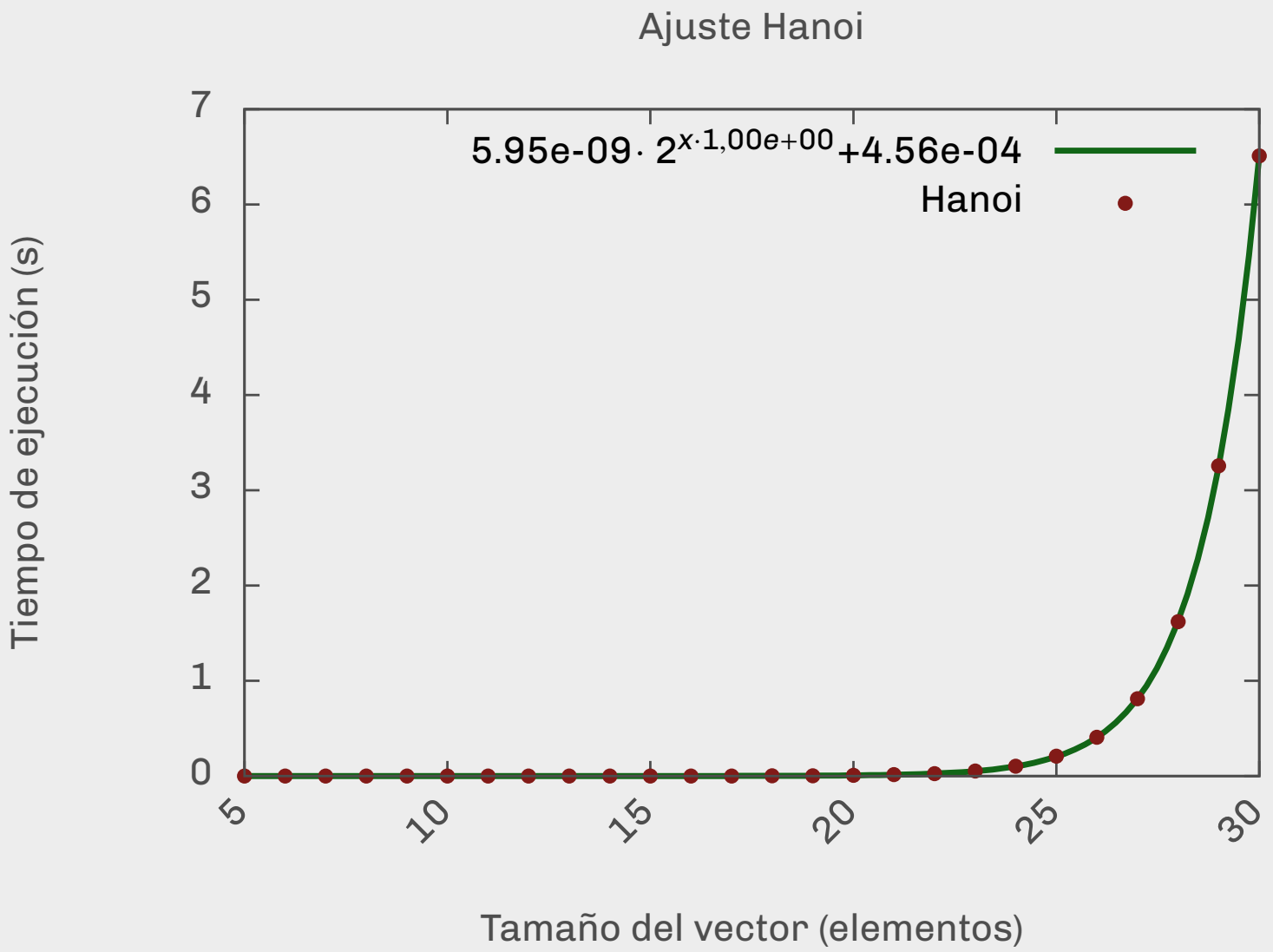
Quicksort


Ajuste Quicksort
1,14e − 08 · x log x + 7,66e − 09x + 4,91e − 06
Quicksort
Tiempo de ejecución (s)
Tamaño del vector (elementos)
0,025
0,02
0,015
0,01
0,005
0
0
20000
40000
60000
80000
100000
120000
140000
160000

Floyd


Ajuste Floyd
5.38e-09x^3+4.48e-07x^2+-2.70e-04x+3.11e-02
Floyd
Tiempo de ejecución (s)
Tamaño del vector (elementos)
0
2
4
6
8
10
12
14
16
18
0
200
400
600
800
1000
1200
1400
1600

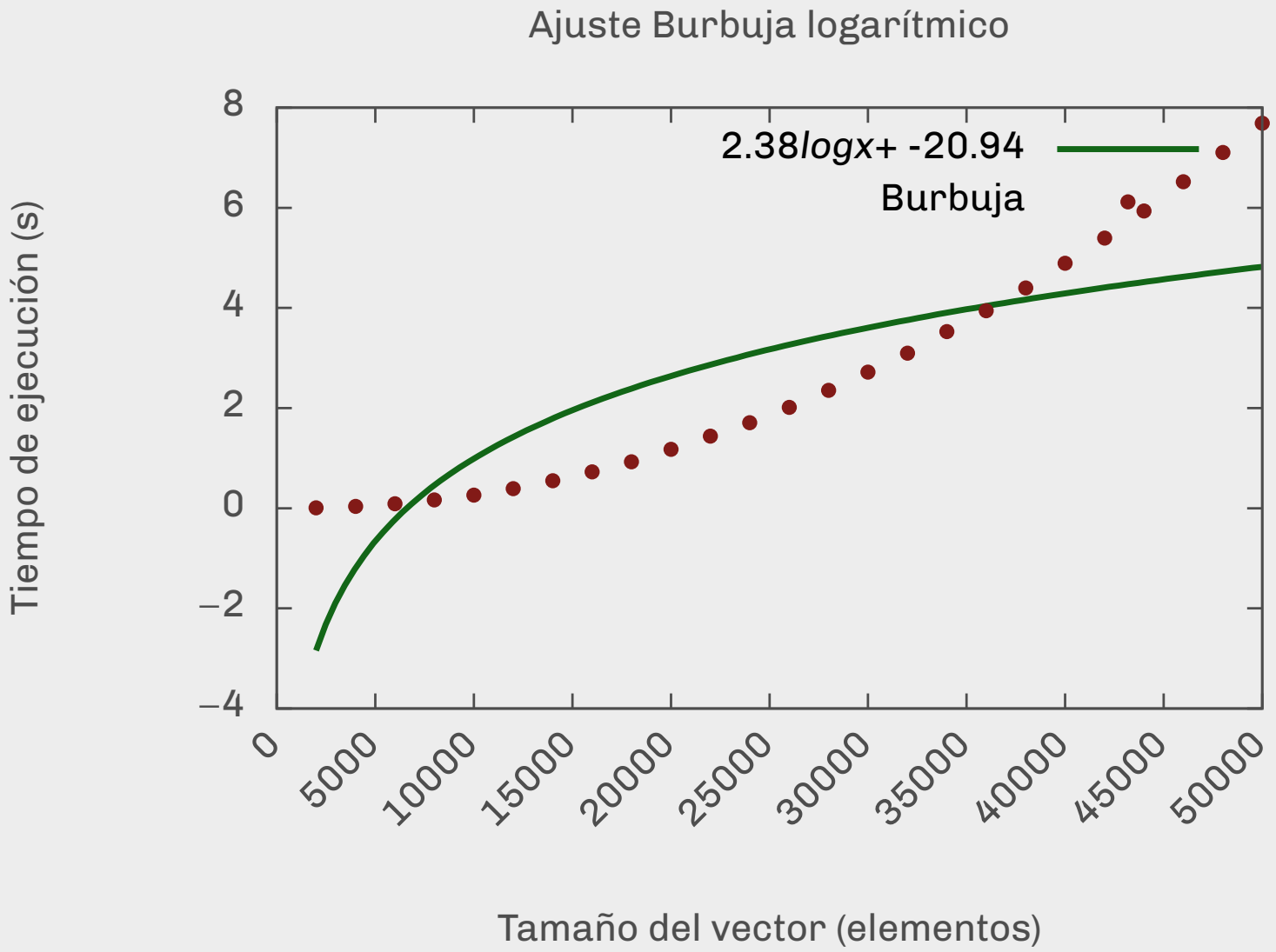
Hanoi


Ajuste Hanoi
5.95e-09· 2^x·1,00e+00+4.56e-04
Hanoi
Tiempo de ejecución (s)
Tamaño del vector (elementos)
0
1
2
3
4
5
6
7
5
10
15
20
25
30

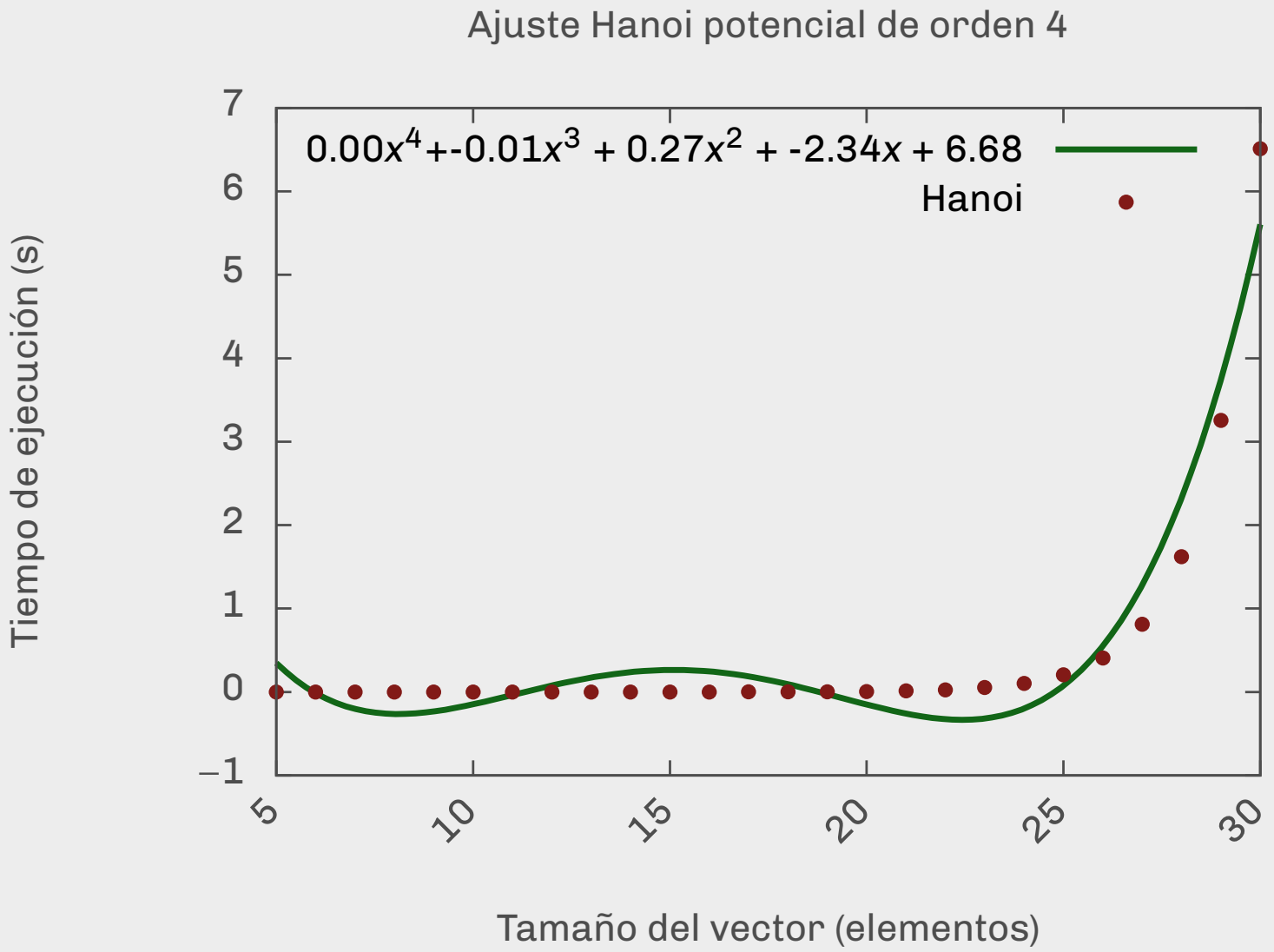
Ajustes con otras funciones

Veamos por último lo que ocurre si intentamos ajustar los datos recogidos experimentalmente a una función que no coincide con la eficiencia teórica del algoritmo.

Burbuja con función logarítmica



Hanoi con función potencial de orden 4



Características de los PC donde se ha ejecutado

- Apple MacBook Pro, Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz, 8GB RAM.
  Compilador: clang-800.0.38
  Sistema operativo: macOS Sierra

- Dell XPS 13, Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz, 8GB RAM.
  Compilador: g++ 6.3.1
  Sistema operativo: Arch Linux