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

Introducción

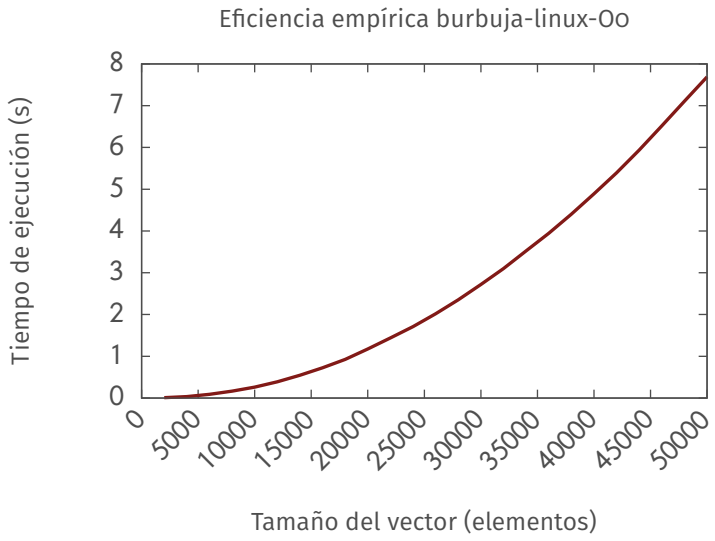
El objetivo de esta práctica es estudiar la eficiencia de los algoritmos que se nos proponen. Para ello realizaremos un estudio teórico, empírico e híbrido de cada uno de ellos. El estudio teórico consiste en expresar el número de operaciones $T(n)$ requeridas para un problema concreto en función del tamaño n, siempre en el *caso peor*. Para el estudio empírico hemos medido los tiempos de ejecución de cada algoritmo para cada uno de los tamaños de las entradas. Para el estudio híbrido, la idea es determinar las constantes ocultas en la expresión de la eficiencia teórica, y comprobar mediante un ajuste por mínimos cuadrados si coincide con la eficiencia empírica.

Algoritmos de ordenación

Eficiencia n^2

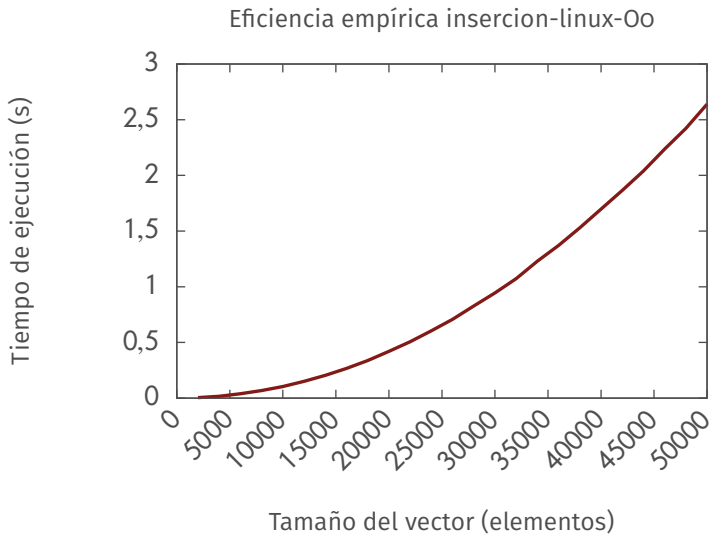
Burbuja

Revisa cada elemento de la lista con el siguiente, intercambiándose de posición si no están en el orden correcto. Su eficiencia teórica es $O(n^2)$.



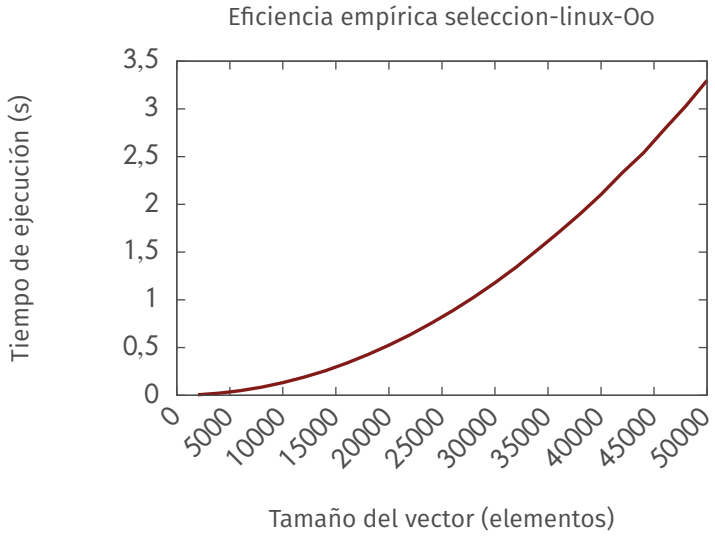
Inserción

Consideramos el elemento N-ésimo de la lista y lo ordenamos respecto de los elementos desde el primero hasta el N-1-ésimo. Su eficiencia teórica es $O(n^2)$.



Selección

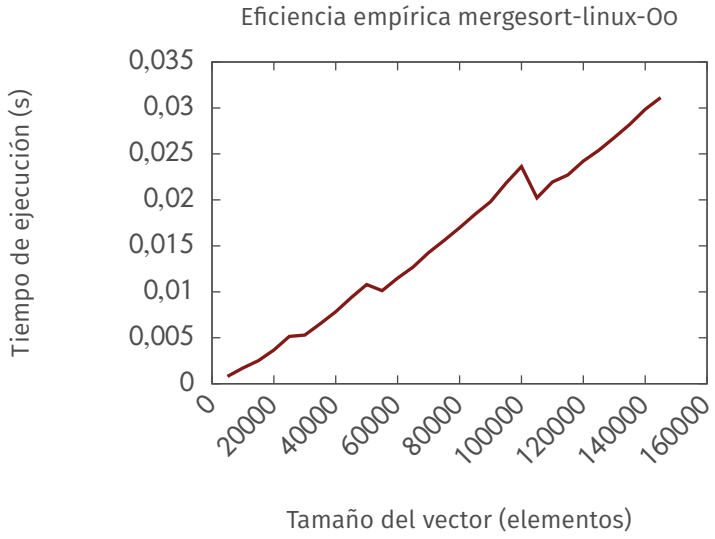
Consiste en encontrar el menor de todos los elementos de la lista e intercambiarlo con el de la primera posición. Luego con el segundo, y así sucesivamente hasta ordenarlo todo. De nuevo, su eficiencia teórica es $O(n^2)$.



Eficiencia $nlogn$

Mergesort

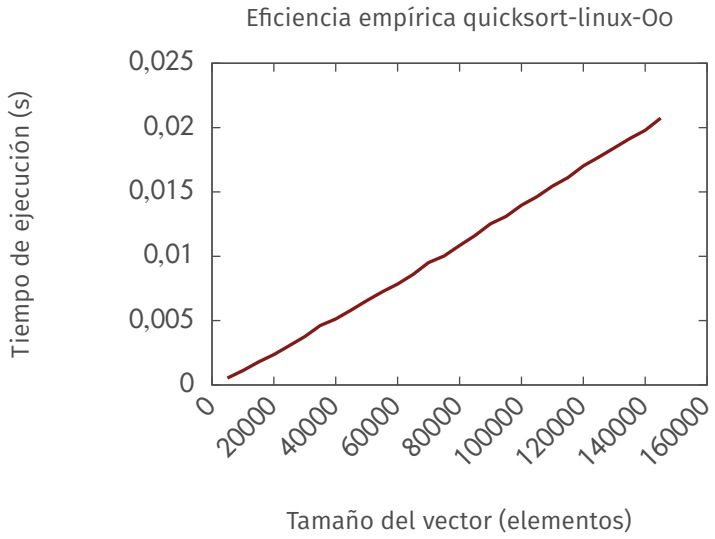
Se basa en la técnica de divide y vencerás. Consiste en dividir la lista en sublistas de la mitad de tamaño, ordenando cada una de ellas de forma recursiva. Si el tamaño de la lista es 0 ó 1 la lista ya está ordenada. Para acabar juntamos todas las sublistas en una sola. Su eficiencia teórica es $O(n \log n)$.



Podemos observar algunos picos en la gráfica pero no sabemos a que puede ser debido.

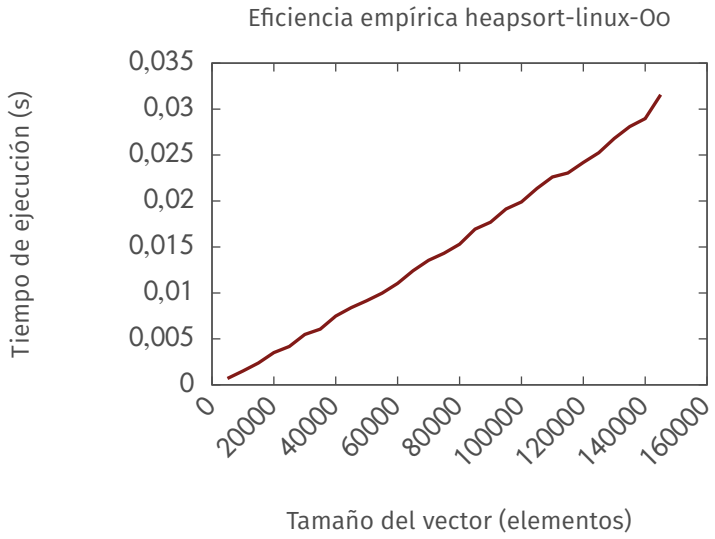
Quicksort

También se basa en la técnica de divide y vencerás. En primer lugar elegimos un elemento de la lista, que llamaremos *pivote*. A continuación los elementos de la lista se ordenarán de forma que la derecha del pivote queden los mayores y a la izquierda los menores. De esta forma dividimos la lista en dos sublistas, la de la derecha y la de la izquierda. Repetiremos el proceso mientras las sublistas tengan más de un elemento. Su eficiencia teórica también es $O(n \log n)$.



Heapsort

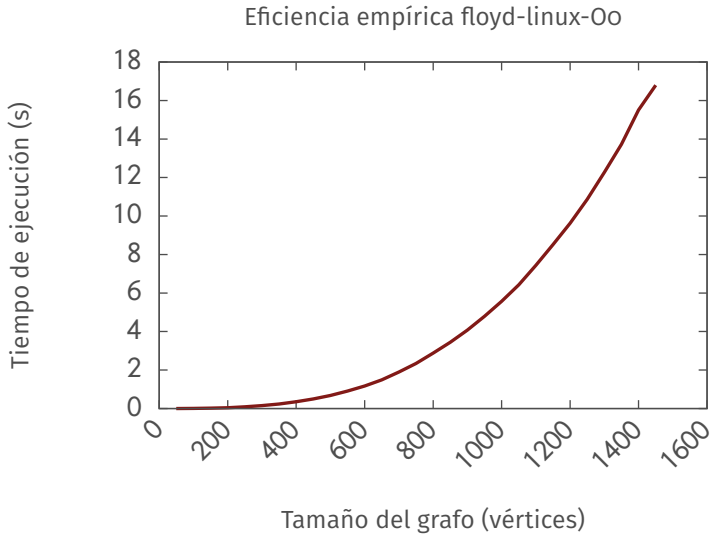
Este algoritmo consiste en almacenar todos los elementos del vector a ordenar en una estructura de datos llamada montículo (*heap*). Luego, se extrae el nodo que queda como nodo raíz del montículo (cima) en sucesivas iteraciones obteniendo el conjunto ordenado. Basa su funcionamiento en una propiedad de los montículos, por la cual, la cima contiene siempre el menor elemento (o el mayor, según se haya definido el montículo) de todos los almacenados en él. Su eficiencia teórica es $O(n \log n)$.



Otros algoritmos

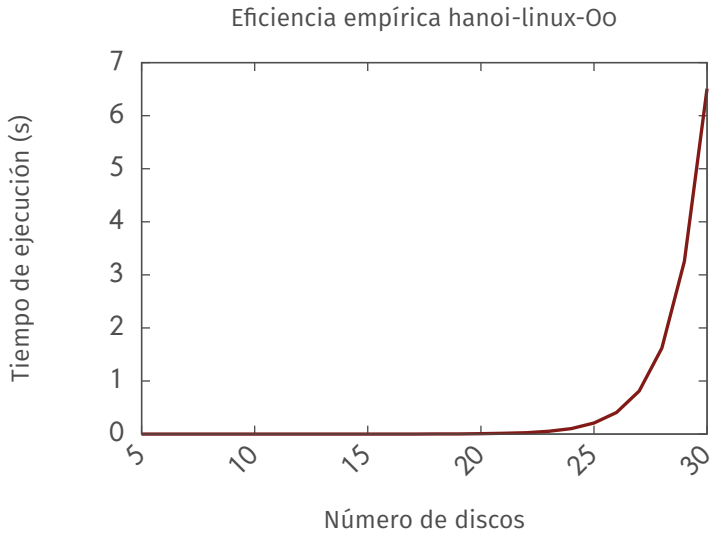
Floyd

Es un algoritmo de análisis sobre grados 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 un ejemplo de **programación dinámica**, y su eficiencia teórica es $O(n^3)$.



Hanoi

Hay tres pilas de discos, llamadas origen, auxiliar y destino. La primera de ellas está ordenada según tamaño creciente de los discos, de arriba hacia abajo. Se moverá un disco de la pila origen a la destino si hay un único disco en la pila origen. En caso contrario, se moverán todos los discos a la auxiliar, excepto el más grande. Por último, moveremos el disco mayor al destino, y movemos los $n - 1$ restantes encima del mayor. El número de pasos crece exponencialmente con el número de discos, y su eficiencia teórica es $O(2^n)$.



Cálculo de la eficiencia empírica

Puesto que la eficiencia teórica de cada algoritmo es diferente, no podemos realizar las mediciones para los mismos valores de entrada en todos los algoritmos. Así, los agrupamos según su orden de eficiencia.

Para el cálculo de la eficiencia empírica, hemos utilizado un *script* que realiza tantas ejecuciones de cada algoritmo como le indiquemos, tomando como parámetros el valor inicial, el incremento, y el valor final de los datos de entrada.

Las gráficas anteriores han sido realizadas a partir de los datos recogidos en las siguientes tablas.

Tablas

Algoritmos que son $O(n^2)$ (tiempos en segundos)

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

Algoritmos que son $O(nlog(n))$ (tiempos en segundos)

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 $O(n^3)$ (tiempos en segundos)

Elementos	Floyd
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

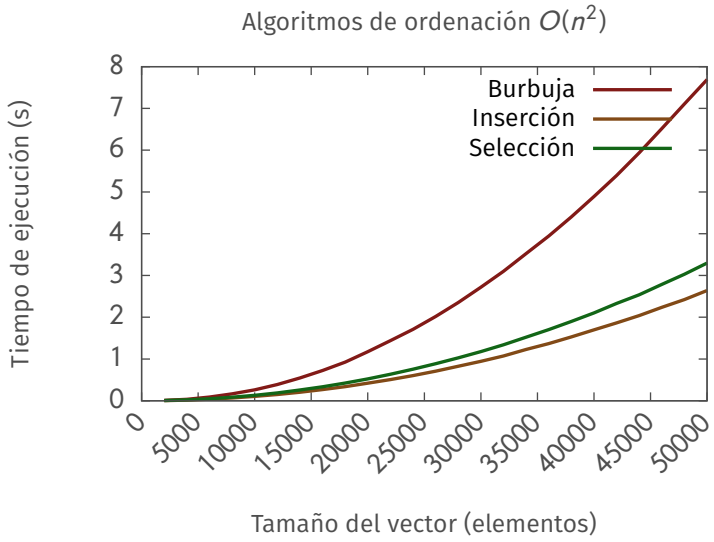
Algoritmos que son $O(2^n)$ (tiempos en segundos)

Elementos	Hanoi
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

Gráficos

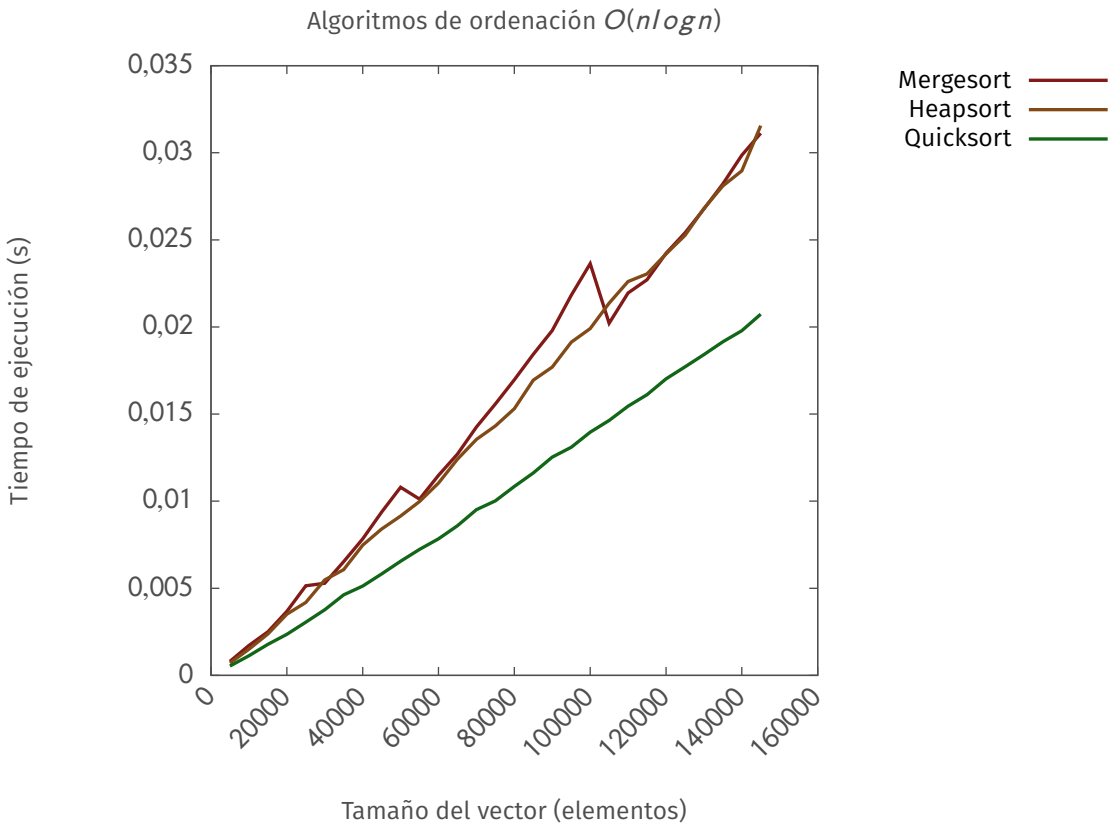
A continuación se muestran los gráficos que corresponden a la comparación de aquellos algoritmos que son de un mismo orden.

Algoritmos que son $O(n^2)$



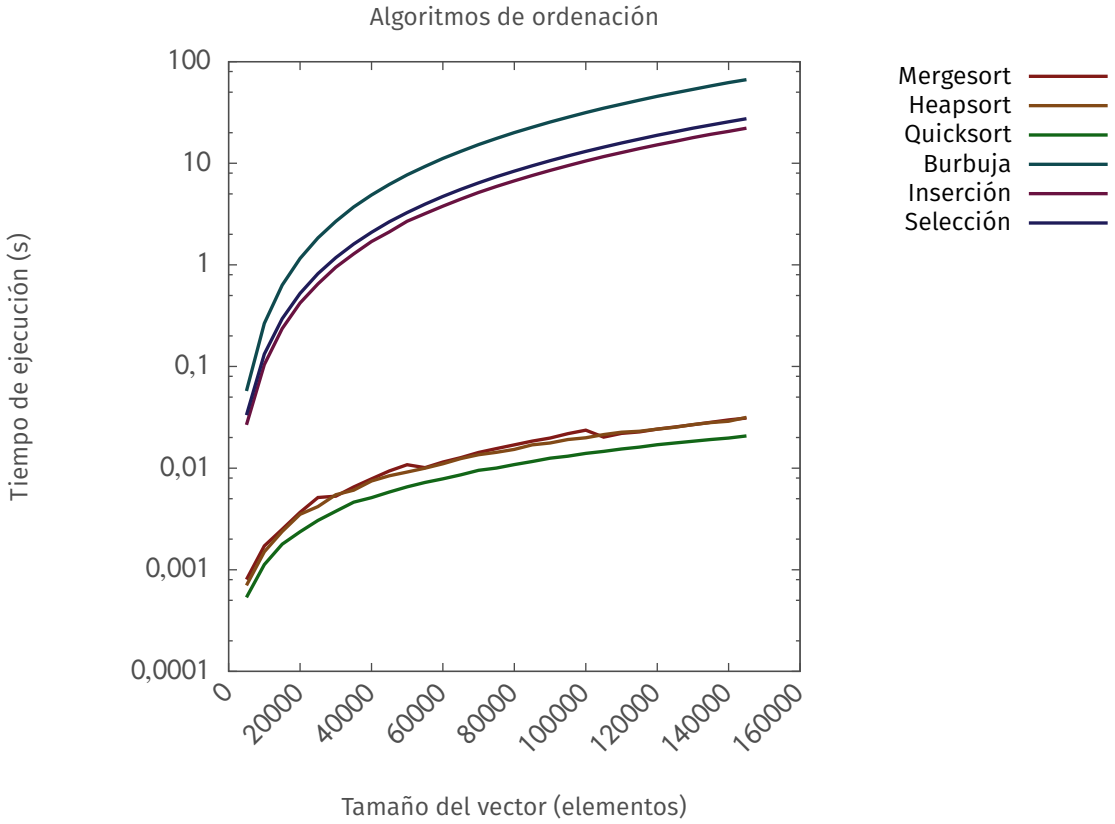
Observamos que el algoritmo burbuja es claramente más lento para estos valores. Esto se verá reflejado cuando calculemos los valores de las constantes ocultas.

Algoritmos que son $O(n \log n)$



Estos algoritmos se ejecutan en tiempos similares, aunque quicksort es ligeramente más rápido, manteniendo de forma uniforme la diferencia con los otros dos algoritmos a medida que aumenta el número de valores.

Algoritmos de ordenación



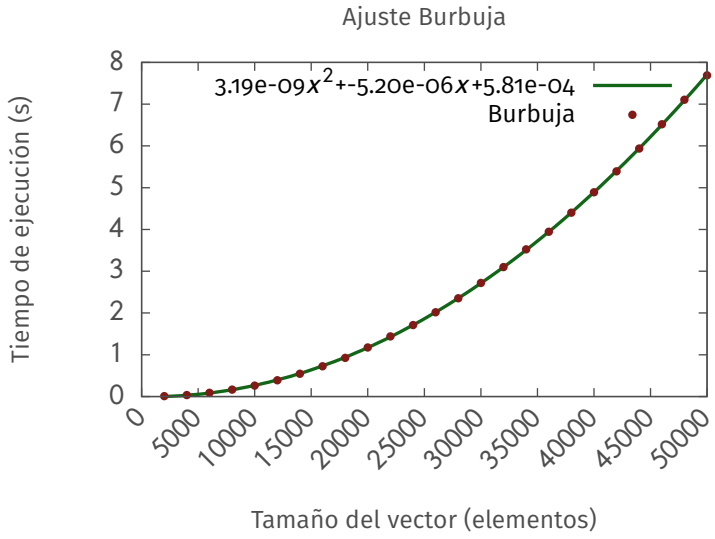
En este último gráfico, puede observarse claramente la tendencia de los algoritmos cuyo orden de eficiencia es $O(n \log n)$ a ser más rápidos que aquellos que son $O(n^2)$. Se ha utilizado una escala logarítmica para poder representar todos los algoritmos en un mismo gráfico.

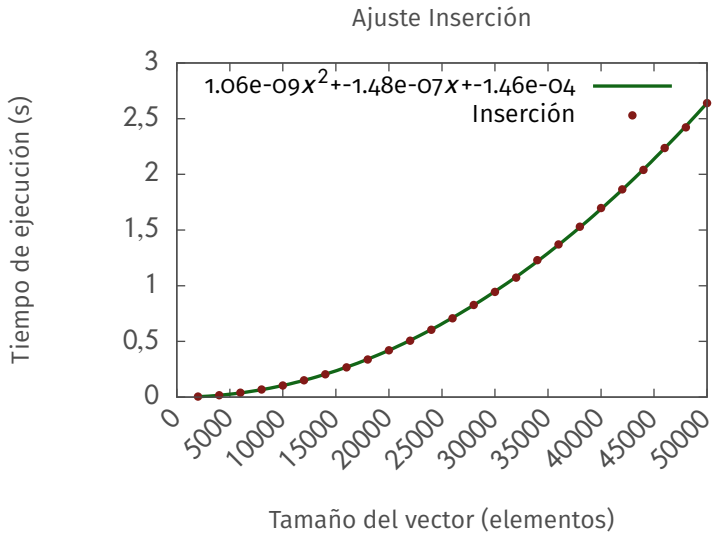
Cálculo de la eficiencia híbrida

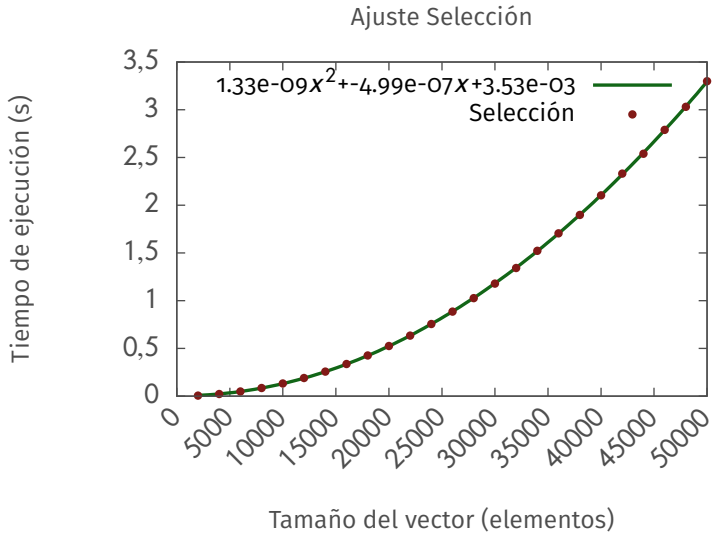
A continuación se recogen los gráficos que muestran tanto la eficiencia empírica como la función ajustada o *eficiencia híbrida* de cada algoritmo.

Eficiencia n^2

Para ajustar las algoritmos de ordenación hemos usado una función $f(x)$ de la forma: $f(x) = a_0x^2 + a_1x + a_2$ y con ayuda de gnuplot hemos calculado los valores de las constantes ocultas.

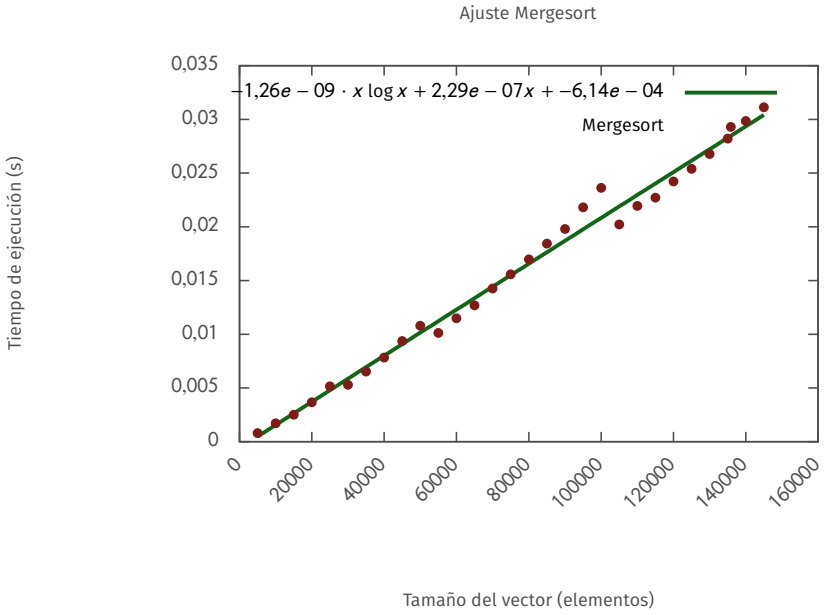


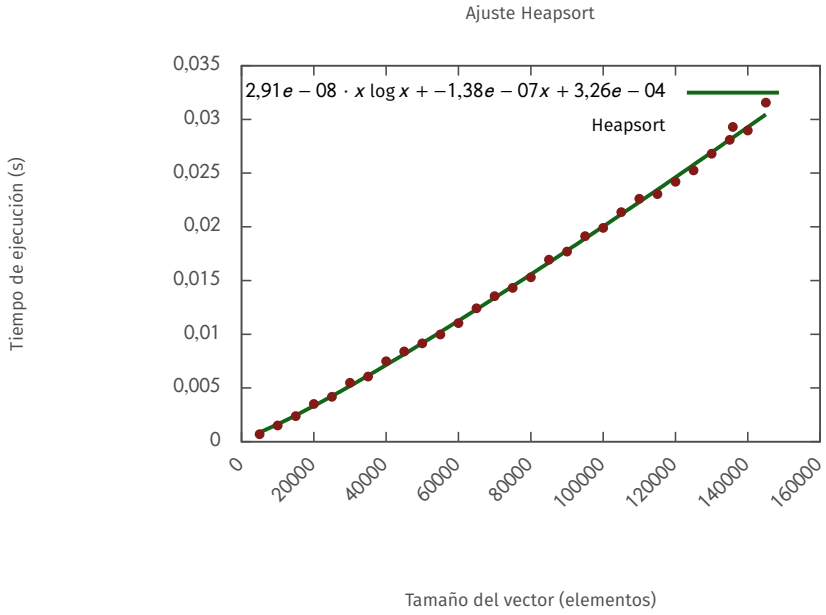


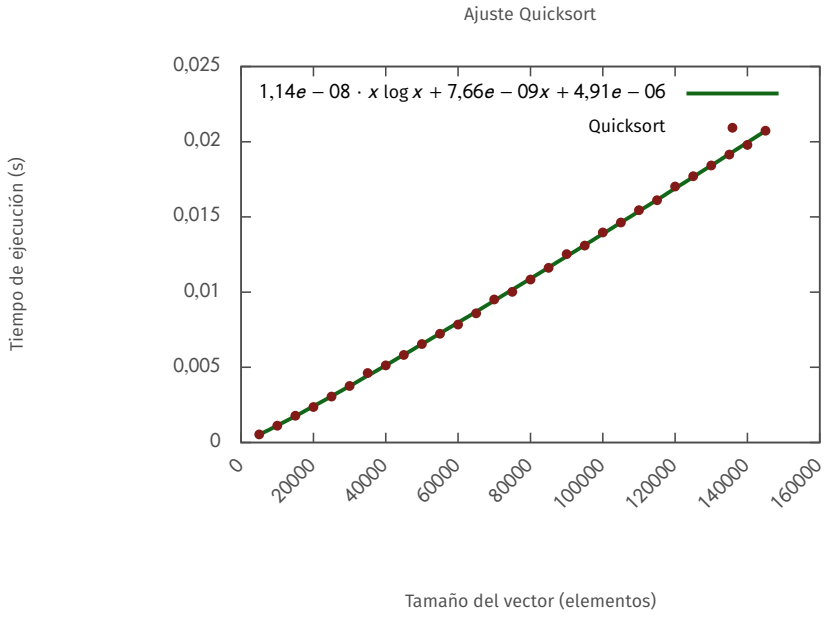


Eficiencia $nlogn$

La función utilizada para ajustar los valores en estos algoritmos de ordenación es: $f(x) = a_0x + a_1xlog(x)$

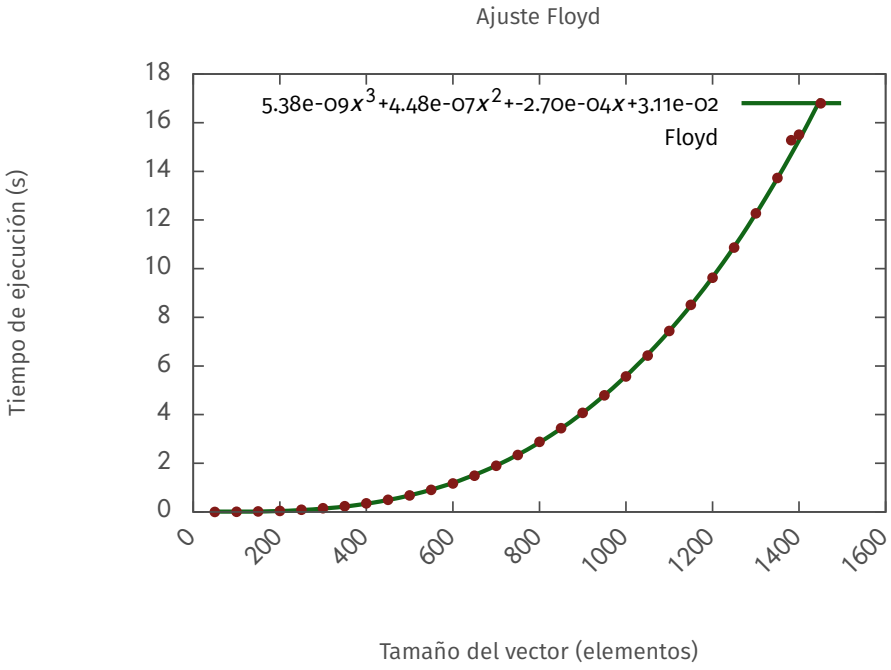






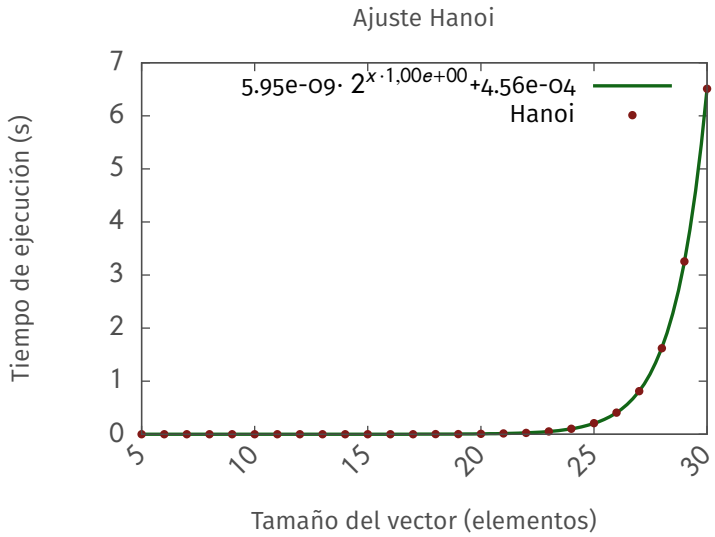
Eficiencia n^3

En el caso del algoritmo de Floyd la función $f(x)$ utilizada es: $f(x) = a_0x^3 + a_1x^2 + a_2x + a_3$.



Eficiencia 2^n

Por último, ajustamos el algoritmo de Hanoi mediante la función: $f(x) = a_0 2^{a_1 x + a_2}$ obteniendo el resultado mostrado a continuación.

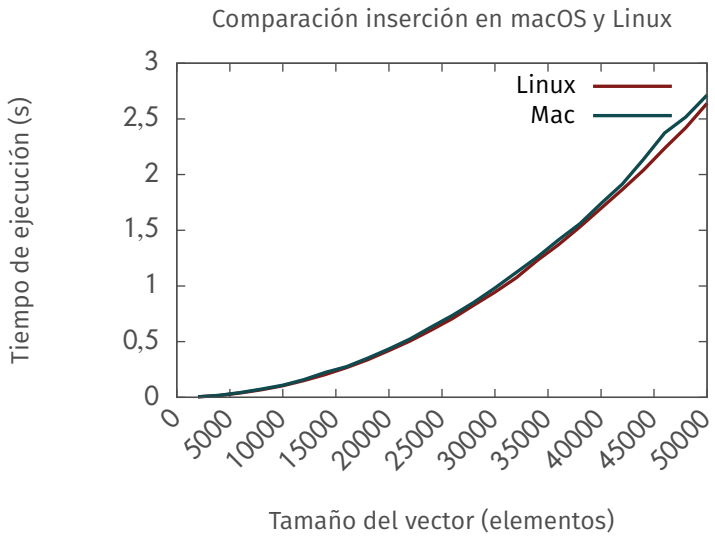


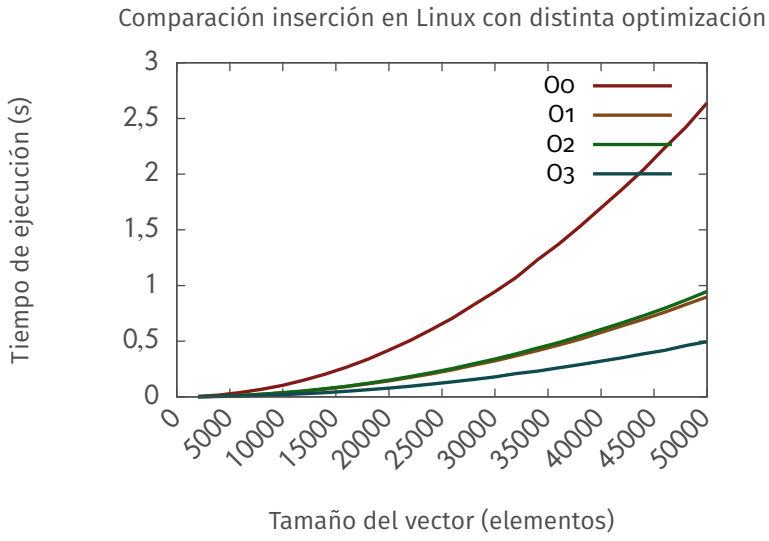
Comparativa según optimización y sistema operativo

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

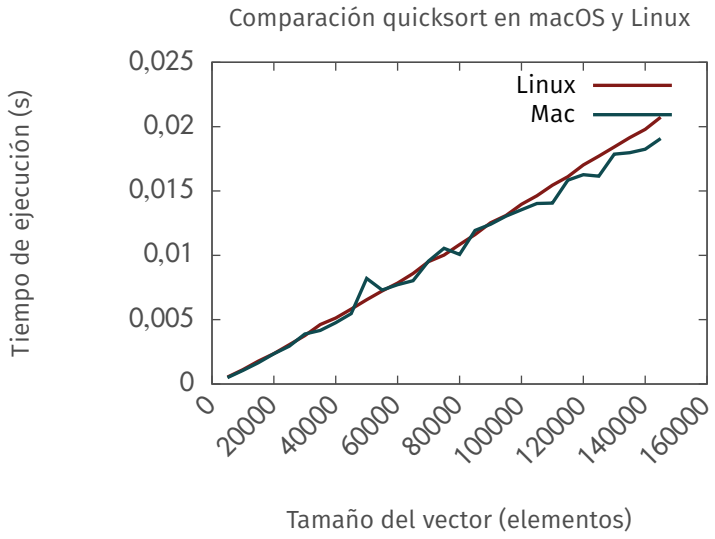
Para cada algoritmo veremos dos gráficas: una que compara la ejecución sin optimización en los sistemas operativos macOS y Linux; y otra que compara la ejecución en Linux con los distintos tipos de ejecución.

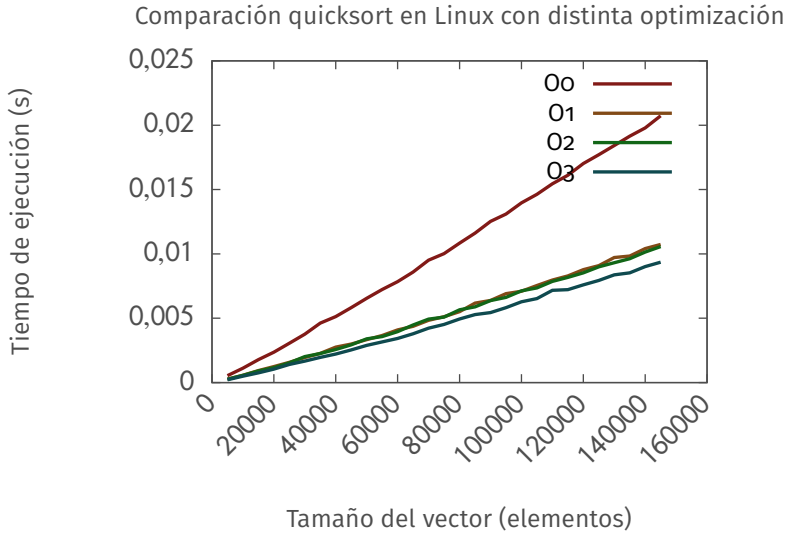
Representante de $O(n^2)$



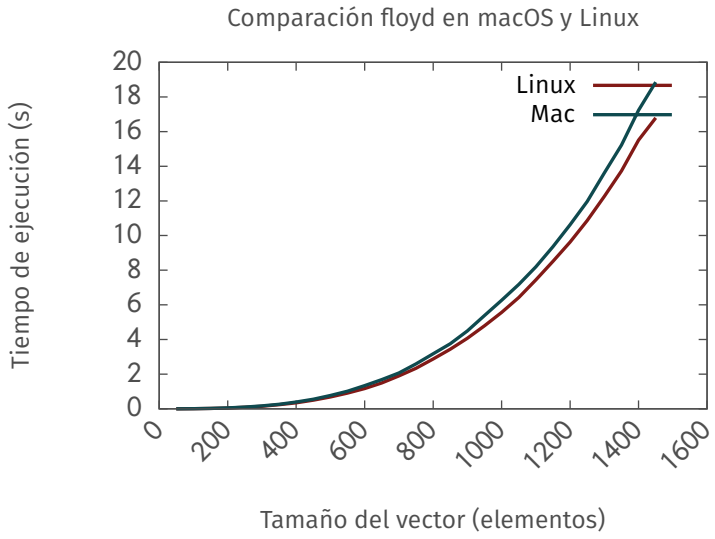


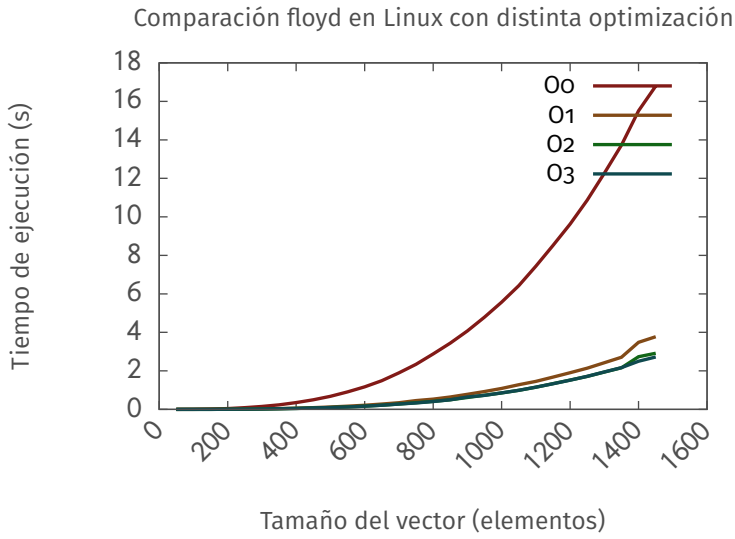
Representante de $O(n \log n)$



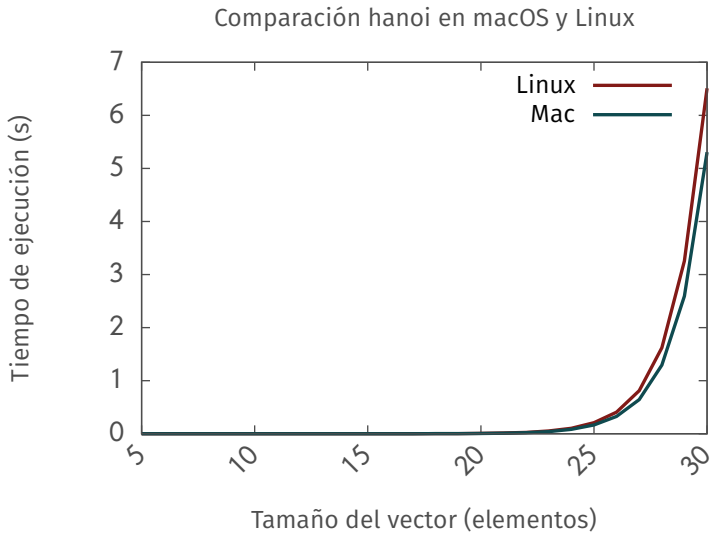


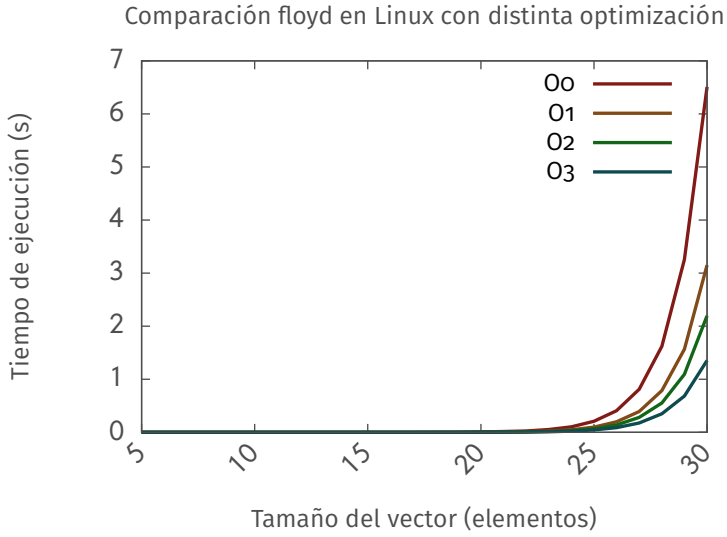
Representante de $O(n^3)$





Representante de $O(2^n)$





Anexo

Características de los ordenadores donde se ha compilado