Universidad Simón Bolívar

Departamento de Computación y Tecnología de la Información

CI-2692-Laboratorio de Algoritmos y Estructuras II

Estudio experimental de algoritmos de ordenamiento

Rubmary Rojas 13-11264

Juan Ortiz 13-11021

Febrero-2016

1 Introducción

Se realizó un estudio experimental sobre algoritmos de ordenamiento, comparando *heapsort* y algunas variantes de de *quicksort*. Para cada uno de los algoritmos se hicieron pruebas con arreglos de diferentes tamaños con los conjuntos numéricos especificados. También se hicieron unas pruebas adicionales con un par de algoritmos modificados.

En la sección 2 se especifican las características del equipo utilizado. En la sección 3 se presenta, para cada uno de los conjuntos pedidos, una tabla con los resultados obtenidos y la gráfica respectiva, que muestra la tendencia del tiempo de ejecución de cada uno de los algoritmos. La sección 4 contiene un análisis de los resultados obtenidos.

En la sección 5 se propone una modificación para los algoritmos que presentan un comportamiento poco deseado para algunos casos, explicando el motivo de dicha modificación. Y en la última sección se presentan las gráficas respectivas de los resultados experimentales de los algoritmos modificados.

2 Tipo de equipo:

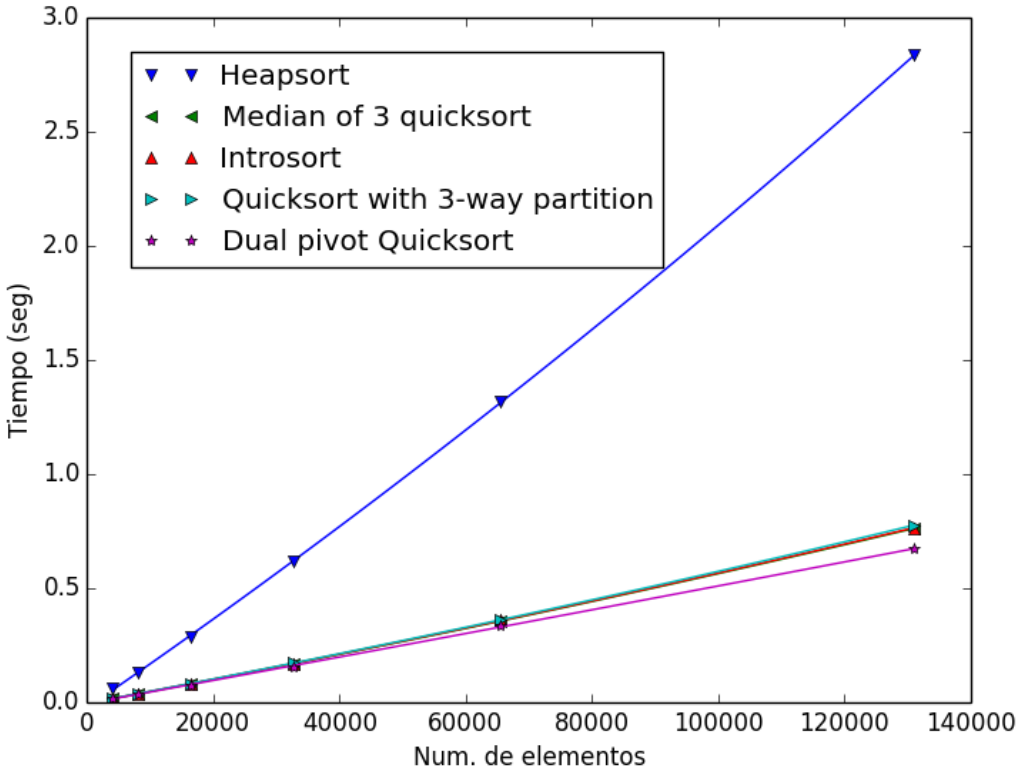
- Modelo de CPU: Intel Core i5-3337U CPU @ 1.80GHz x 4
- Cantidad de memoria RAM: 3,7 GiB
- Sistema Operativo: Ubuntu 14.04 LTS

3 Resultados:

3.1 Punto Flotante

N	Heapsort	Med-of-3 QS	Introsort	Dual pivot QS	QS with 3-way
4096	0.06234	0.01840	0.01848	0.01626	0.01882
8192	0.12952	0.03765	0.03965	0.03835	0.03882
16384	0.28543	0.07934	0.07973	0.07717	0.08172
32768	0.61882	0.16783	0.16878	0.15401	0.17192
65536	1.31733	0.35840	0.36067	0.33537	0.36297
131072	2.83552	0.76226	0.76555	0.67330	0.77701

Tabla 1: Punto Flotante

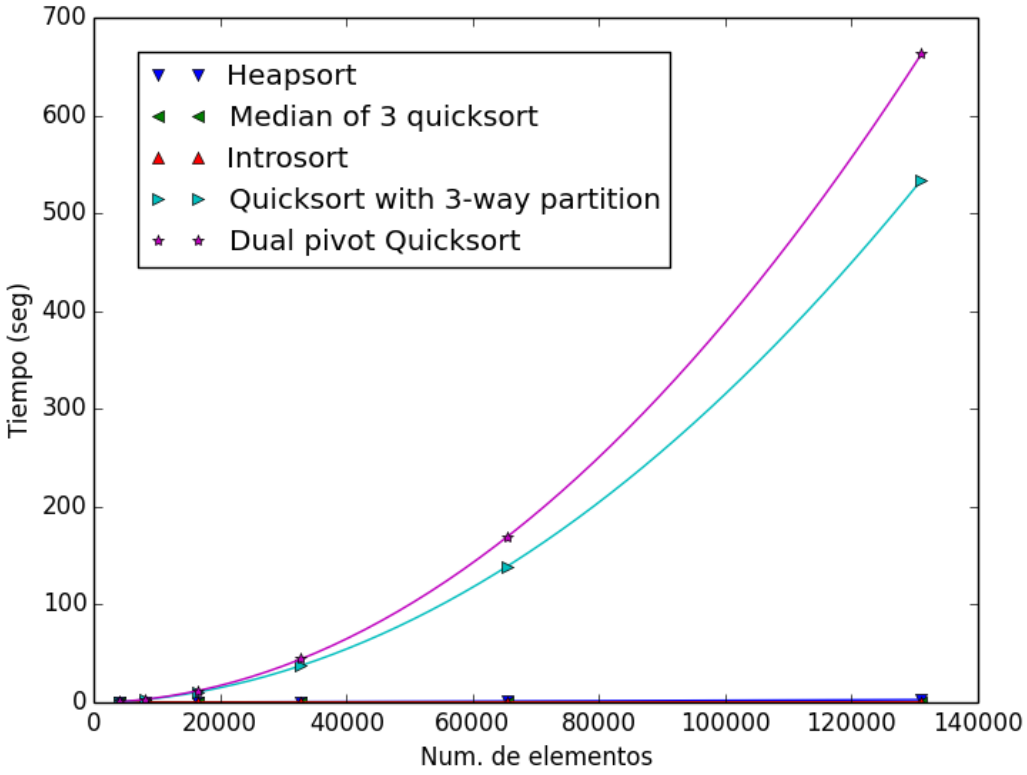


Gráfica 1: Punto Flotante

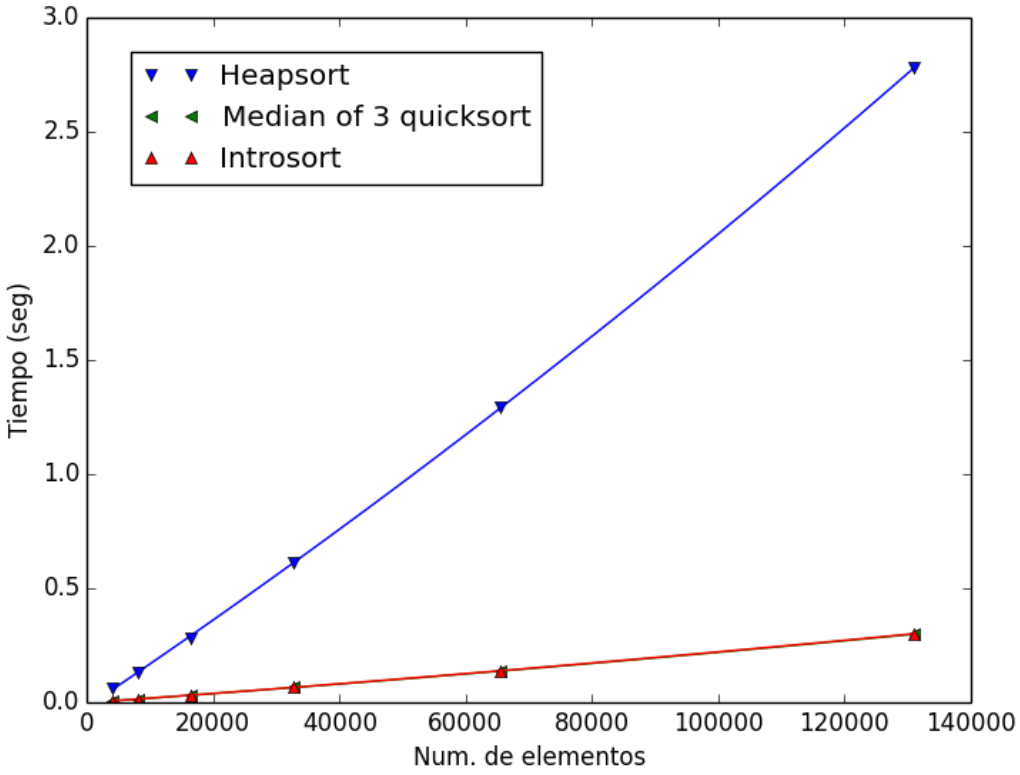
3.2 Ordenado

N	Heapsort	Med-of-3 QS	Introsort	Dual pivot QS	QS with 3-way
4096	0.06187	0.00664	0.00728	0.80946	0.68602
8192	0.13327	0.01458	0.01455	2.96090	2.601132
16384	0.28345	0.03092	0.03119	11.59523	10.20066
32768	0.61146	0.06592	0.06668	44.79973	38.08166
65536	1.29309	0.13694	0.13849	169.14331	139.00569
131072	2.78129	0.29904	0.30109	663.03806	534.22607

Tabla 2: Ordenado



Gráfica 2: Ordenado

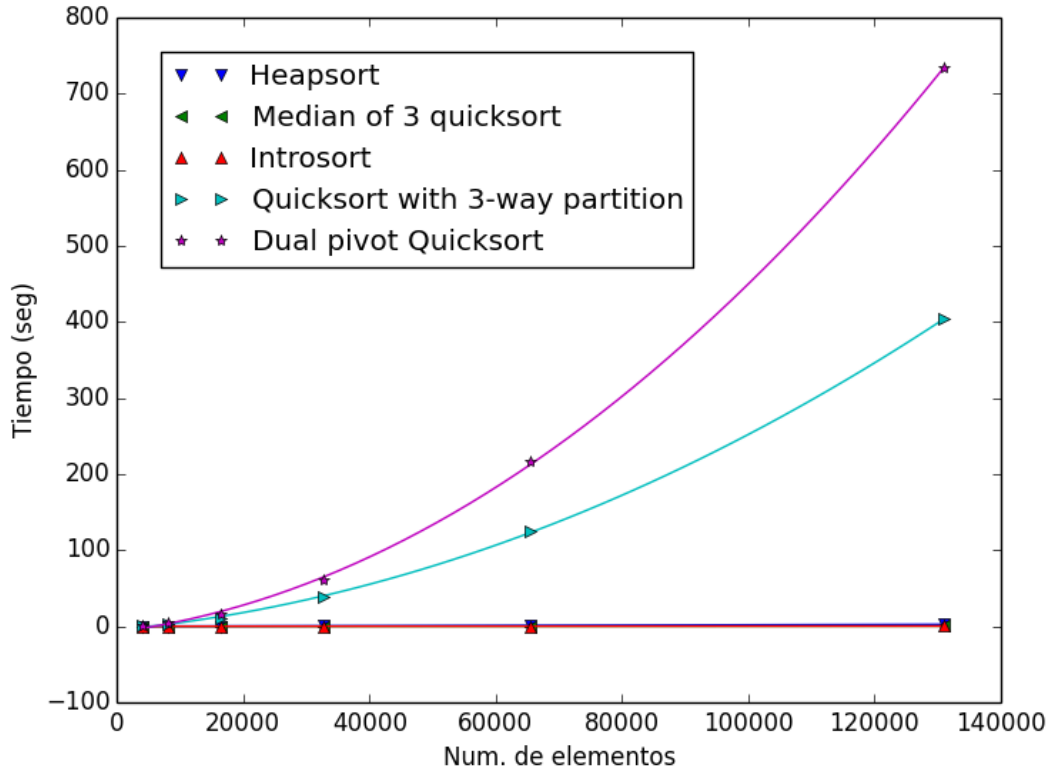


Gráfica 3: Ordenado (sin los tiempos cuadráticos)

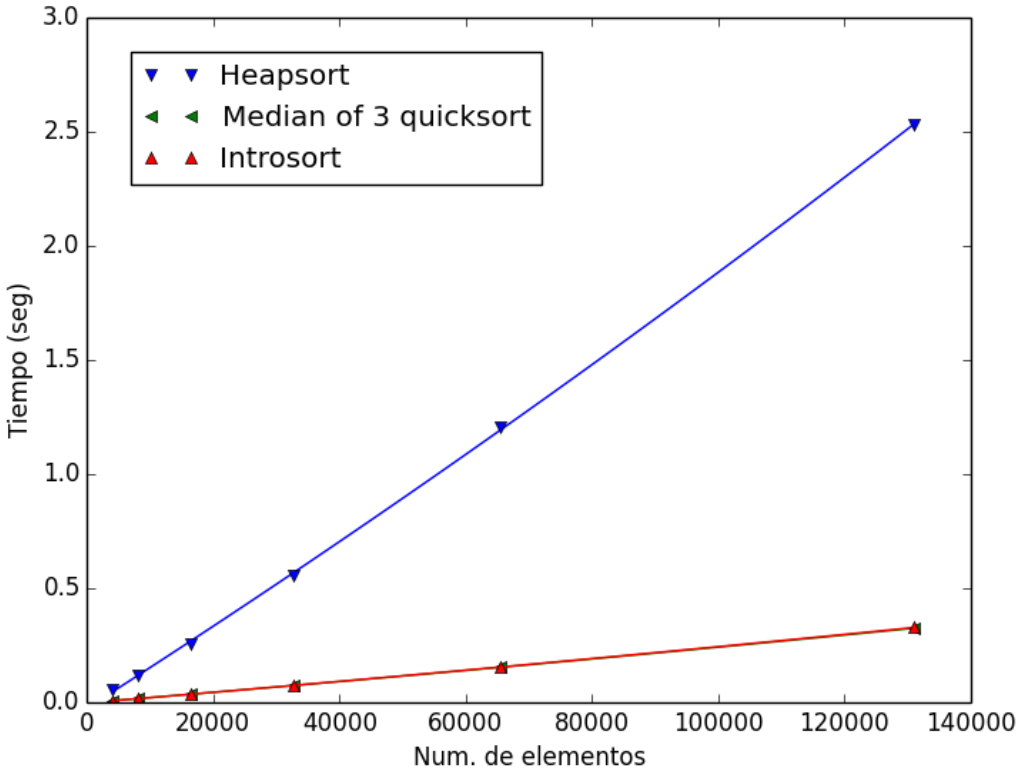
3.3 Orden inverso

N	Heapsort	Med-of-3 QS	Introsort	Dual pivot QS	QS with 3-way
4096	0.05550	0.00804	0.00791	1.05548	0.74969
8192	0.12003	0.01607	0.01650	4.13607	2.87758
16384	0.25904	0.03621	0.03575	16.14384	10.70392
32768	0.55836	0.07372	0.07324	60.33847	38.13866
65536	1.20339	0.15399	0.15596	216.94025	125.35005
131072	2.53364	0.32524	0.328250	734.53780	404.07549

Tabla 3: Orden inverso



Gráfica 4: Orden Inverso

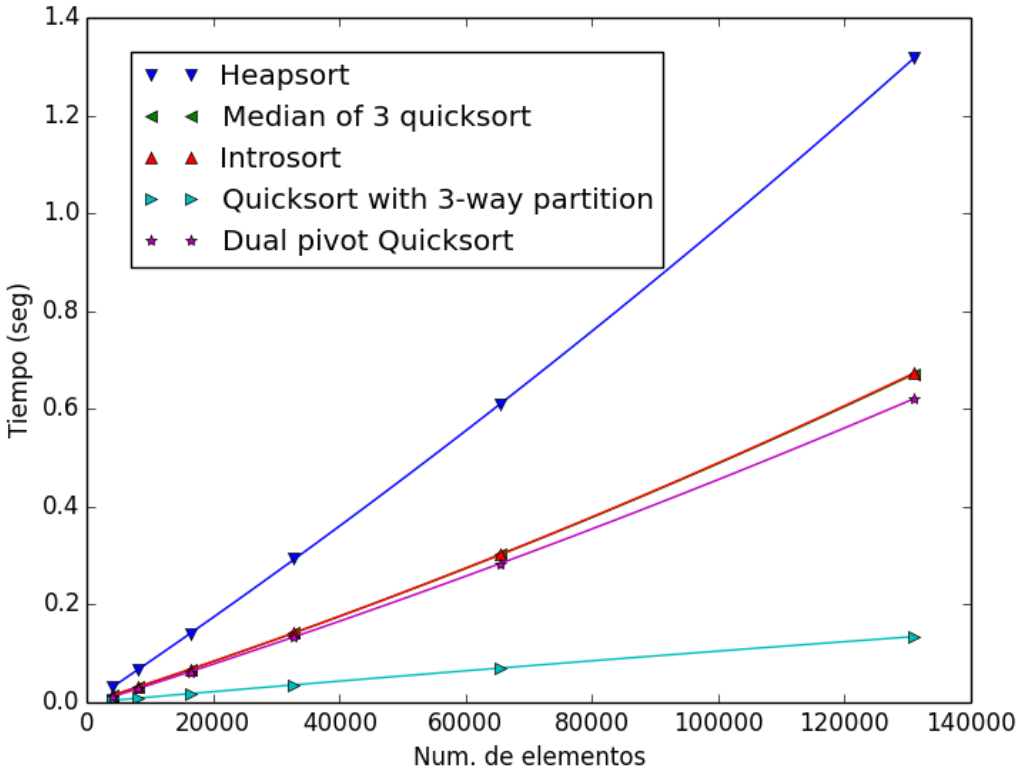


Gráfica 5: Orden Inverso (sin los tiempos cuadráticos)

3.4 Cero-uno

N	Heapsort	Med-of-3 QS	Introsort	Dual pivot QS	QS with 3-way
4096	0.03107	0.01478	0.01460	0.01249	0.00448
8192	0.06715	0.03234	0.03192	0.02830	0.00904
16384	0.13961	0.06668	0.06721	0.06070	0.01777
32768	0.29569	0.14178	0.14259	0.13517	0.03507
65536	0.60833	0.30230	0.30248	0.28387	0.07078
131072	1.31843	0.67157	0.67373	0.62137	0.13448

Tabla 4: Cero-uno

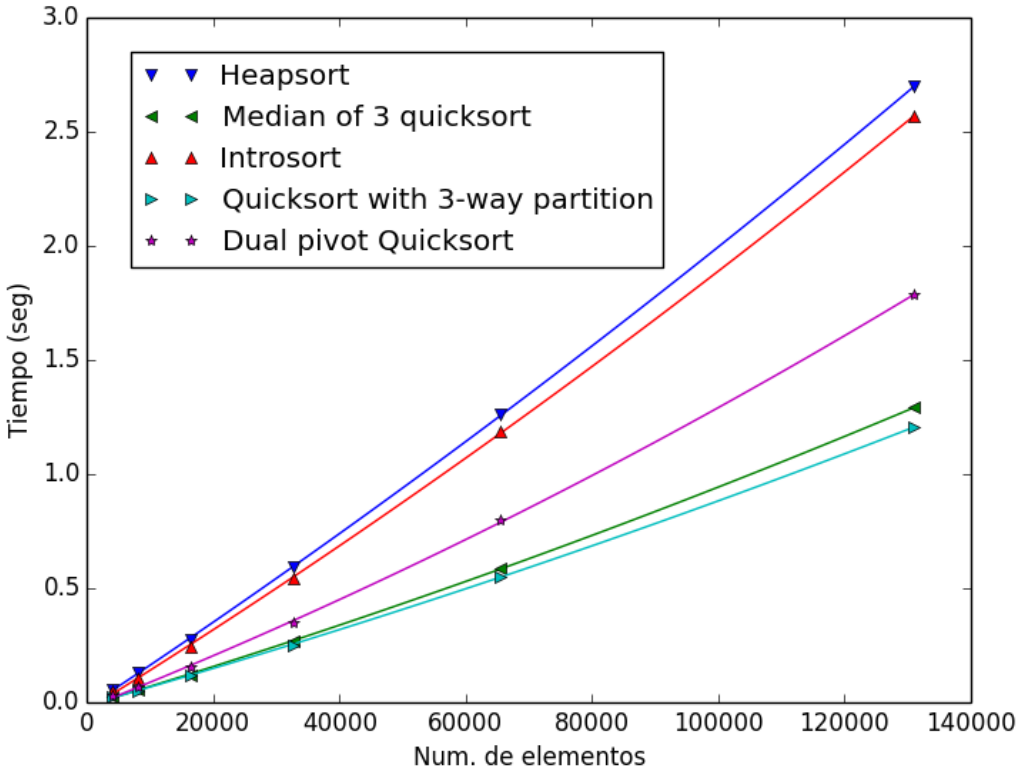


Gráfica 6: Cero-uno

3.5 Mitad

N	Heapsort	Med-of-3 QS	Introsort	Dual pivot QS	QS with 3-way
4096	0.05891	0.02455	0.04757	0.02927	0.02401
8192	0.12851	0.05488	0.10785	0.06744	0.05192
16384	0.27650	0.12063	0.24379	0.15450	0.11598
32768	0.59124	0.26633	0.54126	0.34995	0.25208
65536	1.26245	0.58957	1.18898	0.79782	0.55267
131072	2.69907	1.29148	2.56980	1.78665	1.20622

Tabla 5: Mitad

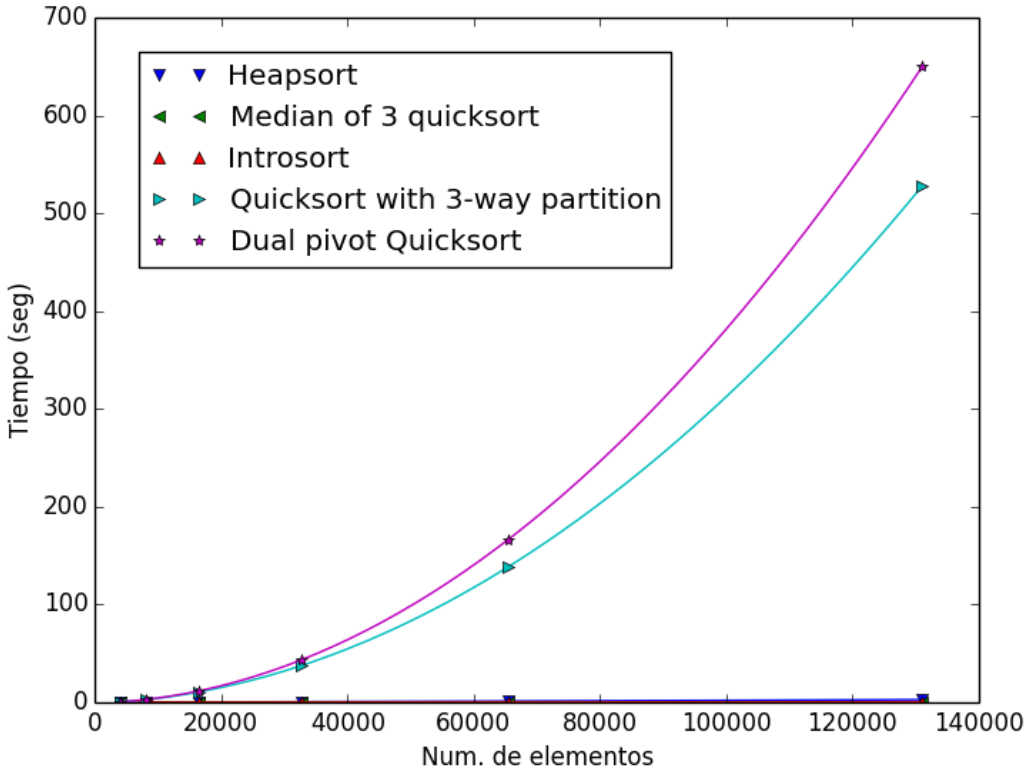


Gráfica 7: Mitad

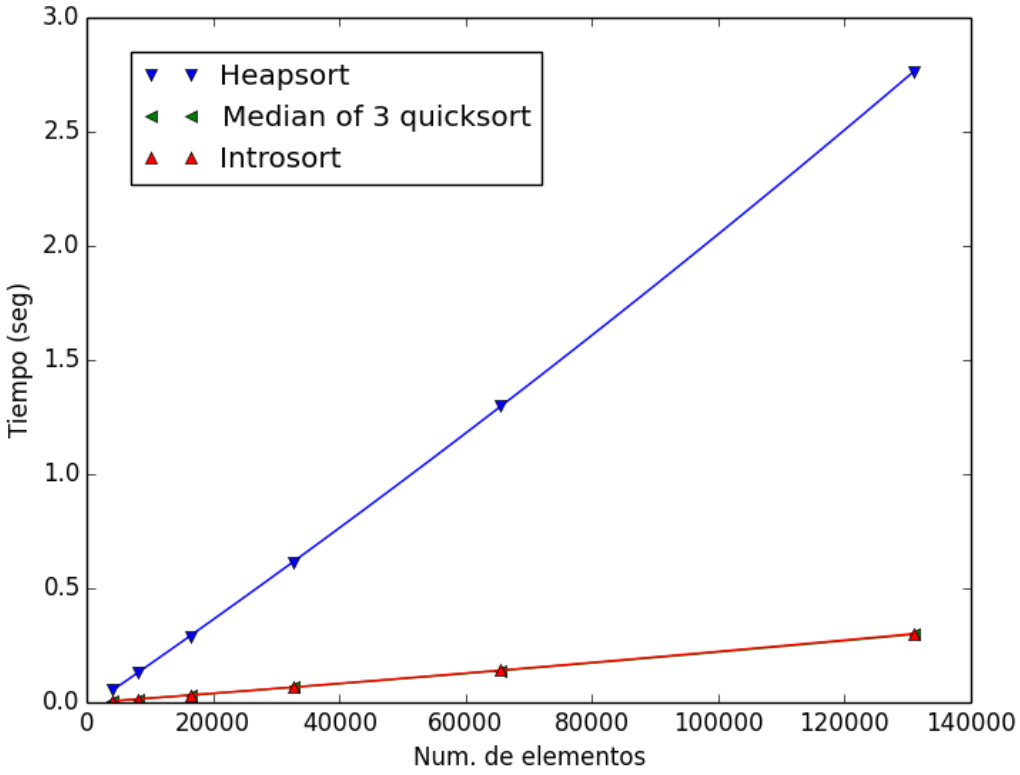
3.6 Casi ordenado 1

N	Heapsort	Med-of-3 QS	Introsort	Dual pivot QS	QS with 3-way
4096	0.05915	0.00640	0.00629	0.70191	0.64802
8192	0.13221	0.01417	0.01463	2.86071	2.60423
16384	0.28573	0.03217	0.03188	11.27212	10.08263
32768	0.61160	0.06661	0.06686	44.20084	38.18231
65536	1.30250	0.13911	0.14110	166.40286	138.62382
131072	2.76558	0.29839	0.30134	528.22088	650.33827

Tabla 6: Casi ordenado 1



Gráfica 8: Casi Ordenado 1

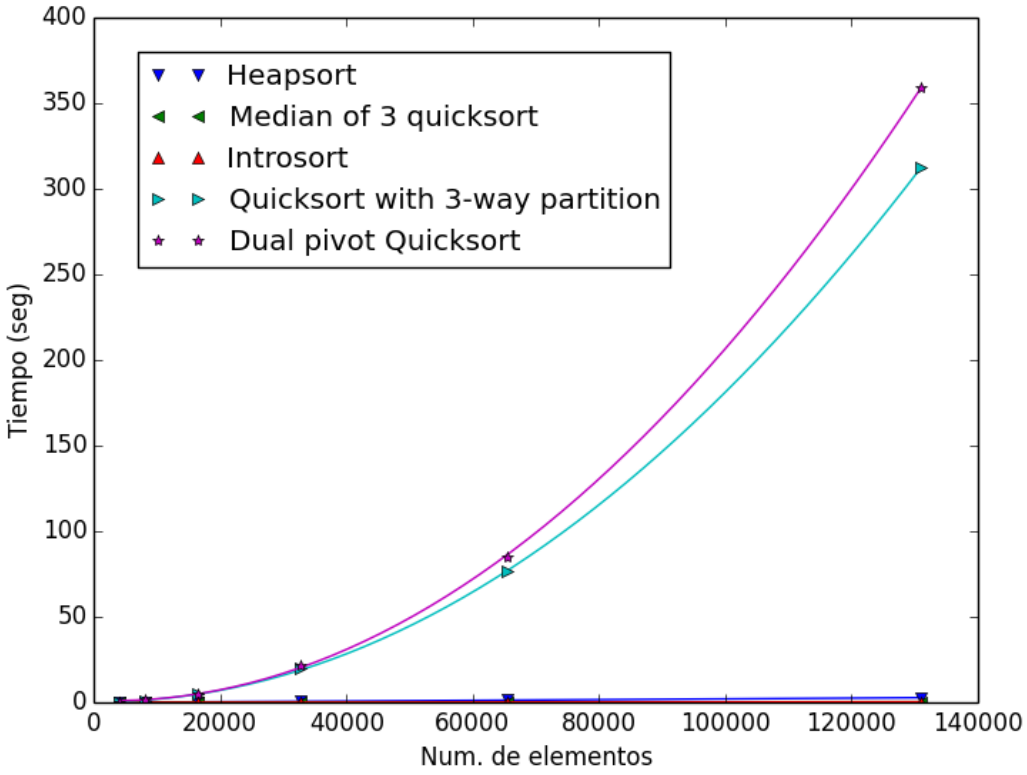


Gráfica 9: Casi Ordenado 1 (sin los tiempos cuadráticos)

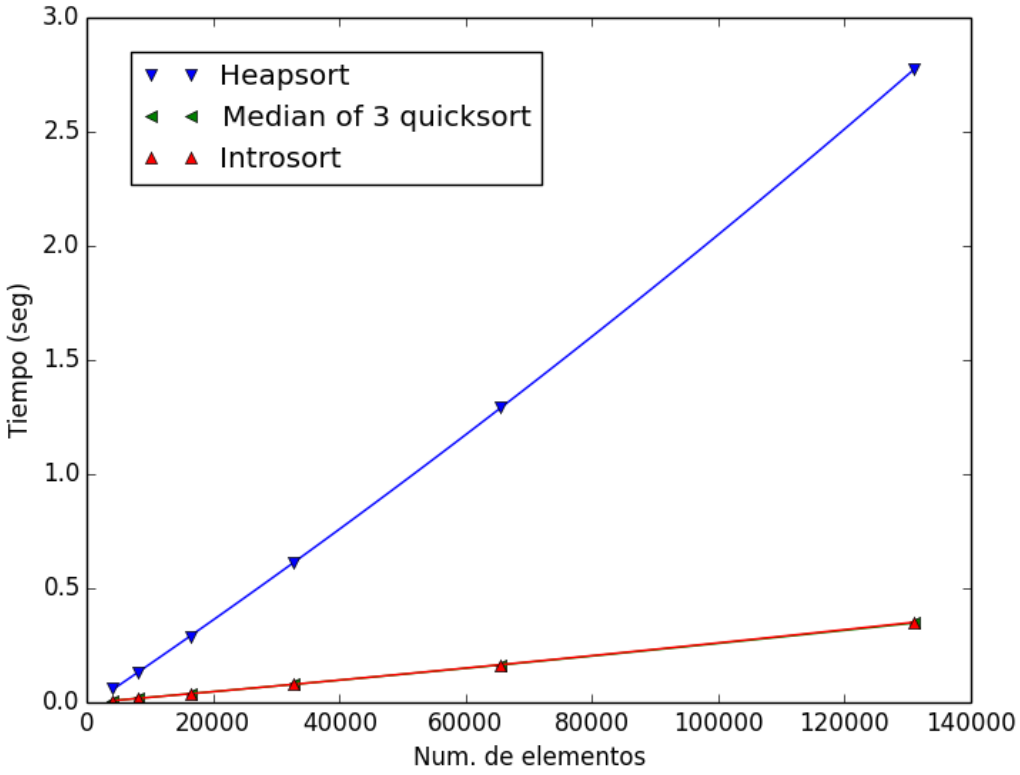
3.7 Casi ordenado 2

N	Heapsort	Med-of-3 QS	Introsort	Dual pivot QS	QS with 3-way
4096	0.05975	0.00777	0.00820	0.30293	0.34659
8192	0.13366	0.01788	0.01678	1.39142	1.23952
16384	0.28519	0.03801	0.03764	5.36986	4.86774
32768	0.61030	0.08017	0.08007	21.71385	19.77836
65536	1.29329	0.16330	0.16496	76.43386	85.32135
131072	2.77425	0.34830	0.35157	359.56906	312.83785

Tabla 7: Casi ordenado 2



Gráfica 10: Casi Ordenado 2



Gráfica 11: Casi Ordenado 2 (sin los tiempos cuadráticos)

4 Análisis

Los algoritmos *Quicksort with 3-way partition*, y *Dual pivot Quicksort* presenta un crecimiento cuadrático para los conjuntos **Ordenado**, **Orden inverso**, **Casi Ordenado 1** y **Casi Ordenado 2**, siendo *Dual pivot Quicksort*, el más ineficiente de los dos. Esto ocurre debido a que las particiones son, siempre (o casi siempre) muy desbalanceadas, alcanzando tiempos de ejecución muy altos para arreglos que estén ordenados (o casi ordenados), bien sea de forma creciente o decreciente. El algoritmo *median of 3 quicksort*, evita este peor caso al tomar la mediana de 3 elementos, por lo que la partición nunca será completamete desbalanceada. Por otra parte, el peor caso de introsort siempre será $\mathcal{O}(n \log n)$ debido a que se utiliza *heapsort* después de una cantidad específica de llamadas recursivas.

Si bien, asintóticamente, *heapsort* y *quicksort* (en cualquiera de sus variantes), tienen la misma complejidad, la constante escondida en el primer algoritmo es mucho mayor, lo que se puede observar en todas las gráficas, excluyendo los casos donde los dos últimos algoritmos se vuelven cuadráticos.

En el primer caso, donde los números son reales en el rango $[0, 1)$ generados de forma aleatoria, las 4 variantes de quicksort tienen un comportamiento similar, siendo *Dual Pivot Quicksort* la más eficiente. Se puede observar, como fue mencionado anteriormente, que *heapsort* es el algoritmo más lento.

El algoritmo *Quicksort with 3-way partition*, está optimizado para arreglos donde los elementos se repitan con mucha frecuencia, pues al elegir un número como pivote excluye todas sus ocurrencias en las próximas llamadas recursivas. Por esta razón, en el caso **Cero-uno**, donde se genera un conjunto formado con solo ceros y unos, este algoritmo es el más eficiente.

En general, las 4 variantes de quicksort presentan un buen rendimiento, con execpción de *Quicksort with 3-way partition* y *Dual pivot Quicksort* cuando los arreglos están casi ordenados.

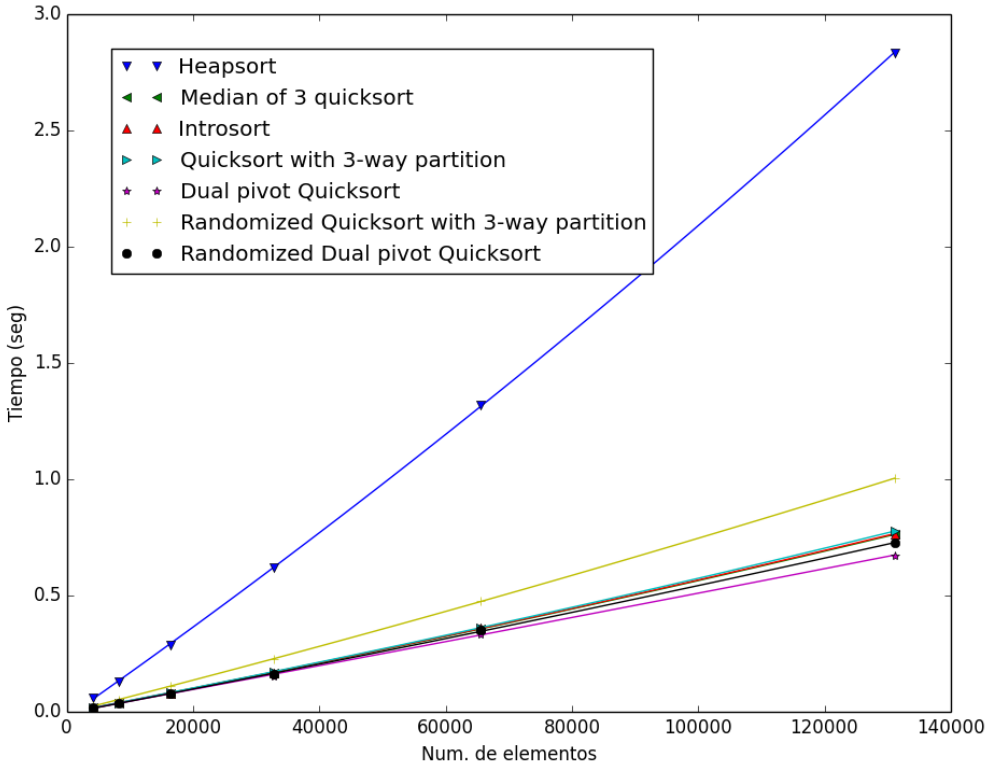
5 Eliminando peores casos

Como se pudo observar en los resultados, existen casos donde algoritmos de *quicksort*, como *Quicksort with 3-way partition* y *Dual pivot Quicksort*, el tiempo de ejecución es cuadrático respecto al tamaño del arreglo. Sin embargo, la cantidad de estos casos son muy pocas (por ejemplo, cuando el arreglo está completamente ordenado de forme creciente o decreciente), por lo tanto, una forma de eliminar estos casos consiste en randomizar el algoritmo. Las posibles formas de randomizarlo son permutando el arreglo o eligiendo el (los) pivote(s) al azar, de esta forma la probabilidad de incurrir en alguno de estos casos es muy poca, considerándose nula en la práctica.

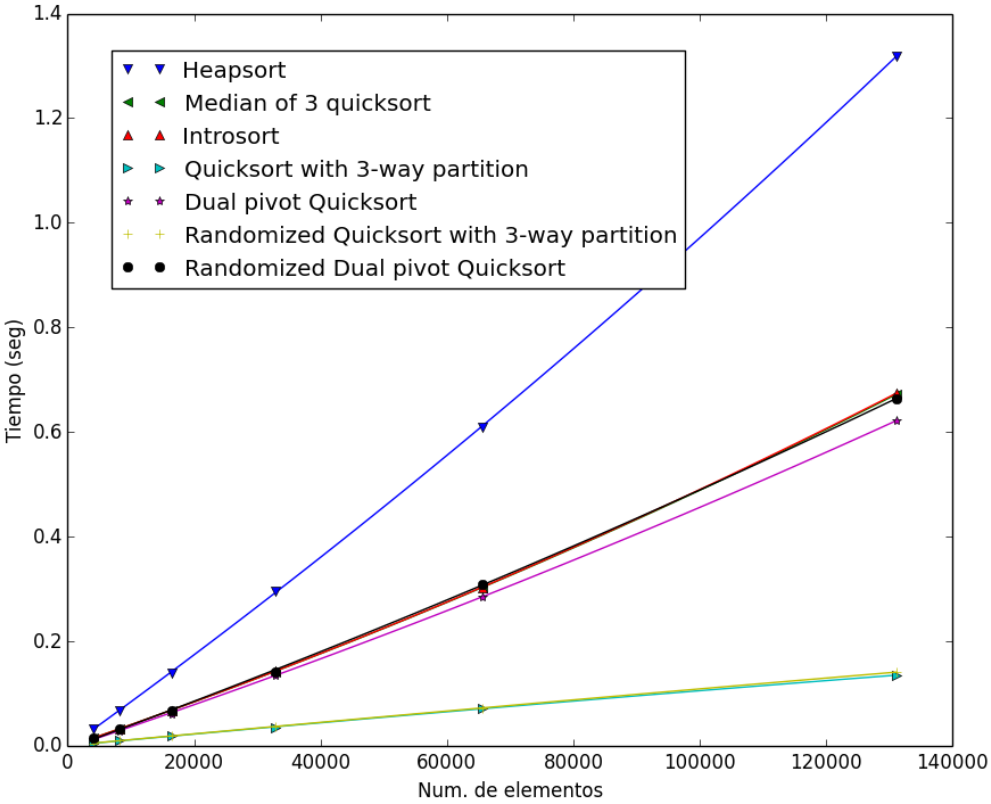
Los algoritmos *Quicksort with 3-way partition* y *Dual pivot Quicksort* fueron randomizados eligiendo el(los) pivote(s) de forma aleatoria. Esto elimina los peores casos, sin embargo, la elección del pivote aumenta un poco el tiempo de ejecución.

6 Algoritmos randomizados, gráficas

Se presentan las gráficas obtenidas al ejecutar los 7 algoritmos con los conjuntos **Punto Flotante**, **Cero-Uno**. Se puede notar que los algoritmos randomizados son menos eficientes que los originales.

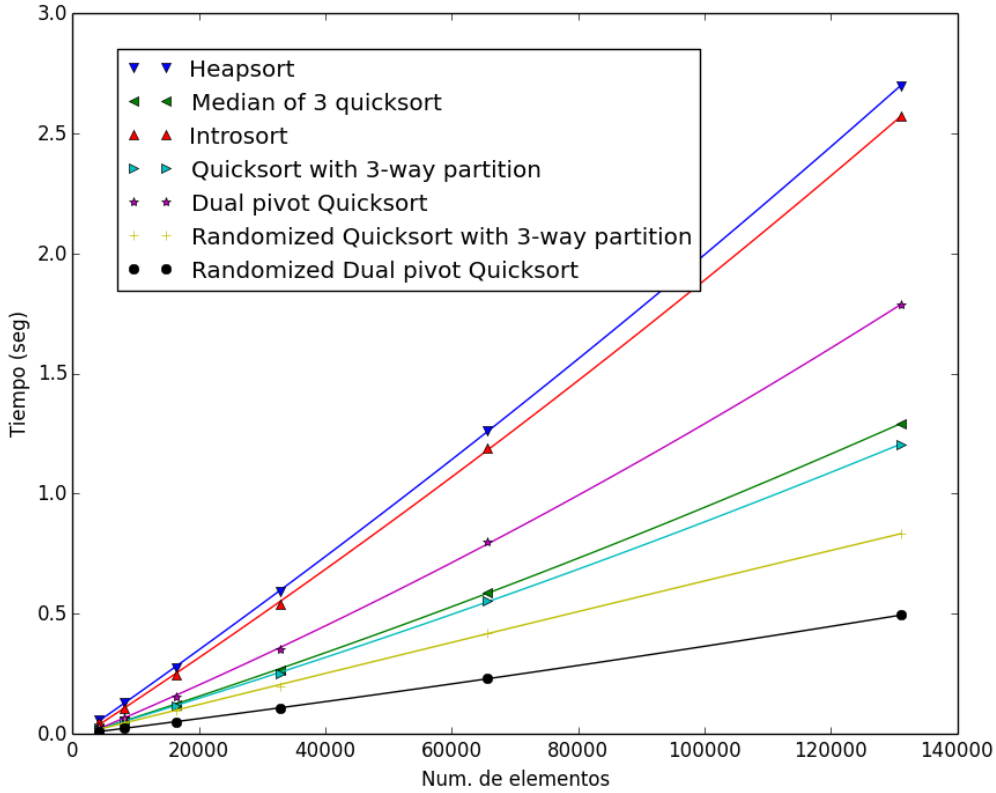


Gráfica 12: Punto Flotante



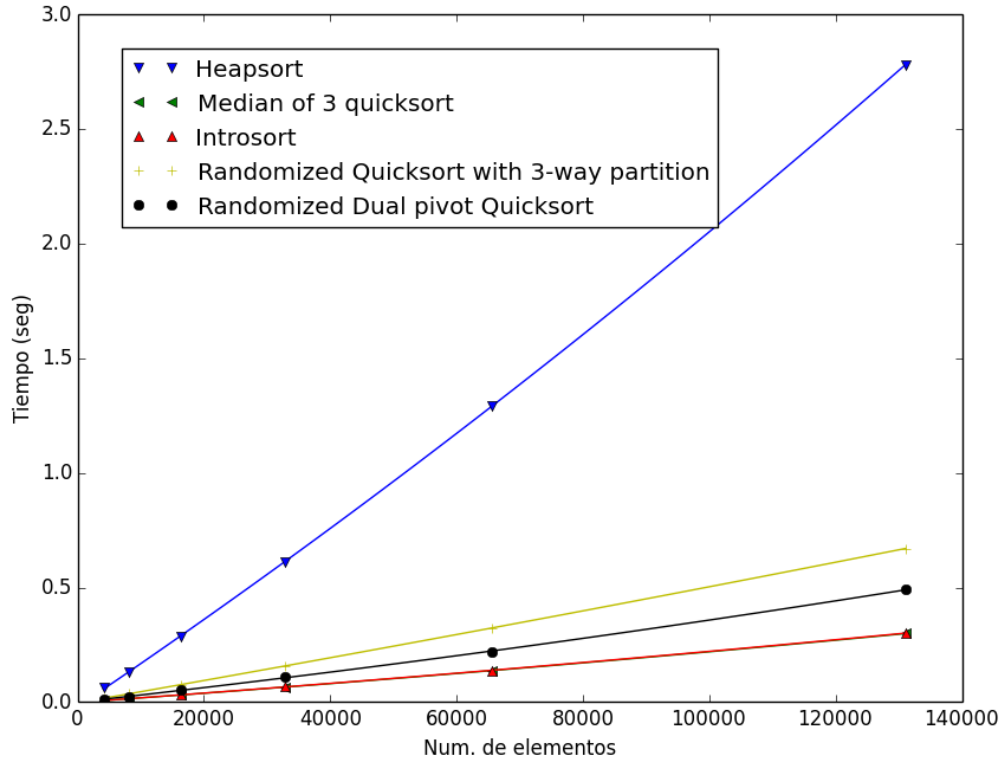
Gráfica 13: Cero-uno

Al correr los algoritmos con el conjunto mitad, se ve una mejora de los resultados, esto se debe a que el arreglo original también está pseudo ordenado

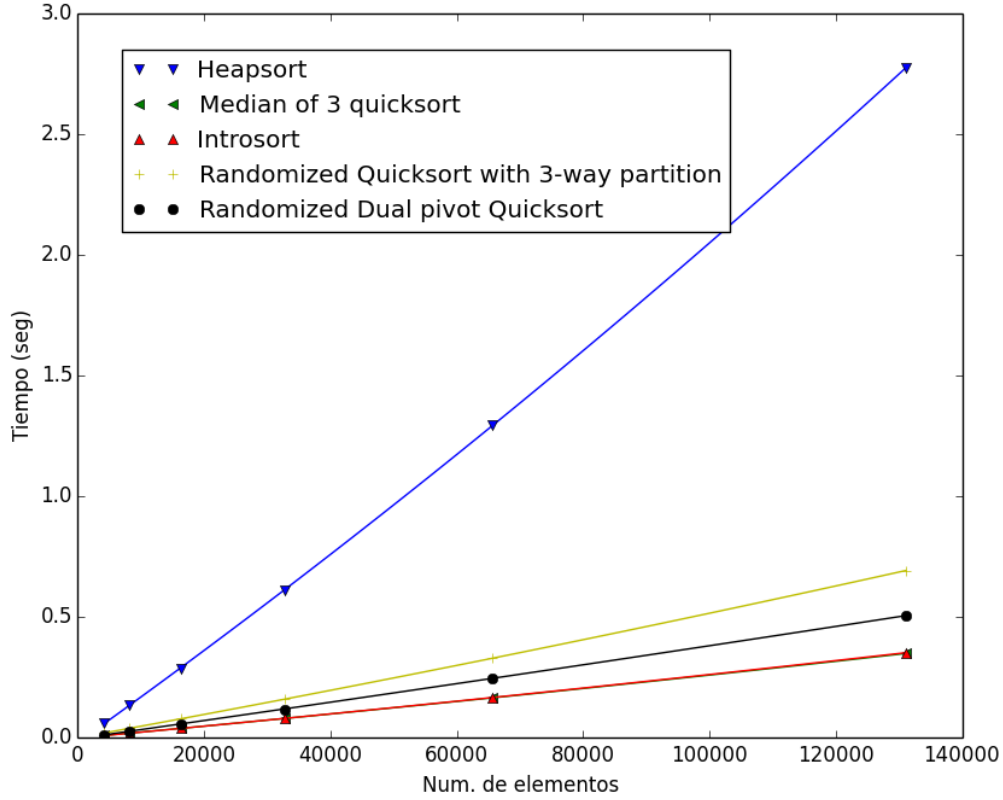


Gráfica 14: Mitad

Las gráficas siguientes, que son los resultados de correr los algoritmos con los conjuntos **Ordenado** y **Casi ordenado 2**, muestra la vetaja de randomizar los algoritmos, ya que eliminan los peores casos que ocurrían cuando el arreglo estaba casi ordenado. En estos casos los algoritmos también presentan un comportamiento cuasi-lineal, cuando los algoritmos originales presentaban un comportamiento cuadrático.



Gráfica 15: Ordenado



Gráfica 16: Casi ordenado 2