REPÚBLICA BOLIVARIANA DE VENEZUELA

UNIVERSIDAD SIMÓN BOLÍVAR

INGENIERÍA DE LA COMPUTACIÓN

LABORATORIO DE ALGORITMOS II

COMPARACIÓN DE ALGORITMOS DE ORDENAMIENTO

ESTUDIANTES:

JESÚS KAUZE #1210273

DAVID SEGURA #1311341

CARACAS, FEBRERO DE 2017

INTRODUCCIÓN

En el siguiente informe realizado con la finalidad de comparar los diferentes algoritmos de ordenamiento, se contempla el análisis de los mismos al ser ejecutados con diferentes cantidades de elementos y cuyo rendimiento será evaluado con el tiempo de procesamiento.

Los algoritmos a evaluar son los vistos hasta ahora en la materia de Algoritmos 2, los cuáles son: Mergesort, Heapsort, Introsort y las diferentes variaciones del Quicksort, entre las más importantes se encuentran la básica y la randomizada.

Para las pruebas de dichos algoritmos se realizó una corrida con las siguientes cantidades de elementos: 4096, 8192, 16384, 32768 y 65536, a su vez evaluada con diferentes tipos de arreglos, los cuales son:

1) **Enteros Aleatorios**: arreglo cuyos elementos son números enteros aleatorios comprendidos entre 0 y 500.
2) **Orden Inverso**: es un arreglo que está ordenado de manera descendente, como por ejemplo si el tamaño del arreglo es 10, entonces tendrá los números enteros ordenado de la siguiente manera: [9,8,7,6,5,4,3,2,1,0].
3) **Ceros-unos**: es un arreglo cuyos elementos sólo serán los números 0 y 1. Ejemplo: [0,0,1,1,1,1,0,0,1,0,1,0].
4) **Ordenado**: si el arreglo tiene 10 elementos, este estará ordenado de manera ascendente.
5) **Reales Aleatorios**: el arreglo obtendrá números reales comprendidos entre 0 y 1 de manera aleatorio. Por ejemplo: [0.8478,0.0002,0.12,0.9994].
6) **Mitad**: es un arreglo que tendrá a los números enteros ordenados de manera ascendente hasta la mitad, y luego la otra mitad estará ordenada de manera descendente. Ejemplo: si el arreglo es de tamaño 10, este será: [0,1,2,3,4,4,3,2,1,0].
7) **Casi Ordenado**: dado un conjunto ordenado de N elementos de tipo entero, se escogen al azar n/4 pares de elementos que se encuentran separados 4 lugares, entonces se intercambian los pares.

Además como elemento importante y fundamental para el tiempo de ejecución de cada prueba se debe considerar la especificaciones de las máquinas sobre la cual se realizaron:

Información de Sistema Operativo: GNU/LINUX Ubuntu 16.04

```
Ubuntu 16.04.1 LTS \n \l
```

Información del CPU: Intel core i7 - 3770 - 3.40Ghz

```
processor       : 7
vendor_id       : GenuineIntel
cpu family      : 6
model           : 58
model name      : Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz
stepping        : 9
microcode       : 0x1c
cpu MHz         : 1599.992
cache size      : 8192 KB
physical id     : 0
siblings        : 8
core id         : 3
cpu cores       : 4
apicid          : 7
initial apicid  : 7
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
```

Información de la memoria RAM: 8 Gb de memoria RAM ddr3

```
MemTotal:        8138224 kB
MemFree:         7519924 kB
MemAvailable:    7671148 kB
Buffers:           52440 kB
Cached:           321620 kB
SwapCached:            0 kB
Active:           222708 kB
Inactive:         268852 kB
```

Ahora bien, una vez conocida las especificaciones principales que afectan el tiempo final de cada ordenamiento ordenamiento, se dará una *breve* introducción al funcionamiento de los distintos algoritmos ya mencionados anteriormente en el head de la página 1.

Heap_Sort: Algoritmo basado en comparaciones tal que construye un heap (montículo) binario a partir del arreglo N para luego verificar que cumplan la propiedad de max-heap, es decir, el padre de cada i-nodo debe ser mayor que sus dos hijos, para luego ordenarlos en un arreglos de N elementos “In-place” en un tiempo de O(nlogn).

Merge_Sort: El Mergesort es un algoritmo basado en comparaciones, se basa en la técnica divide y vencerás. tiene una complejidad de Θ(nLogn) en el peor de los casos así como los tiempos Promedios. Su codigo se basa en dividir un arreglo de N elementos en dos sub-arreglos de N/2 cada uno hasta llegar varios sub-arreglos mínimos. Luego procede a ordenar de forma

recursiva cada sub-arreglo generado para luego combinarlos en una secuencia ordenada.

Quick_Sort_Aleatorio: Es un algoritmo de ordenamiento que en el peor de los casos tiene un tiempo de corrida de Θ(n^2) mientras que en el caso promedio tiende a Θ(nLogn). Al igual que MergeSort es un algoritmo in place y usa la técnica de divide y vencerás, ya que utiliza un pivote para la división del arreglo en otros subarreglos, comparando qué elementos son mayores y menores que él, en este caso pivote es un elemento aleatorio del arreglo. Luego de tener los diferentes subarreglos, se encarga de ordenar y combinar en una nueva secuencia ordenada. Los peores casos ocurren cuando los subarreglos son desbalanceados (hay mucha desproporción entre los sub-arreglos iniciales).

Median_of_3_Quick_Sort: Utiliza la misma que el Quick_Sort, a diferencia que toma como pivote la mediana entre el primer, medio y último elemento del arreglo de Tamaño N, como lo recomienda Sedgewick. De esta manera se estaría disminuyendo la constante del peor de los casos del Quick_Sort para tener un tiempo promedio de Θ(nLogn).

Intro_Sort: Es una versión basada del Quicksort, que limita la profundidad de la recursión, es decir cuando exceda este límite el realiza un llamado a Heap_Sort, lo cual hace que tenga un tiempo de ejecución en el peor de los casos de O(nLogn).

Quick_Sort_3_Way: Es otra de las versiones basadas en el partition de Quick_Sort. Este añade la peculiaridad de que él divide el partition, es decir, toma una parte que son para los elementos menores al pivote, otra para los mayores al pivote y una parte haciendo referencia a los elementos repetidos.

Dual_Pivot_Quick_Sort: En 2009 Vladimir Yaroslavskiy propuso a la lista de correo de los desarrolladores de las librerías JAVA 4, una variante de Quicksort la cual usa dos pivotes. Este algoritmo fue escogido para ser el algoritmo de ordenamiento de arreglos por defecto de la librería de JAVA 7 de Oracle, después de un exhaustivo estudio experimental. También este algoritmo forma parte de la librerías de programación de la plataforma Android de Google. Wild estudiaron esta versión de Quicksort y encontraron que su buen rendimiento se debe en buena parte a su método de particionamiento que hace que el algoritmo tenga en promedio menos comparaciones, que otras versiones de Quicksort. Su tiempo promedio de ejecución es Θ(nLogn)

RESULTADOS

A continuación se procede con los resultados de cada prueba, mostrando así sus tiempos junto con sus gráficas. En caso de existir una gráfica con un algoritmo de tiempo cuadrático que opaque los otros, se incorporará una segunda gráfica en la que se pueda estudiar la gráfica de los otros algoritmos.

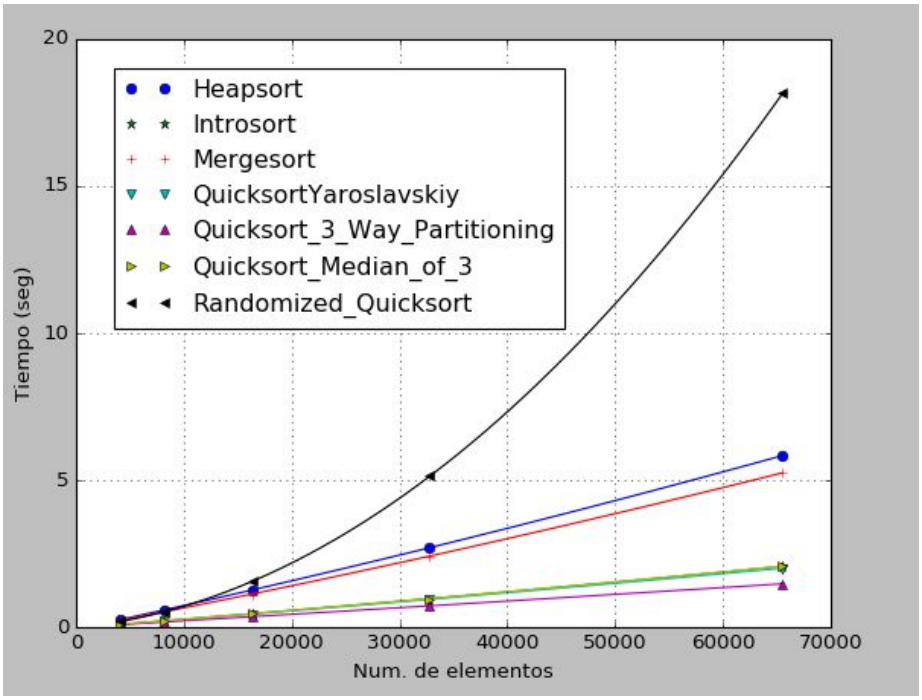
PRUEBA 1

*Tabla de resultados de la prueba 1 (**Enteros Aleatorios)***

Prueba 1

N	Mergeso	Heapsort	QS Random	Med-of-3 Q	Introsort	Dual Pivot	QS 3-way
4096	0.237	0.264	0.19	0.098	0.096	0.085	0.08
8192	0.514	0.577	0.53	0.21	0.21	0.199	0.18
16384	1.119	1.263	1.55	0.481	0.48	0.434	0.35
32768	2.421	2.704	5.13	0.957	0.959	0.946	0.72
65536	5.247	5.836	18.16	2.079	2.069	2.001	1.47

*Gráfica de resultados de la prueba 1 (**Enteros Aleatorios)***



Analisis:

El resultado de la primera prueba **Enteros Aleatorios** Se puede analizar a simple vista un Quicksort Randomizado con tendencia de tiempo N^2 siendo N el tamaño del arreglo. Como se puede apreciar en la tabla de resultados, en las últimas pruebas demora 18,16 segundos en ordenar más de 60.000 elementos y 5,13 en ordenar más de 30.000 elemento mientras que por otro lado el más eficiente de esta prueba viene siendo el QuickSort_3_way demorando 1,47 con más de 60.000 elementos y 0,72 con más de 30.000.

En el caso del Quicksort Randomizado se debe a la escogencia aleatoria del pivote, pudiendo dar un caso poco eficiente por las probabilidades que tiene de la escogencia de un pivote que no genere un split Desbalanceado. Mientras que el Quicksort_3_way al dividir la partición en 3 (los menores al pivote, el pivote con los no verificado que cumplan que no quiebren la condición y los mayores al pivote.

El heapsort junto con el mergesort tienen una eficiencia parecida. en el caso del heap porque los últimos elementos (los hijos) vienen siendo siempre mayor que su padre, ya que el arreglo está ordenado. y en el caso del merge se debe a que el sigue verificando el orden independiente de que estén ordenados inicialmente (recordando que el los divide hasta que sean sub-arreglos mínimos y los comprueba).

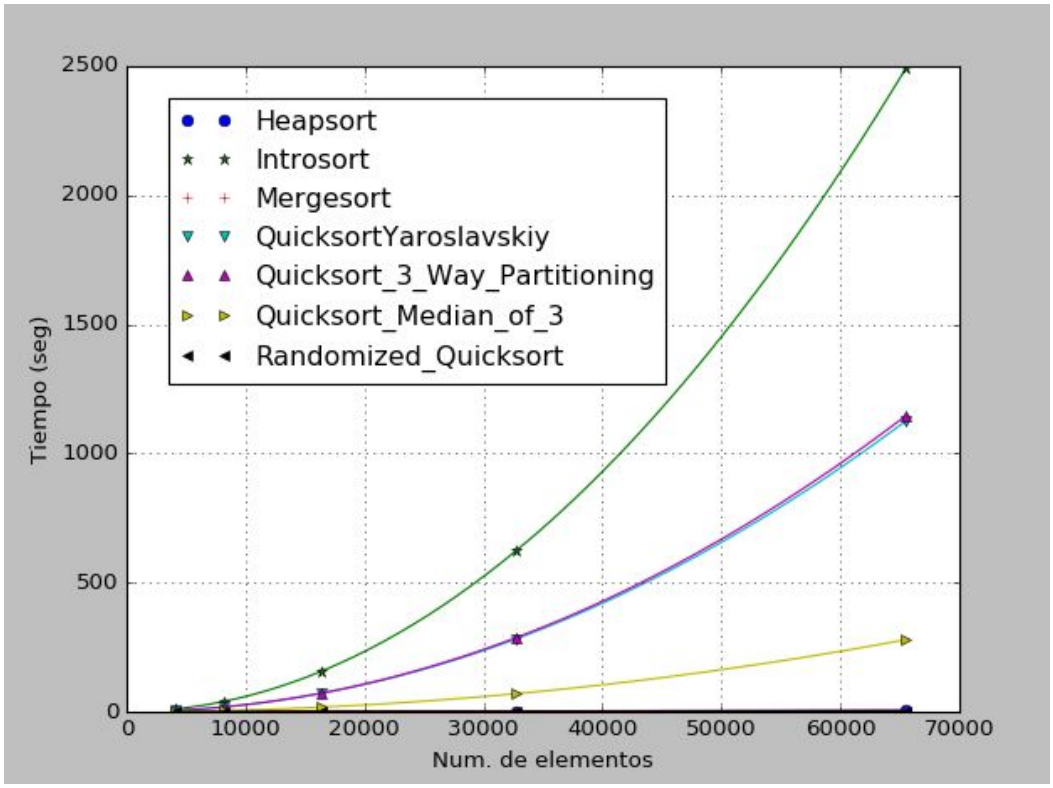
Los mas eficiente junto con el Quicksor_3_way fueron el Quicksort_median_of_3 y el Dual_pivot_Quicksort. El median of 3 prácticamente toma el elemento del medio ocasionando que las particiones sean balanceadas, lo cual tiende a optar por el tiempo NlogN en el mejor de sus casos. El dual pivot al tomar 2 pivotes y tener un arreglo ordenado es capaz de analizar y ordenar rápidamente los elementos, esto es a causa de que el primer pivote verificará los elementos menores a él mientras que el segundo comprobará los mayores a el, quedando en el medio los elementos que están entre ellos dos. Al tener el arreglo ordenado no será necesario intercambiar los elementos porque ya están en donde tienen que estar.

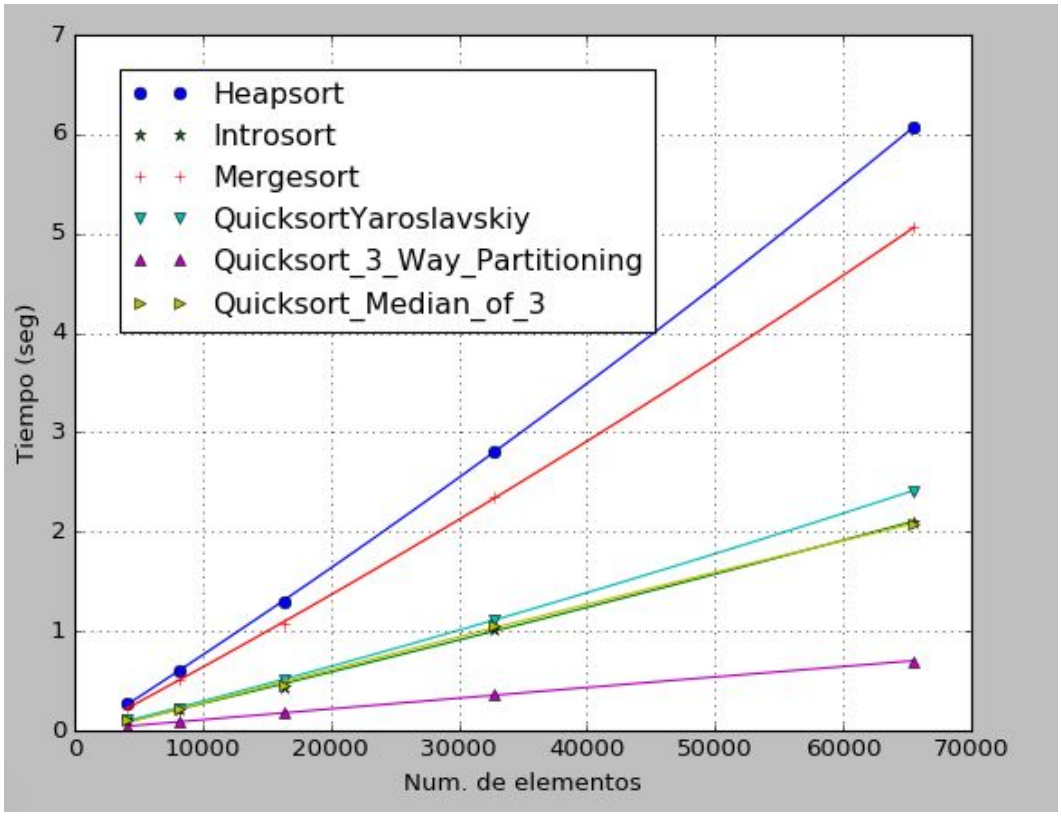
PRUEBA 2

*Tabla de resultados de la prueba 2 (**Orden Inverso**)*

	N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual Pivot QS	QS 3-way
Prueba 2	4096	0,214	0,236	0,151	1,283	10,389	4,603	4,654
	8192	0,456	0,518	0,347	4,739	40,724	18,703	18,618
	16384	0,988	1,275	0,752	17,563	155,526	71,473	72,732
	32768	2,146	2,516	1,421	69,963	624,905	281,861	286,844
	65536	4,623	5,294	3,065	279,23	2493,92	1127,103	1147,398

*Gráfica de resultados de la prueba 2 (**Orden Inverso**)*





Analisis:

En esta prueba el algoritmo más ineficiente fue el Intro_Sort, tomando un tiempo de procesamiento de 41 minutos, que de acuerdo a la teoría rompe el esquema del resultado esperado, de esta manera el toma un tiempo aproximadamente cuadrático. De igual manera, le siguen el Quicksort_3_way y el Dual_Pivot_Quicksort con tiempo de 20 minutos, esto debido a que los pivotes deben recorrer todo el arreglo ya que este se encuentra ordenado descendentemente y así dividir el arreglo como un ordenamiento Quicksort debe hacerlo.

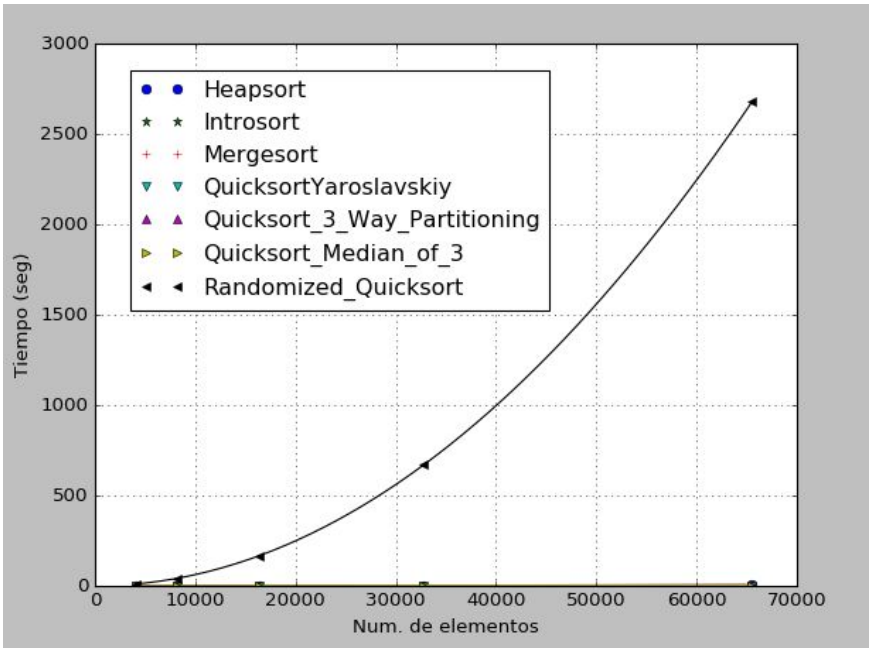
El Quicksort_3_way en este caso es el más eficiente alcanzando 3 segundos en el máximo arreglo, sin embargo tanto Heapsort como Mergesort siguen manteniendo una eficiencia de 5 segundos aproximadamente, los cuales seguiremos tomando en cuenta su rendimiento en las siguientes pruebas.

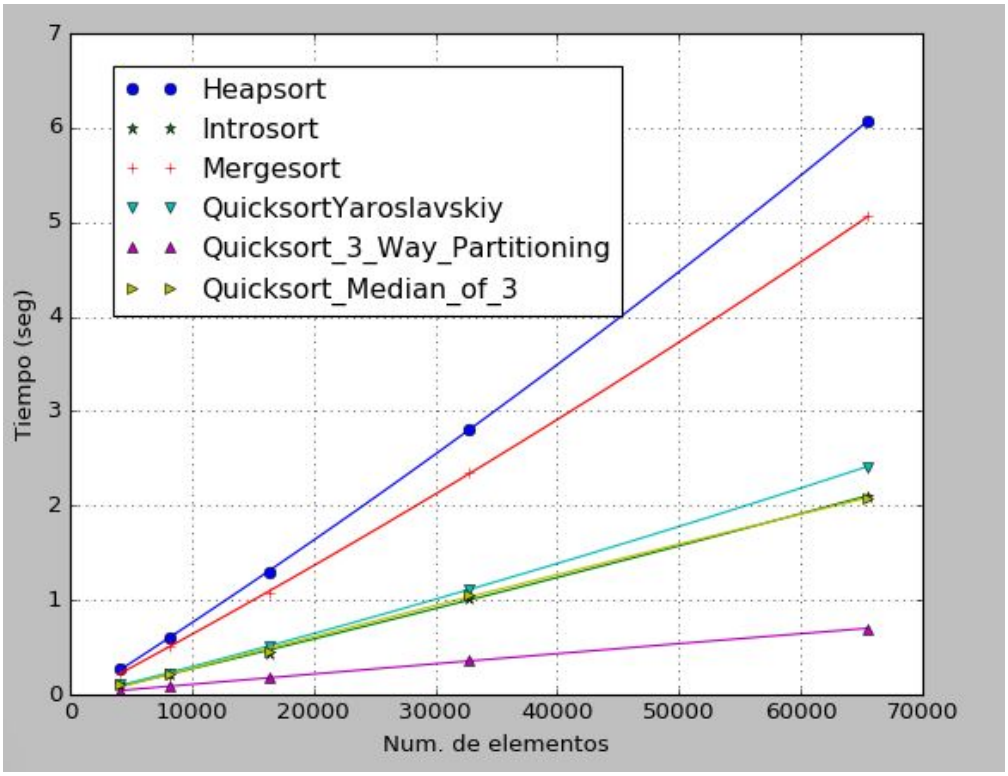
PRUEBA 3

*Tabla de resultados de la prueba 3 (**Ceros-unos**)*

Prueba 3	N	Mergeso	Heapsort	QS Random	Med-of-3	Introsort	Dual Pivot	QS 3-way
	4096	0.225	0.271	10.46	0.097	0.097	0.098	0.04
	8192	0.483	0.573	41.78	0.206	0.207	0.215	0.08
	16384	1.042	1.243	166.87	0.437	0.437	0.477	0.17
	32768	2.246	2.673	668.49	0.981	1.026	1.053	0.34
	65536	4.848	5.721	2678.15	2.118	2.117	2.327	0.69

*Gráfica de resultados de la prueba 3 (**Ceros-unos**)*





Analisis:

En la prueba se observa un caso muy peculiar, cuando el arreglo está comprendido entre puros 0s y 1s el quicksort randomizado es el de peor rendimiento, tomando forma cuadrática (n^2) pero si se detiene a analizar observamos que el quickso_median_of_3 debería también tomar una forma del estilo cuadrática. Esto es porque en el randomizado el elemento que escoja como pivote poco afecta ya que solo tienes 2 posibilidades y el número de recursiones que requiere es mucho comparado con los otros algoritmos, esto es porque al se verá obligado a estar ordenando y cambiando porque los elementos son 0s o son 1s. Por esta razón el median_of_3 debería tomar de igual forma n^2 ya que el elemento mediana de él primero la mitad y el último será 0 o será 1 y correrá el quicksort al igual que el randomizado.

Sin embargo vemos que el Quicksort_median_of_3 se encuentra entre los más eficientes, caso inesperado para nuestros análisis de eficiencia (concorde a la teoría). en términos de ineficiencia el que le sigue al Quicksort randomizado es el Heapsort. La razón de ellos es que la forma en la que distribuyen los padres e hijos ocasiona casi siempre el peor de los casos en el que el hijo es peor o igual que padre. en el caso del mergesort, al tener una distribución aleatoria de 0s y 1s se ve en la obligación de iterar tantas veces los cambios como 0 o 1 repetidos tengan los subarreglos a combinar en la nueva secuencia. El Dual_Pivot_Quicksort, al tomar 2 elementos de pivote aumenta (0 y 1) aumenta la eficiencia de separación del arreglo a entregar ya que al igual que el 3 way, almacena en el medio los

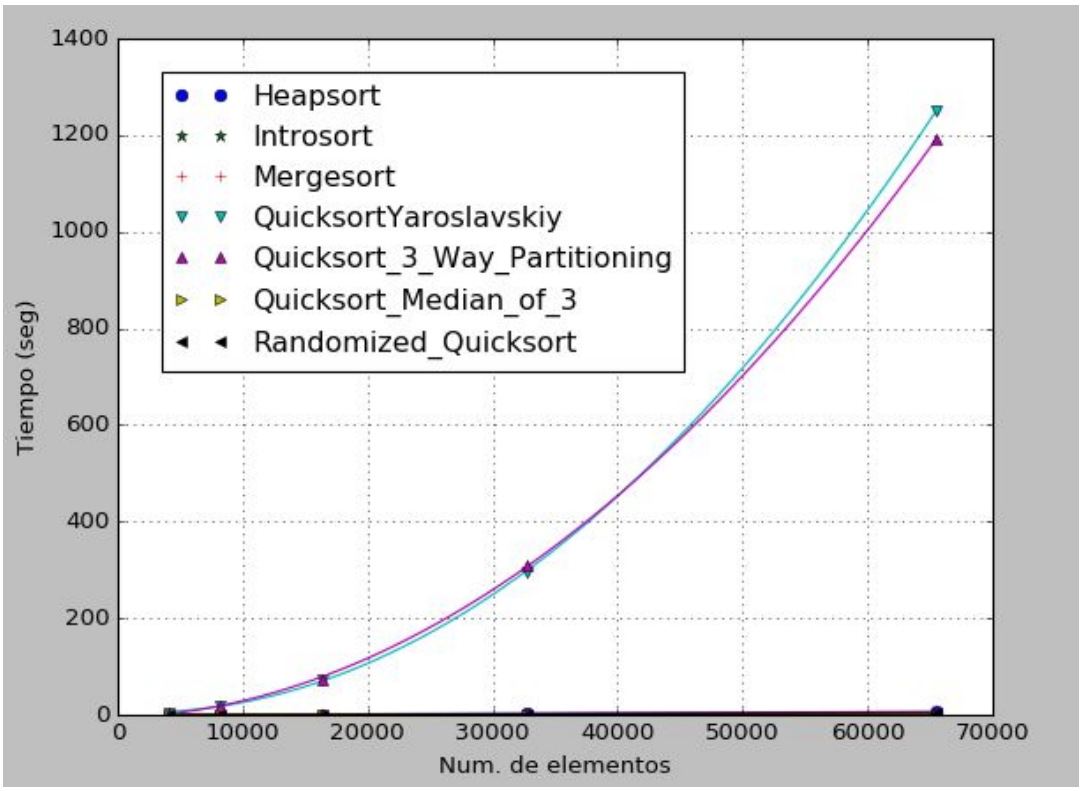
elementos menores o iguales a pivote 2 y mayores o iguales a pivote 1. El caso del 3 way, es el caso de prueba más eficiente para ejecutarse, ya que este algoritmo es sumamente optimo en arreglos donde tiene elementos repetidos.

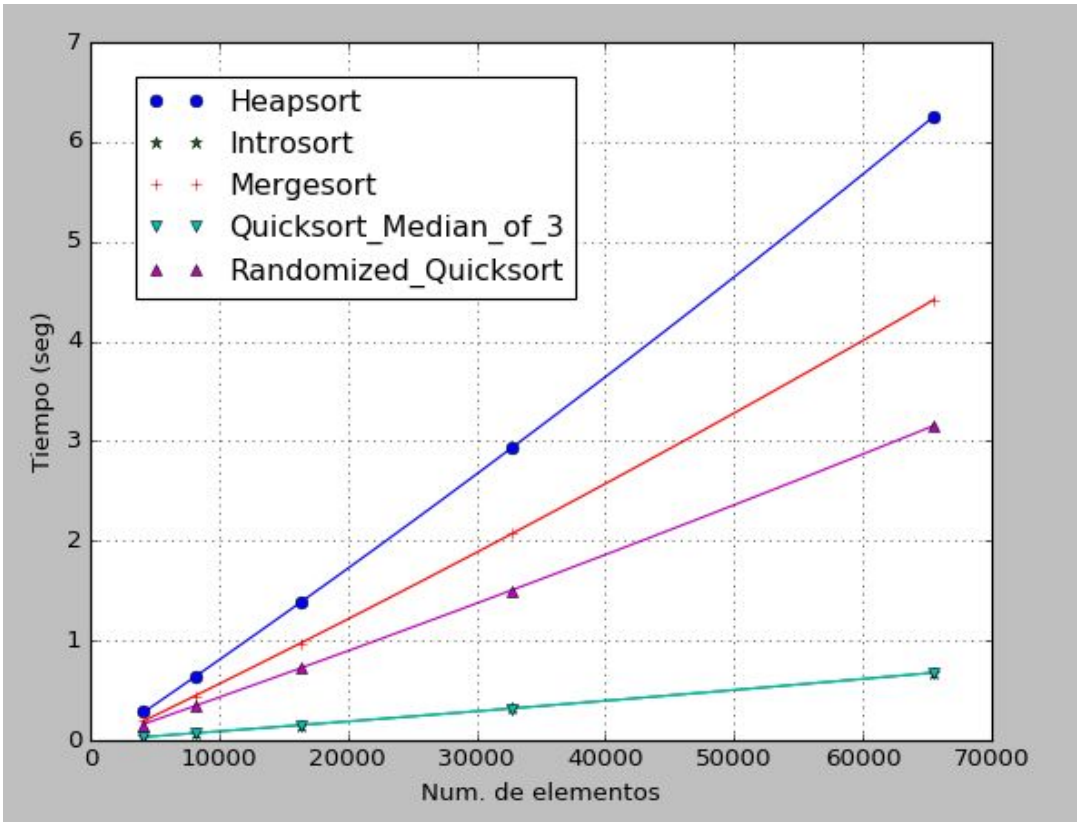
PRUEBA 4

*Tabla de resultados de la prueba 4 (**Ordenado**)*

	N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual Pivot QS	QS 3-way
Prueba 4	4096	0,198	0,279	0,149	0,032	0,032	4,621	4,57
	8192	0,431	0,609	0,339	0,068	0,068	18,32	18,203
	16384	0,931	1,315	0,7	0,146	0,145	73,344	72,804
	32768	2,03	2,841	1,605	0,31	0,309	269,717	311,173
	65536	4,605	6,443	3,3	0,657	0,757	1252,342	1192,529

*Gráfica de resultados de la prueba 4 (**Ordenado**)*





Analisis:

Tanto el Dual_Pivot_Quicksort y el Quicksort_3_way llegan a ser los más ineficientes a la hora de intentar ordenar un arreglo que ya está ordenado tomando un tiempo de 20 minutos cada uno, esto se debe a que como ambos son unas variaciones del Quicksort básico, quien también arroja resultados ineficientes al momento de procesar este mismo caso, ya que el pivote tiene que chequear y particionar elementos que ya están ordenados y por ende debe particionar elemento por elemento, aunque los dos algoritmos trabajan algo diferentes el resultado es prácticamente el mismo.

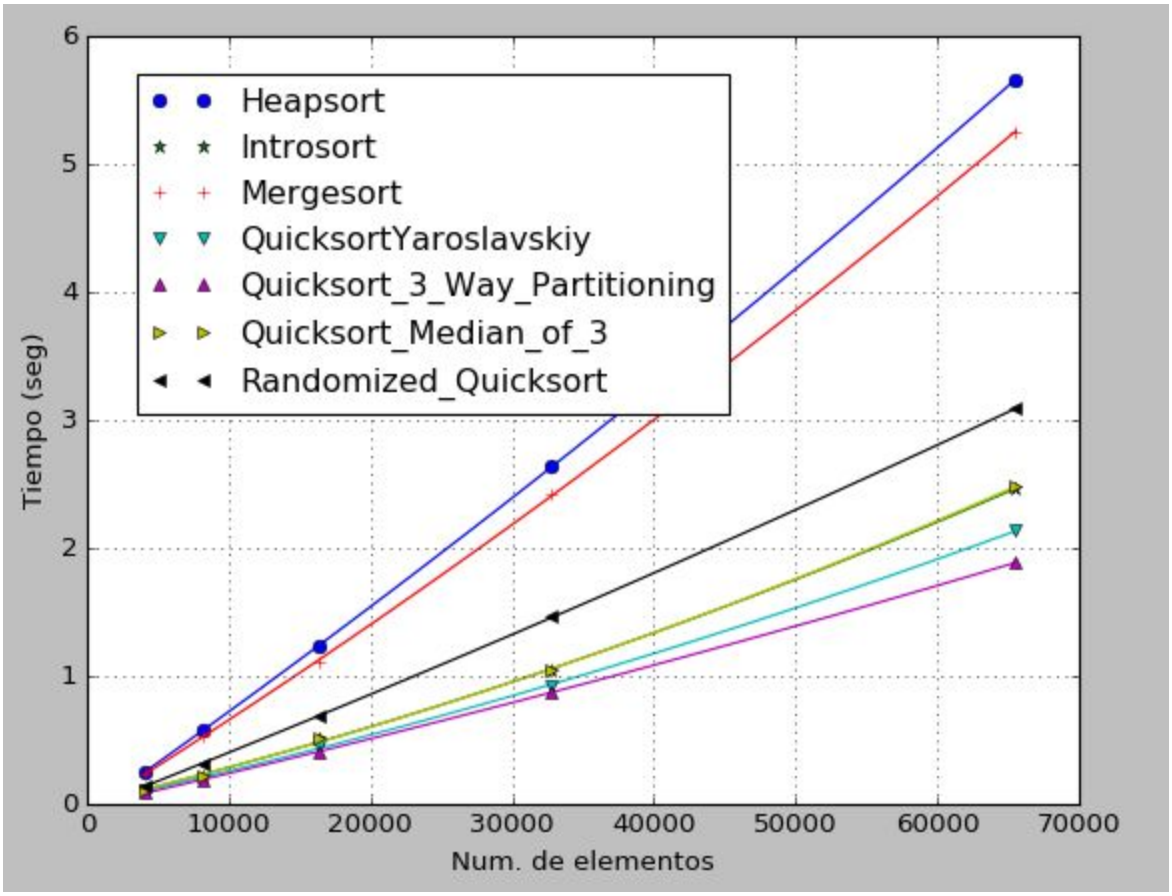
En este caso los algoritmos más eficientes son el Intro_sort y el Quicksort_Media_of_3 con un tiempo que no supera el segundo (0,7 segundos cada uno).

PRUEBA 5

*Tabla de resultados de la prueba 5 (**Reales aleatorios**)*

	N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual Pivot QS	QS 3-way
Prueba 5	4096	0,238	0,256	0,145	0,113	0,111	0,102	0,087
	8192	0,529	0,578	0,318	0,225	0,221	0,2	0,191
	16384	1,113	1,238	0,681	0,509	0,51	0,454	0,406
	32768	2,414	2,634	1,462	1,042	1,043	0,924	0,874
	65536	5,256	5,661	3,089	2,484	2,469	2,135	1,886

*Gráfica de resultados de la prueba 5 (**Reales aleatorios**)*



Analisis:

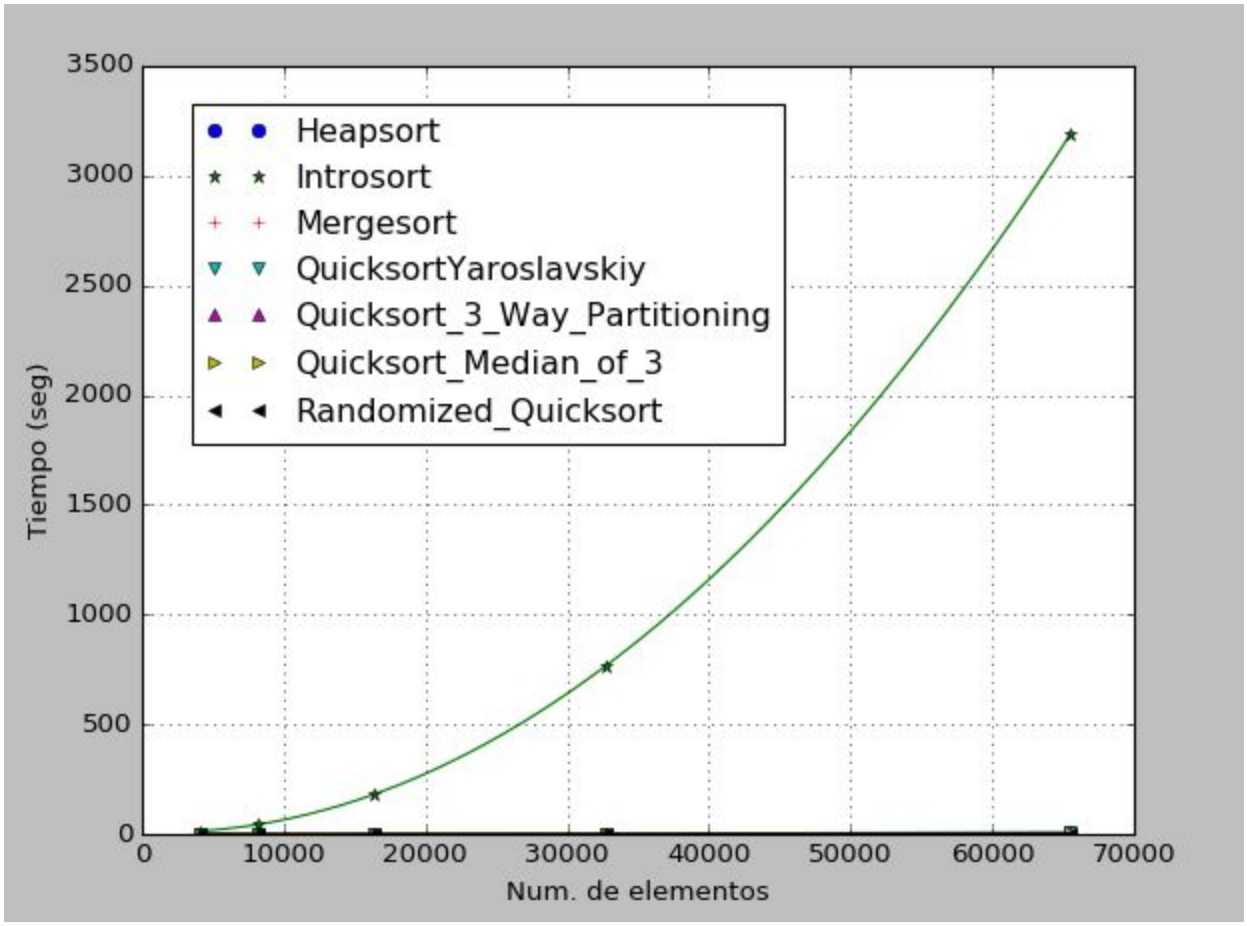
Este es el mejor caso para todos los algoritmos de ordenamiento, aunque tanto Heapsort y Mergesort son en esta prueba los más ineficientes siguen manteniendo su tiempo constante de 5 segundos y su eficiencia a lo largo de todas las pruebas. Podemos notar como arreglando elementos aleatorios, las variaciones del Quicksort han resultado ser las más eficiente y dar un mejor desempeño en diferencia a las demás pruebas, siendo el mejor el Quicksort_3_way con un comportamiento de n log n y un tiempo de 2 segundos.

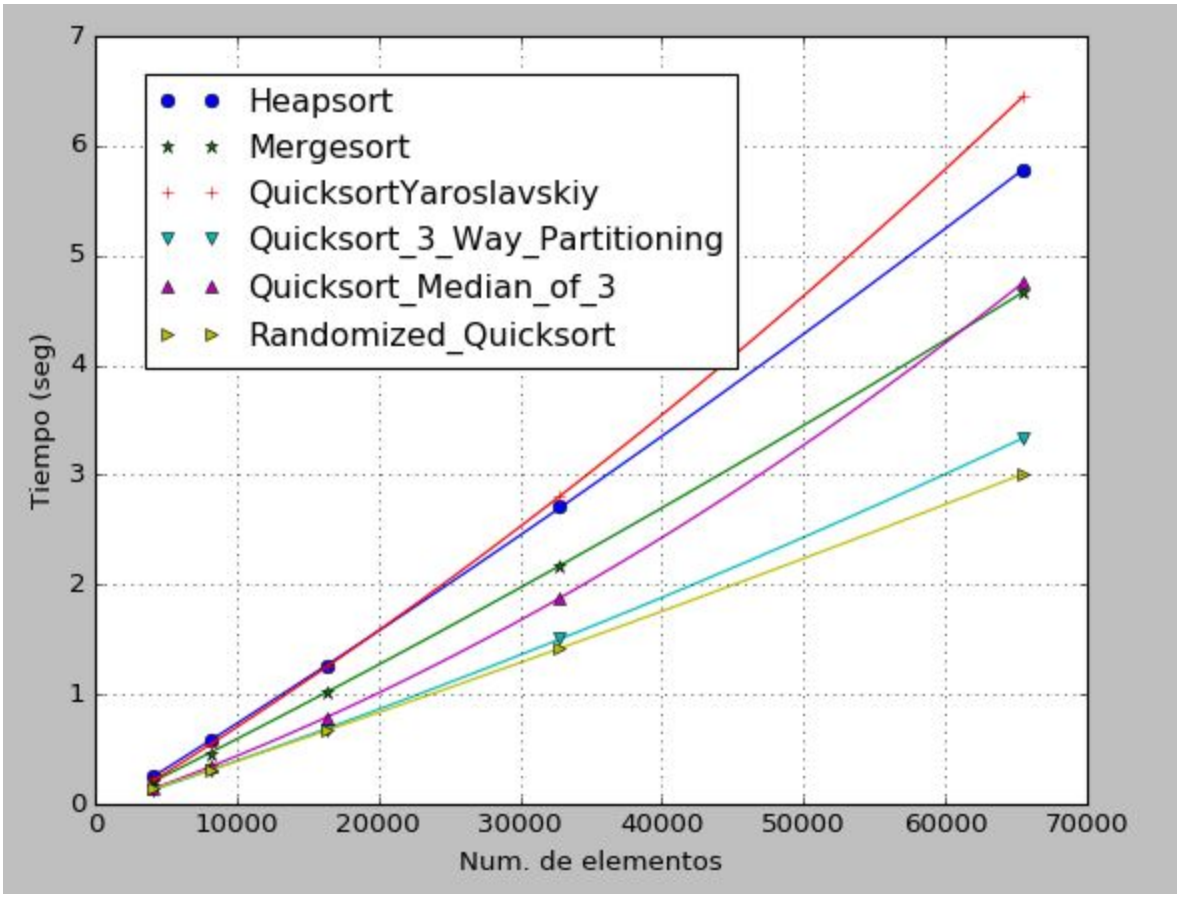
PRUEBA 6

*Tabla de resultados de la prueba 6 (**Mitad**)*

Prueba 6	N	Mergeso	Heapsort	QS Random	Med-of-3 Q	Introsort	Dual Pivot	QS 3-way
	4096	0.21	0.27	0.15	0.146	10.57	0.232	0.14
	8192	0.455	0.587	0.31	0.336	45.1	0.539	0.3
	16384	0.976	1.274	0.72	0.78	184.55	1.253	0.68
	32768	2.127	2.76	1.5	1.895	765.36	2.857	1.51
	65536	4.602	5.901	3.28	4.744	3191.09	6.562	3.34

*Gráfica de resultados de la prueba 6 (**Mitad**)*





Analisis:

El resultado de la primera gráfica deja pensativo la teoría del algoritmo Introsort. Esto es debido a que el Introsort en el peor de los casos tiene un tiempo de nLogn. sin embargo observamos como en la gráfica generada con tantos elementos (ver gráfica) es el peor de los algoritmos en un arreglo cuya primera mitad va ascendiente y segunda mitad descendiente. Dejando solo una duda al aire, *¿estará el Introsort asignándole todo el trabajo de ordenamiento a la llamada función Insertion_Sort()?* ya que es el caso en el su eficiencia es de n^2.

Saltando a los siguientes casos, se observa que efectivamente el Quicksort_3_way se encuentra entre los primeros (debido a que existen 2 de cada elemento repetido y esto por teoría es favorable para el código de ordenamiento del 3 way) pero más sorprendente aún es la eficiencia del Quicksort_Randomizado, lo cual hace pensar que la escogencia del pivote al azar fue de gran eficiencia para un Quicksort común.

observamos raramente el resultado del Dual_pivot_Quicksort comparándolo con los algoritmos por debajo de él. aunque partiendo de la teoría, al tomar el primer elemento y el último del arreglo estaría tomando el mismo elemento, lo cual lo estaría ejecutando un tiempo como el de un quicksort común y corriente en la selección del pivote. El caso del

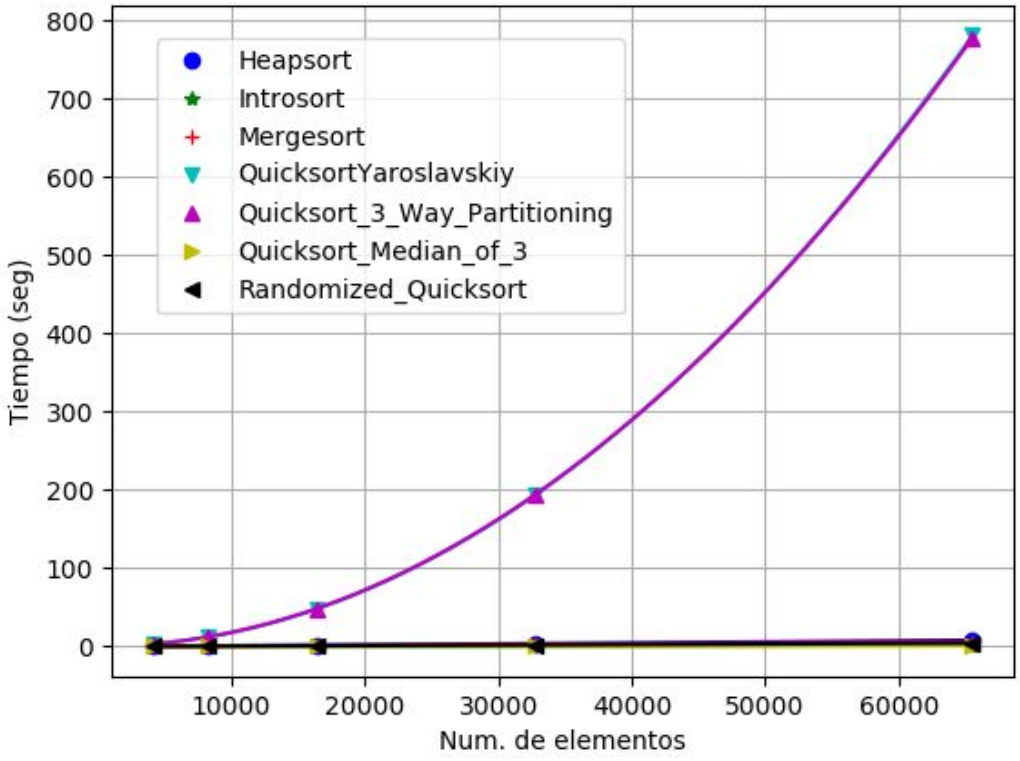
Quicksort_median_of_3 es un caso tal que la al seleccionar la mediana de los 3 elementos estaría seleccionando un elemento intermedio (middle) con respecto a los elementos del arreglo lo cual facilita el trabajo comparativo del *partition().* algo curioso en términos de tamaño del arreglo, mientras más grande sea, más ineficiente es con respecto al mergesort.

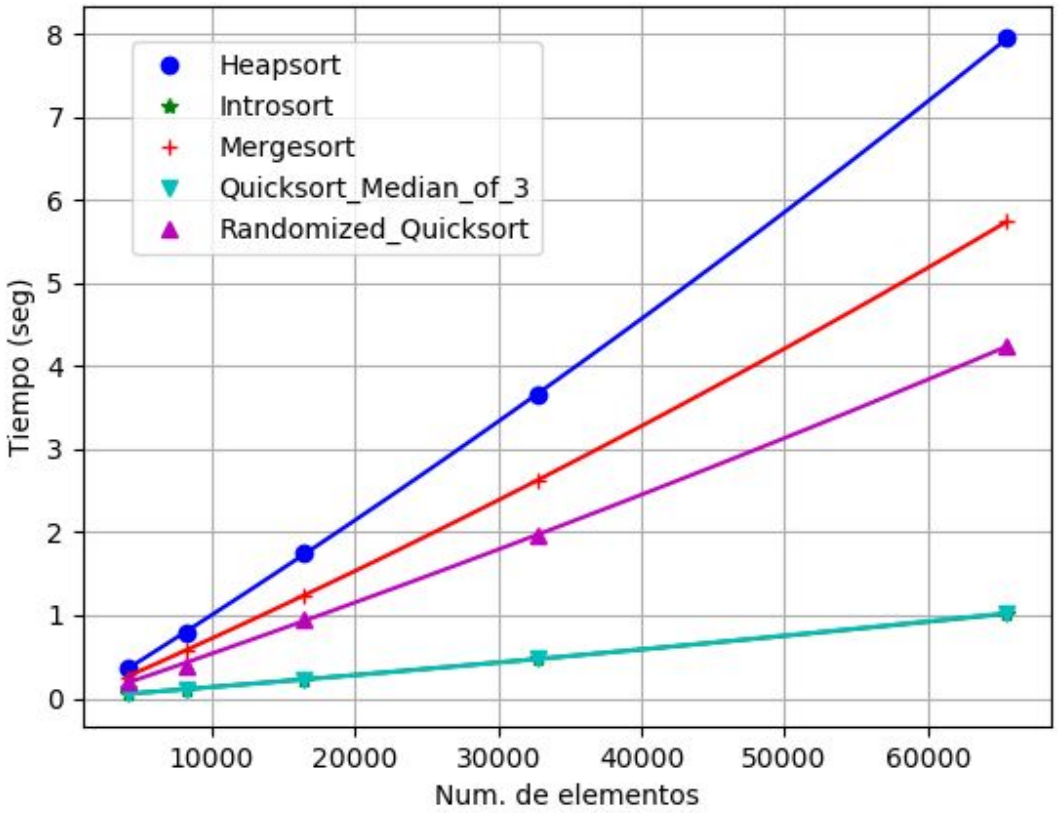
PRUEBA 7

*Tabla de resultados de la prueba 7 (**Casi Ordenado**)*

Prueba 7	N	Mergeso	Heapsort	QS Random	Med-of-3 C	Introsort	Dual Pivot	QS 3-way
	4096	0.279	0.365	0.195	0.05	0.051	3.11	2.99
	8192	0.574	0.817	0.408	0.11	0.11	12.47	12.26
	16384	1.222	1.748	0.891	0.23	0.229	48.9	48.13
	32768	2.646	3.749	1.852	0.48	0.481	195.97	193.58
	65536	6.059	8.095	4.359	1.04	1.04	790.19	784.79

*Gráfica de resultados de la prueba 7 (**Casi Ordenado**)*





Analisis:

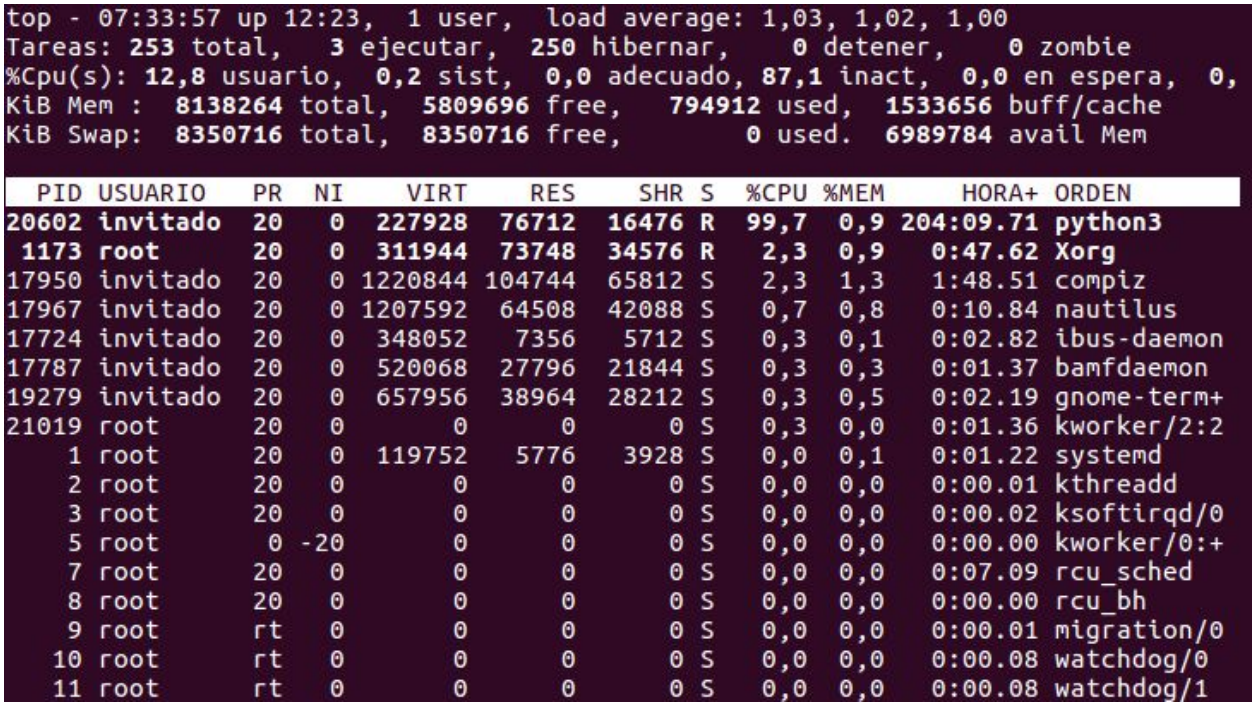
En la última prueba, una prueba donde todos los elementos del arreglo están casi desordenados y no se repiten elementos, era casi evidente que el Quicksort_3_way y el Dual_pivot_Quicksort serian los mas ineficientes, ya que son los que más iteraciones deben realizar al momento de ordenarlo pues no tienen elementos repetidos y se encuentra totalmente aleatorios lo cual les tomará más tiempo ordenarlos los elementos menores/mayores/intermediarios al/los pivote.

Luego en el análisis observamos un heapsort no tan ineficiente como los 2 anteriores, pero tampoco tan eficiente como los siguientes. esto a lo largo de las pruebas nos da una idea de lo “constante” o “estable” que son los tiempos promedios del merge, es decir, que el tiempo promedio suele ocurrir en la mayoría de los casos por el ordenamiento por montículo, tal que cumpla la condición de max-heap (el padre mayor que sus hijos). Seguido tenemos al Mergesort, corriendo en un tiempo totalmente promedio, típico de él mismo tomando su Θ(nLogn).

El quicksort randomizado toma un tiempo eficiente en este tipo de pruebas y esto depende debido a lo aleatorio del arreglo como lo aleatorio de la escogencia del pivote. Por último liderando esta prueba tenemos al Quicksort_median_of_3 junto con el introsort (algoritmo que nos sorprendió debido a la eficiente del heapsort). el quicksort median of 3 al tomar 3 elementos (primero, mitad y último) en un arreglo casi ordenado, tiene más probabilidad de que esos 3 elementos le arrojen el pivote perfecto para la realización del split(partition). Sin embargo el introsort al ejecutar el heapsort dentro de su codigo, debería por teoría y norma tener similitud con sus tiempos de eficiencia, pero la práctica dice que son tiempos totalmente discordantes.

ANÁLISIS EXTRA DEL RENDIMIENTO CPU

imagen del terminal de ubuntu: Comando TOP, monitoreo de procesos



Durante las pruebas y los análisis hubo un dato bastante curioso sobre el "rendimiento del cpu" puesto que al utilizar el comando: top (en shell bash) para monitorear la utilización de recursos en el sistemas, observamos que "%CPU" llegaba a valores inexplicables.

Tal como se observa en la imagen del terminal de ubuntu, el %CPU rondaba entre los 99% y 100% algo bastante curioso porque de ser así el procesador no tendría más memoria para ejecutar otros procesos. Sin embargo investigando en la red se encontró el siguiente dato:

"You are in a multi-core/multi-CPU environment and "top" is working in Irix mode. That

means that your process (vlc) is performing a computation that keeps 1.2 CPUs/cores busy. That could mean 100%+20%, 60%+60%, etc.

*Press 'I' to switch to Solaris mode. You get the same value divided by the number of cores/CPUs." -**http://unix.stackexchange.com/questions/34435/top-output-cpu-usage-100***

Lo cual ese 100% hace referencia a uno de los núcleos,threats del cpu. Al ser un i7 y ser multicore era capaz de sobrepasar ese 100% (haciendo referencia a 100% de un núcleo) por el mismo comando top que trabaja en Irix mode.

CONCLUSIÓN

Este informe nos ha servido para realizar comparaciones y aprender sobre los distintos algoritmos de ordenamiento, si bien hemos tenido alguna que otras dudas en cuanto al rendimiento de un algoritmo en la práctica con respecto a lo que dice la teoría y sobre todo en los algoritmos de las variaciones de Quicksort, que inesperadamente arrojaron resultados que no nos esperábamos.

Si hay algo que esta práctica nos ha enseñado es la eficiencia del Mergesort y del Heapsort, que a pesar de las distintas pruebas que se hicieron siempre fueron constante en el tiempo en que se ejecutaron, manteniendo una media entre 5 y 8 segundos en cada una de las pruebas, siendo las más constante y que no se ven afectada en ningún caso, su corrida siempre será nlogn.

BIBLIOGRAFÍA

http://algs4.cs.princeton.edu/lectures/23DemoPartitioning.pdf

https://www.toptal.com/developers/sorting-algorithms/quick-sort-3-way

http://maracaibo.ldc.usb.ve/~blai/courses/ci2612/handouts/ci2612-lec06.pdf

http://maracaibo.ldc.usb.ve/~blai/courses/ci2612/handouts/ci2612-lec08.pdf

https://es.wikipedia.org/wiki/Quicksort#T.C3.A9cnicas_de_elecci.C3.B3n_del_pivote