UNIVERSIDAD SIMÓN BOLÍVAR
DEPARTAMENTO DE COMPUTACIÓN Y TECNOLOGÍA DE LA INFORMACIÓN
CI-2692 - LABORATORIO DE ALGORITMOS Y ESTRUCTURAS II
TRIMESTRE ENERO-MARZO 2016

PROYECTO # 1:

Estudio experimental de algoritmos de ordenamiento

Integrantes:

Aurivan Castro Carné: 14-10205

Sandra Vera Carné: 14-11130

Caracas, febrero de 2017

INFORME DE RESULTADOS

En el presente trabajo se planea presentar y analizar los resultados obtenidos por medio de la aplicación del módulo cliente_ordenamiento.py, que provee el tiempo promedio de ejecución de varios algoritmos de ordenamiento dentro del módulo ordenamiento.py. A su vez permite la realización de diferentes pruebas con distintos tipos de arreglos para evaluar el comportamiento de cada algoritmo en cada ocasión.

A continuación se presentarán las diferentes pruebas, en conjunto con una tabla y una gráfica del rendimiento obtenido, seguidas por su respectivo análisis. Con el objeto de evitar la repetición, los nombres de los algoritmos se mencionan como máximo una vez dentro de varias pruebas y se procede a utilizar los acrónimos siguientes para identificarlas:

En las Tablas	En el Análisis	Significado
QS 3-way	QTW	Quicksort with three way partition
Dual Pivot QS	QDP	Dual Pivot Quicksort or Quicksort with Dual Pivot
QS Random	QSA	Quicksort aleatorio
Heapsort	HS	Heapsort
Mergesort	MS	Mergesort
Introsort	IS	Introsort
Med-of-3 QS	M3Q	Median-of-3 Quicksort

Tabla 0.1. Explicación de los términos utilizados

Aunado a lo anterior, cuando la prueba lo amerite, se presentará una segunda gráfica y tabla de la misma prueba omitiendo uno o más algoritmos, para luego continuar con el análisis. En dicho caso se indicará con un subtítulo.

Además, se colocará luego de cada subtítulo el computador donde dicha prueba fue realizada. A continuación se presentan las especificaciones de dichas maquinas.

Computador	A
Sistema Operativo	Debian GNU/Linux 8.7
CPU	Pentium(R) Dual-Core CPU, E5500 @ 2.80GHz, 64 bits
RAM	2 GB

Tabla 0.2. Especificación del Computador A

Computador	B
Sistema Operativo	Linux Mint 17.2 Rafaela
CPU	AMD E-450 APU with Radeon(tm) HD Graphics, 1.87 GHz
RAM	4 GB

Tabla 0.3. Especificación del Computador B

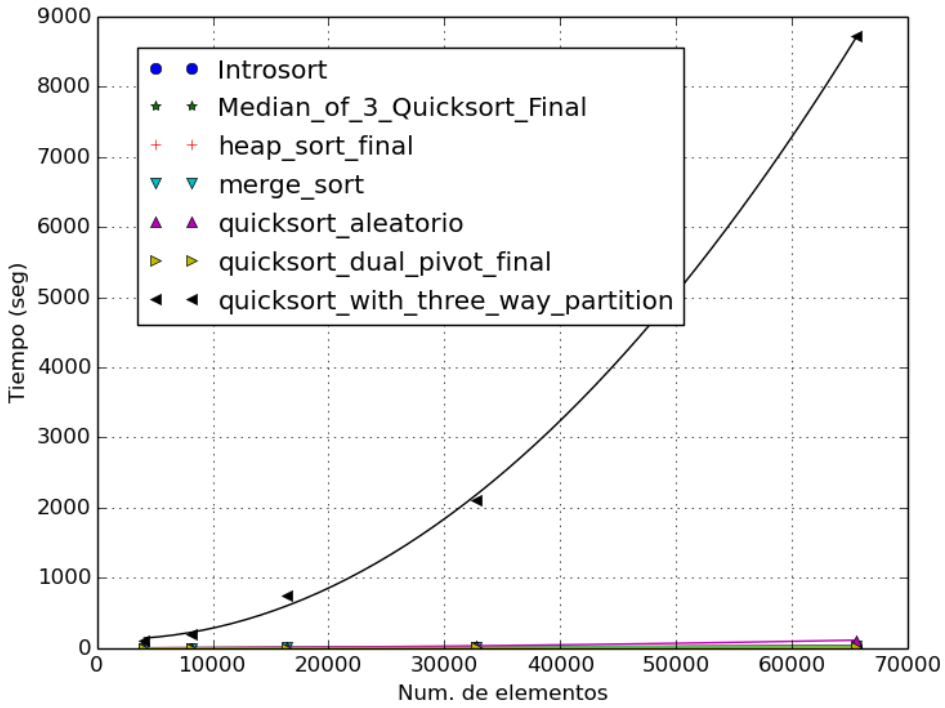
Computadora	C
Sistema Operativo	Ubuntu 16.04.1 LTS
CPU	Intel(R) Core(TM) i5-3230M CPU @ 2.60GHz
RAM	4 GB

Tabla 0.4. Especificación del Computador C

RESULTADOS Y ANÁLISIS

PRUEBA 1: Arreglo de enteros

Realizada en: Computador B



Gráfica 1.1. Prueba 1 – Enteros aleatorios

Prueba 1	**Enteros randomizados**						
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS	QS 3-way
4096	1,374	1,634	1,21	0,734	0,739	0,604	98,15
8192	3,302	3,261	3,427	1,719	1,716	1,134	196,3
16384	6,323	7,552	9,671	3,91	3,89	2,76	799,45
32768	13,899	16,364	33,037	10,998	11,036	5,893	2040,7
65536	29,185	35,277	114,848	35,639	30,687	12,151	8780,64

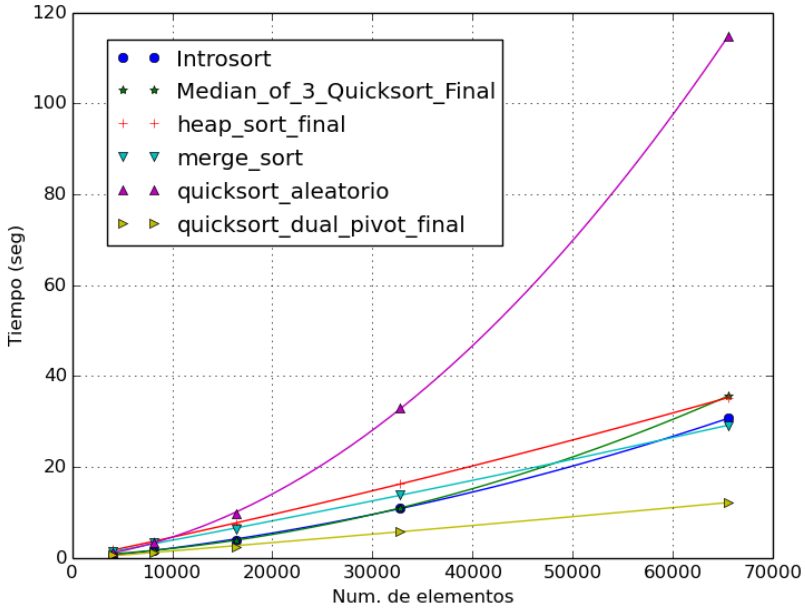
Tabla 1.1. Prueba 1 – Enteros aleatorios

Luego de la ejecución del algoritmo, al realizar la tabla comparativa, pudo notarse que el algoritmo de Quicksort with three way partition (QTW) fue el procedimiento que necesita mucho más tiempo para ordenar el arreglo. La gráfica presentada con anterioridad de la corrida de quicksort en conjunto con los otros algoritmos presenta un crecimiento bastante pronunciado del tiempo en comparación con los algoritmos anteriores, que se mezclan y poco pueden diferenciarse en la parte baja de la gráfica. Se concluye por lo tanto que dicho algoritmo presenta un crecimiento de tiempo mucho mayor, ameritando una segunda corrida sin dicho algoritmo.

Por otra parte, la razón de este suceso debe ameritarse al código de QTW. Además de ser un código recursivo, esto es, que se llama a si mismo dentro del código, utiliza ciclos sin límite especificado (ciclos usando 'while True') y otros dentro de dichos ciclos ('while' dentro de 'while True'). Esto ralentiza el proceso considerablemente, aunado a su vez a los ciclos que siguen. Asimismo, la llamada recursiva dentro del código es doble, en otras palabras, hace dos llamadas recursivas dentro de sí mismo, sobre distintas partes el algoritmo. Estas se irán acumulando y alentando el proceso de ordenamiento en sobremanera. Si bien otros algoritmos como Quicksort con pivote dual (QDP) también presentan llamadas recursivas (en el caso de DPQ, son tres, y en este sentido, más que las que tiene QTW), no presentan ciclos indiferenciados ('while True') que se detendrán sólo cuando un 'break' ocurra dentro del código. A su vez es importante recalcar que QTW posee además muchas condiciones y comparaciones que debe realizar para ejecutar otros comandos, lo que lleva a su vez a que el CPU tome más tiempo en procesarlo.

Ahora, como el crecimiento de QTW amerita la segunda corrida sin el mismo, se colocarán las gráficas a continuación:

- Prueba 1 sin QTW



Gráfica 1.2. Prueba 1 sin QTW – Enteros aleatorios

Prueba 1	**Enteros randomizados sin QS 3-way**					
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS
4096	1,2	1,876	1,02	0,88	0,68	0,61
8192	3,195	3,65	3,409	1,809	1,678	1,095
16384	6,3	7,68	10,034	3,67	3,98	2,98
32768	14,003	16,073	34,001	11,098	11,23	5,763
65536	30,65	34,09	115	36,7	30,6	12,18

Tabla 1.2. Prueba 1 sin QTW – Enteros aleatorios

Lo más notable de la gráfica es el crecimiento del algoritmo de quicksort_aleatorio (QSA) en comparación los algoritmos anteriores. Si bien para algoritmos de tamaño menor a 10000 es aparentemente eficiente, logrando un mejor rendimiento que Heapsort (HS) y que Mergesort (MS), para arreglos de mayor tamaño su rendimiento empieza a desmejorar. Aunque sigue siendo mucho más eficiente que QWT, es notable que todos los demás algoritmos (en esta corrida) tienen menor tiempo de corrida para arreglos grandes. Observando el código de QSA, puede notarse que toma un elemento al azar entre los del arreglo como pivote. Ya que se generan enteros entre [0,500], para arreglos demasiado grandes (entre 30000 y 60000) con un rango tan pequeño, es probable que QSA haga particiones desbalanceadas al tomar los pivotes, de manera que necesita más comparaciones para arreglar los elementos.

Los siguientes con tiempo muy parecido son HS y MS. Asintóticamente no presentan muchas diferencias, aunque HS corre en mayor tiempo que MS. Esto se debe a que MS divide las secuencias en mitades hasta obtener el caso trivial (un solo elemento), y luego las combina de

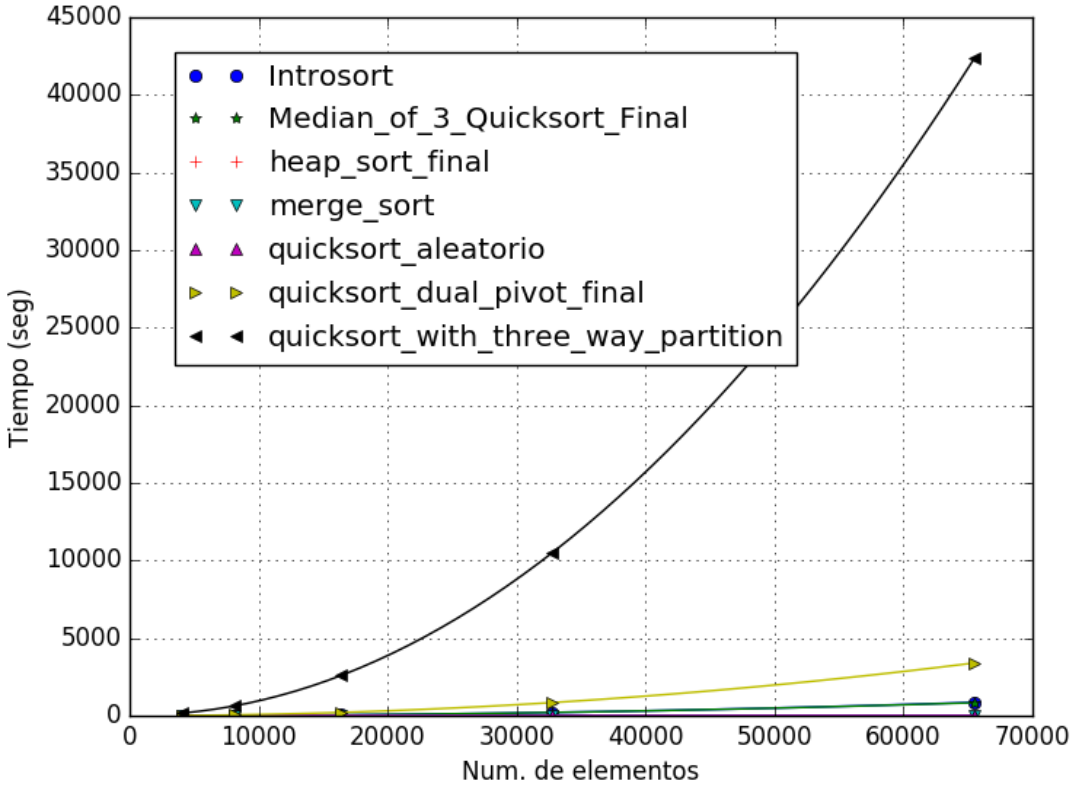
manera ordenada. Por otro lado, HS debe reestablecer el heap cada vez que un elemento se coloca al final de la lista, por lo que necesita más tiempo para arreglos de mayor tamaño que MS.

Luego, observando Introsort (IS) y Median_of_3_Quicksort (M3Q), es notable que su desempeño es casi el mismo hasta arreglos de alrededor de 32000 elementos. Para arreglos mayores a 40000 elementos, M3Q es más lento que IS. Esto se debe a que M3Q toma la mediana del primero, el Último y el elemento de la mitad, por lo que para arreglos muy grandes en el rango [0,500], es posible que tome elementos muy parecidos y las particiones sean un poco menos balanceadas que en IS. Cabe destacar que IS limita la profundidad de la recursión, lo que también contribuye a una mayor rapidez.

Finalmente, el más rápido es quicksort_dual_pivot (QDP). Lo anterior es posible por el modo de funcionamiento del algoritmo, el cual consiste en tomar dos pivotes, uno al final y otro al principio del segmento de arreglo a ordenar, y luego ordenar los elementos de manera de que todos los que sean menores al menor pivote tomado se coloquen al principio del arreglo y antes de dicho pivote, y los que sean mayores al mayor pivote se colocan luego de dicho pivote, hacia el final del arreglo. Todo esto contribuye a que el tiempo de ejecución en esta prueba sea considerablemente menor en comparación a los otros algoritmos, haciendo a QDP el más eficiente cuando los elementos son randomizados.

PRUEBA 2: Arreglo de enteros ordenados de manera decreciente

Realizada en: Computador C



Gráfica 2.1. Prueba 2 – Arreglo ordenado de forma decreciente

Prueba 2	**Arreglo ordenado de manera decreciente**						
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS	QS 3-way
4096	0,398	0,534	0,297	3,001	2,275	12,03	157,003
8192	1,44	1,874	1,013	13,273	13,821	53,563	647,075
16384	2,066	3,764	1,584	50,205	51,058	209,369	2582,782
32768	5,662	7,583	4,054	209,759	205,905	845,553	10493,136
65536	10,822	15,556	8,952	842,234	860,445	3403,512	42410,643

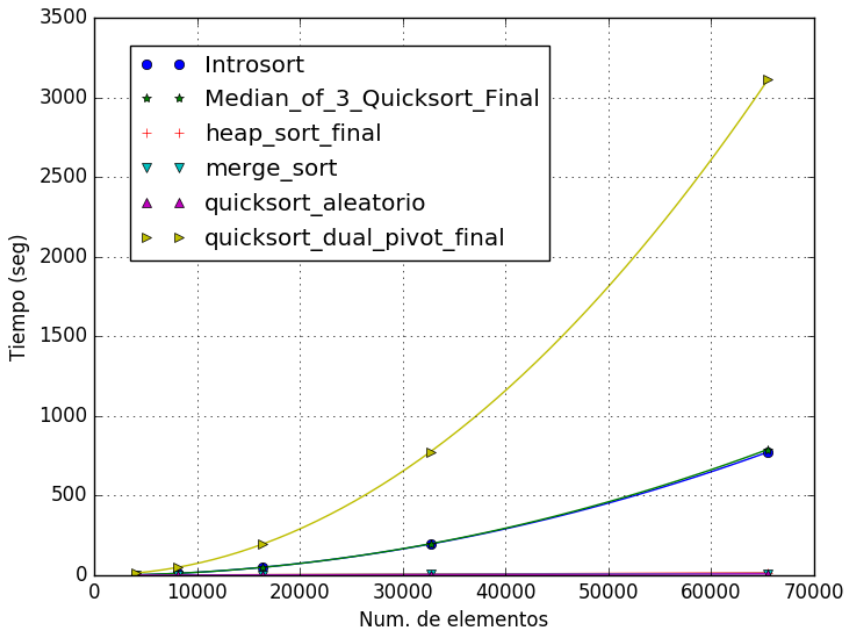
Tabla 2.1. Prueba 2 – Arreglo ordenado de forma decreciente

Luego de la ejecución del algoritmo en esta prueba, al realizar la tabla comparativa y observando la gráfica, pudo notarse que el algoritmo QTW fue el que necesitó más tiempo para ordenar el arreglo, además que, claramente, presenta un crecimiento muy pronunciado en la gráfica. Dicho crecimiento si bien no alcanza un comportamiento cuadrático, se dispara de manera que es imposible distinguir la parte que contiene el comportamiento de los otros algoritmos en la gráfica, lo que supone la realización de una segunda corrida sin dicho algoritmo.

Por otro lado, la razón de dicho comportamiento del algoritmo se debe al código de QTW. Además de ser un código recursivo que utiliza ciclos sin límite especificado (usando 'while True') y otros dentro de dichos ciclos ('while' dentro de 'while True') como fue expuesto en análisis anteriores, realiza muchas comparaciones y se deben comprobar muchas condiciones para que ejecute un comando. Es importante destacar la manera en la que el algoritmo toma los pivotes. En el código se observa que toma el primero y el Último como pivotes. Ya que todos serán menores al primer elemento y mayores al Último (por ser un arreglo ordenado de manera descendente) se realizan muchas comparaciones y particiones desbalanceadas, repitiendo este proceso cada vez que se realiza una llamada recursiva del algoritmo.

Debido al crecimiento de QTW, se ameritó una segunda corrida sin el mismo, se colocarán las gráficas a continuación:

- Prueba 2 sin QTW



Gráfica 2.2. Prueba 2 sin QTW – Arreglo ordenado de forma decreciente

Prueba 2	Arreglo ordenado de manera decreciente sin QS 3-way					
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS
4096	0,366	0,448	0,705	3,491	2,778	12,716
8192	1,434	1,298	0,638	12,824	12,335	50,142
16384	2,284	3,511	1,903	48,521	48,649	198,804
32768	5,397	6,857	3,36	199,684	197,897	776,188
65536	10,859	14,346	8,994	790,733	774,002	3113,783

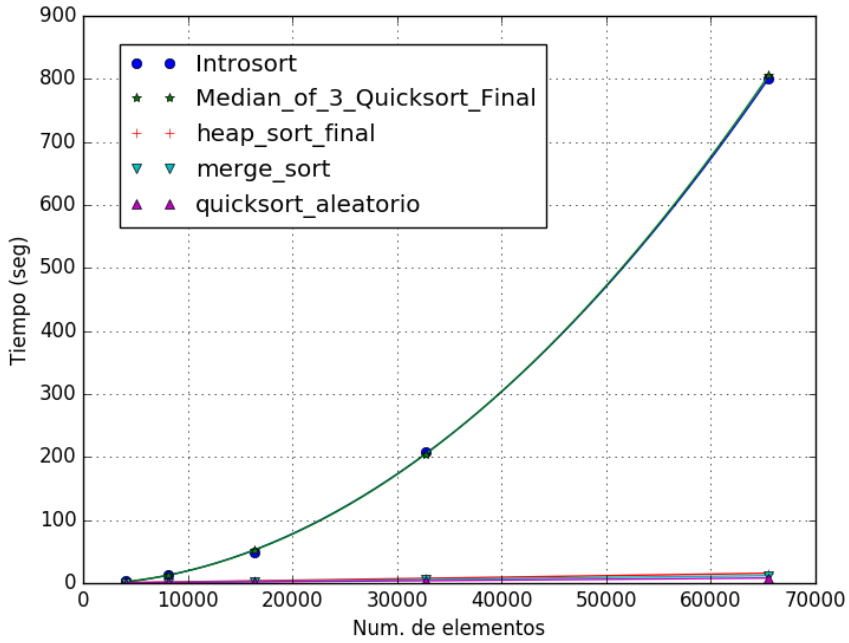
Tabla 2.2. Prueba 2 sin QTW – Arreglo ordenado de forma decreciente

En esta prueba se puede observar que, comparado con los demás algoritmos, QDP presenta un mayor crecimiento asintótico que no permite el análisis del resto de los algoritmos, lo que indica que es necesario realizar otra corrida sin dicho algoritmo. Además se evidencia que posiblemente M3Q e IS presentarán un mayor crecimiento que HS, MS y QSA.

Observando el código de QDP, observamos que toma como pivotes el primer y el Último elemento de arreglo, es decir el mayor elemento y el menor, luego los intercambia y, al momento de ordenar los elementos restantes, claramente no habrá elementos menores al menor ni mayores al mayor. Por lo tanto, al momento de escoger los pivotes auxiliares que están entre los pivotes de los extremos, reordenar los elementos entre ellos y luego llevar a cabo las llamadas recursivas, el algoritmo toma una gran cantidad de tiempo pues el proceso de particionar de esta manera se repite hasta que el arreglo a ordenar es de tamaño 10 y se procede a ordenar con Insertionsort.

Nuevamente, se realiza una nueva corrida sin los algoritmos de QTW y QDP.

- Prueba 2 sin QTW y QDP



Gráfica 2.3. Prueba 2 sin QTW y QDP – Arreglo ordenado de forma decreciente

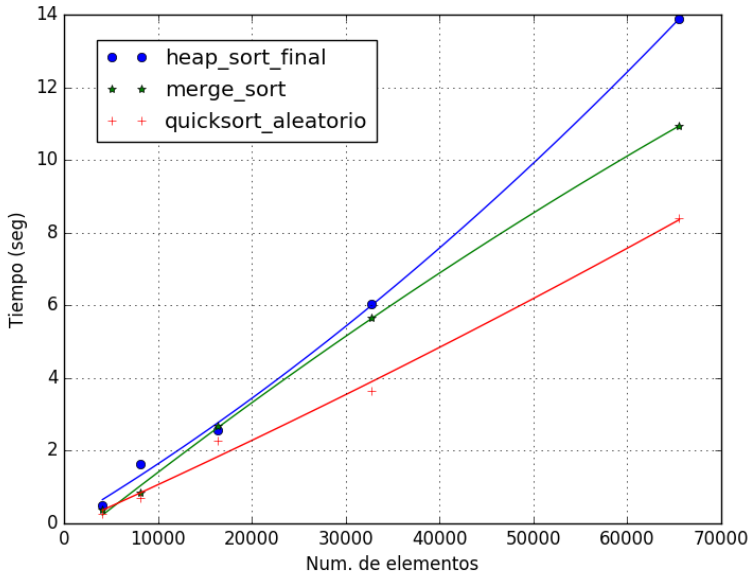
Prueba 2	**Arreglo ordenado de manera decreciente sin QS 3-way y Dual Pivot QS**				
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort
4096	0,429	0,551	0,354	2,396	2,823
8192	0,882	1,76	1,204	12,501	12..751
16384	2,465	3,739	2,205	52,176	49,641
32768	5,804	7,445	3,662	205,229	207,406
65536	11,867	15,649	8,12	805,979	800,882

Tabla 2.3. Prueba 2 sin QTW y QDP – Arreglo ordenado de forma decreciente

Como se sospechó anteriormente, los algoritmos de IS y M3Q presentan un crecimiento asintótico notablemente mayor a los algoritmos HS, MS y QSA, a la vez que es difícil de precisar cuál fue el más rápido. A pesar de tardar menos tiempo que QTW y QDP, para el caso de M3Q, ya que el arreglo está en orden descendente, toma como pivote el elemento de la mitad del arreglo, por lo que debe invertir el arreglo y realizar muchas comparaciones entre los elementos y muchos cambios de posiciones entre ellos. Esto debe repetirse hasta que la cantidad de elementos a ordenar sea igual a 10, cuando el arreglo es ordenado por Insertionsort. Para IS es análogo, ya que la partición se realiza de manera similar a M3Q.

Finalmente, se ejecutaron los algoritmos de HS, MS y QSA.

- Prueba 2 sin QTW, QDP, IS y M3Q



Gráfica 2.4. Prueba 2 sin QTW, QDP, IS y M3Q – Arreglo ordenado de forma decreciente

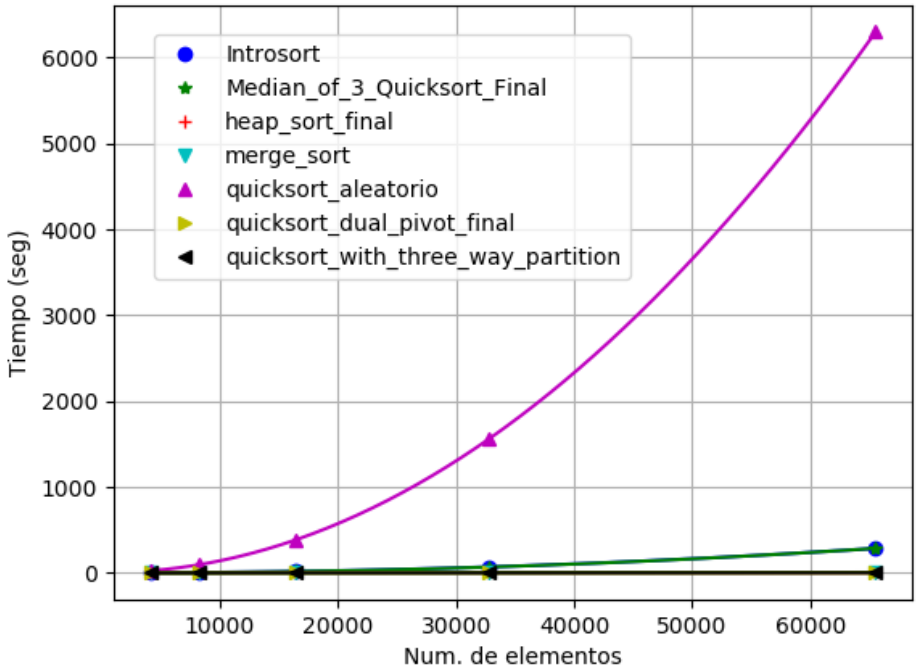
Prueba 2	**Arreglo ordenado de manera decreciente sin QS 3-way, Dual Pivot QS, Introsort y Med-of-3 QS**		
N	Mergesort	Heapsort	QS Random
4096	0,37	0,492	0,266
8192	0,837	1,629	0,705
16384	2,671	2,574	2,287
32768	5,66	6,045	3,642
65536	10,934	13,884	8,396

Tabla 2.4. Prueba 2 sin QTW, QDP, IS y M3Q – Arreglo ordenado de forma decreciente

Una vez analizada la gráfica y los tiempos de ejecución de cada algoritmo, es claro que QSA es el que toma menos tiempo en ordenar los elementos, incluso para grandes cantidades de elementos. Esto se debe a que toma un pivote aleatorio, lo que evita con gran probabilidad los casos donde se toman elementos muy cercanos a los extremos como pivotes, lo que mejora significativamente el tiempo de corrida. QSA es seguido en rapidez por MS, ya que este Último divide el arreglo en dos mitades recursivamente y luego une las mitades de manera ordenada, además de ser un algoritmo asintóticamente óptimo. El tercero más rápido es HS, pues como el arreglo ya se encuentra ordenado de forma descendente, no es necesario establecer un heap máximo. Asimismo, no hace falta reestablecer el heap cada vez que se coloca un elemento al final de la lista.

PRUEBA 3: Arreglo de ceros y unos aleatorios

Realizada en: Computador A



Gráfica 3.1. Prueba 3 – Arreglos de ceros y unos aleatorios

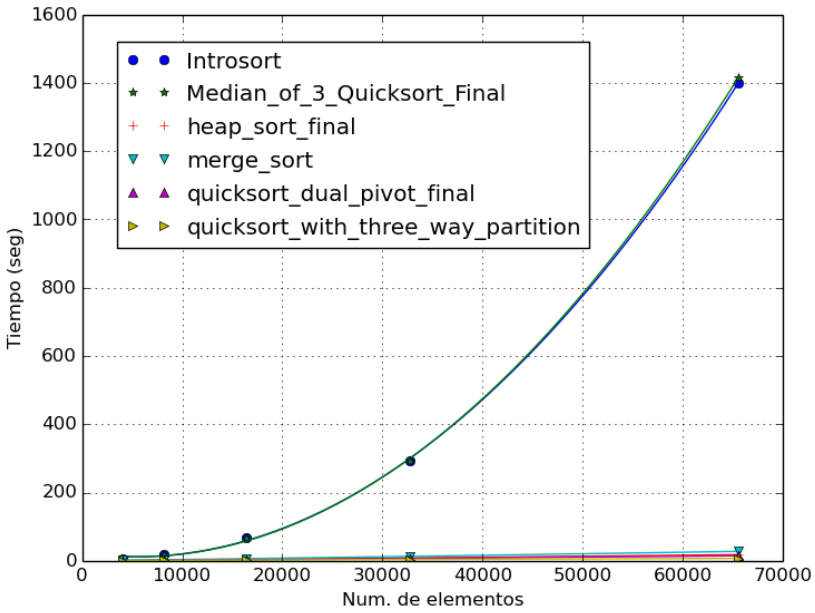
Prueba 3	**Arreglo con ceros y unos**						
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS	QS 3-way
4096	0,314	0,211	24,3	1,138	1,156	0,159	0,108
8192	0,705	0,451	95,822	5,024	5,083	0,358	0,212
16384	1,457	0,973	358,652	19,599	19,558	0,756	0,413
32768	3,2	2,162	1558,165	66,642	68,277	1,61	0,88
65536	7,017	4,466	6301,598	286,544	286,577	3,698	1,717

Tabla 3.1. Prueba 3 – Arreglos de ceros y unos aleatorios

Lo primero que puede notarse es que, contrario a las pruebas anteriores, Quicksort with three way partition (QTW) es de hecho uno de los algoritmos más eficientes en esta prueba, mientras que Quicksort aleatorio (QSA) ha descendido hasta el menos eficiente, llegando a tomar un tiempo de cerca de dos horas para ordenar un arreglo de más de 65000 elementos. Si bien al principio, para arreglos de tamaño menor a 10000, la diferencia entre el tiempo de QSA y los demás algoritmos no es tan grande, a medida que crece el número de elementos por arreglo, lo hace de manera pronunciada el tiempo que tarda QSA en ordenarlo.

Por el código de QSA es posible determinar cuál es la causa de dicho rendimiento. La función dentro de QSA, 'partition', fija un elemento (en este caso el Último del segmento a utilizar dentro de la misma) como el pivote con el cual está comparando todos los demás antes que él. Ahora bien, como la prueba implica que el arreglo sólo poseerá ceros y unos en vez de otros enteros, la comparación de '<=' (menor o igual) de partition, si el pivote tomado es 1, provoca que se ejecute lo que está por debajo de la comparación para todos los elementos de ese segmento del arreglo. Como la posibilidad de tomar 1 como pivote es de 50%, la cantidad de veces que QSA debe realizar cambios de posiciones dentro del arreglo, cuando la longitud del mismo supera a los 10000, aumenta de manera pronunciada, haciendo de QSA el algoritmo menos eficiente en esta prueba.

- Prueba 3 sin QSA



Gráfica 3.2. Prueba 3 sin QSA – Arreglos de ceros y unos aleatorios

Prueba 3	Arreglo con ceros y unos sin QS Random					
N	Mergesort	Heapsort	Med-of-3 QS	Introsort	Dual pivot QS	QS 3-way
4096	1,331	0,818	4,637	4,776	0,628	0,447
8192	2,777	1,732	18,908	19,058	1,34	0,889
16384	6,056	3,798	66,795	67,152	3,041	1,889
32768	13,019	8,126	293,829	294,074	6,644	3,672
65536	27,512	17,664	1417,441	1401,258	14,59	6,943

Tabla 3.2. Prueba 3 sin QSA – Arreglos de ceros y unos aleatorios

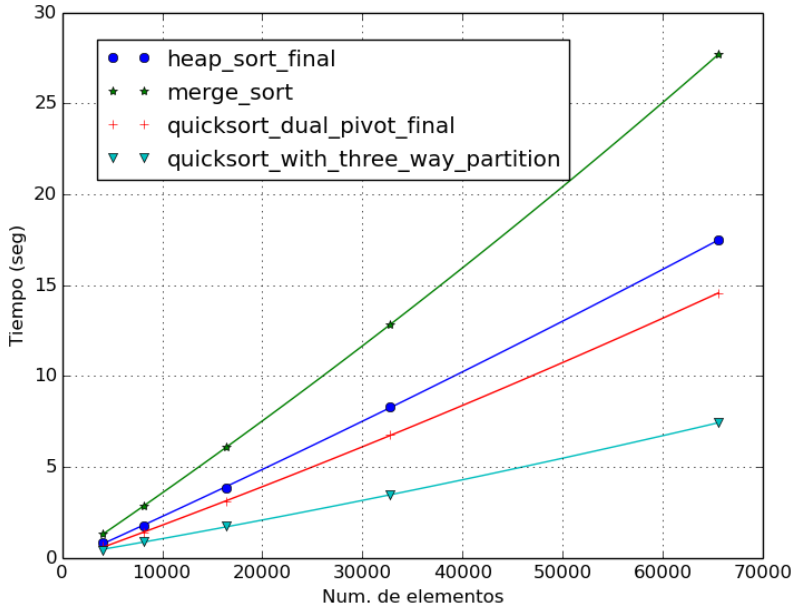
Una vez realizada la prueba nuevamente omitiendo la ejecución de QSA, ocurre una situación similar, pero esta vez con dos algoritmos. Introsort (IS) y Median-of-3 Quicksort (M3Q) tienen un aumento mayor en su tiempo de ejecución que los demás algoritmos restantes en esta prueba. Si bien ninguno puede competir con lo poco eficiente que terminó siendo QSA en esta prueba, ambos llegan a una cantidad de tiempo considerablemente mayor a los demás a medida que aumenta el número de elementos del arreglo a ordenar. Cabe señalar que durante el desarrollo de la prueba los tiempos de ejecución de ambos son muy parecidos, lo que conduce a que en la gráfica ambas representaciones están muy cerca de si.

Por una parte, un factor que hace a IS más eficiente que QSA es que IS trabaja con el límite de recursiones, de manera que si el número de recursiones supera dicho límite, el algoritmo deja de procesar lo que sigue al final del 'while', realizando Heapsort (HS) del arreglo y haciendo 'break', con lo cual termina la ejecución de IS. Ahora bien, lo que hace este algoritmo menos eficiente es el uso de 'partition_median' y 'median_of_3' (encontrándose ambas funciones definidas en la sección de MEDIAN-OF-3 QUICKSORT a partir de la línea 178) dentro de la función 'Instrosort_loop'.

No solamente el hecho que 'partition_median' realiza varias comparaciones, 'median_of_3' también las realiza, aunado a que presenta varios ciclos 'while True' dentro de su código, lo que podrá conllevar a muchas recursiones que aumenten el tiempo de ejecución. Esto surge de que todos los elementos del arreglo sólo puedan ser cero o ser uno, lo que conlleva a que las comparaciones sean algo desbalanceadas, por ejemplo, comparar 0 con 0 y cambiar elementos de posición en el arreglo o decir que 1 es menor o igual a 1 y realizar otro proceso. Esto se torna bastante acumulativo en cuanto a la cantidad de recursos a utilizar.

Por otra parte, M3Q tiene a su vez que utilizar las mismas funciones, 'partition_median' y 'median_of_3', repetidas veces, muy parecido a lo que hace IS. De igual manera, por lo explicado anteriormente, estas contribuyen a que el tiempo de corrida del algoritmo aumente.

- Prueba 3 sin QSA, IS y M3Q



Gráfica 3.3. Prueba 3 sin QSA, IS y M3Q – Arreglos de ceros y unos aleatorios

Prueba 3	**Arreglo con ceros y unos sin QS Random, Introsort y Med-of-3 QS**			
N	Mergesort	Heapsort	Dual pivot QS	QS 3-way
4096	1,32	0,852	0,644	0,436
8192	2,848	1,76	1,368	0,902
16384	6,104	3,863	3,113	1,736
32768	12,83	8,317	6,766	3,445
65536	27,7	17,479	14,569	7,439

Tabla 3.3. Prueba 3 sin QSA, IS y M3Q – Arreglos de ceros y unos aleatorios

Dejando a un lado lo ocurrido con los tres algoritmos mencionados con anterioridad, es necesario observar el comportamiento de los algoritmos restantes. Es importante señalar que todos se ejecutaron, para un arreglo de tamaño mayor a 65000, en un tiempo menor a 30 segundos, lo cual es bastante eficiente. Ahora bien, destacando individualmente el rendimiento, resulta evidente que Quicksort with three way partition (QTW) es el más eficiente de todos en esta prueba, superando a los otros procedimientos.

Antes que nada, es importante señalar que existen varios motivos por los cuales esto ocurre. En primer lugar, a pesar de que QTW realiza muchas comparaciones y ciclos dentro de otros ciclos para llegar a una cierta partición del arreglo, como los elementos del mismo están compuestos solamente de ceros y unos, muchas de las comparaciones tienden a no cumplirse, por ejemplo, el hecho de comparar un pivote igual a 1 con un elemento igual a 1

por medio de la relación menor estricto para entrar dentro de un ciclo. El poder saltarse muchos pasos y comparaciones hace de este algoritmo bastante eficiente para ordenar arreglos de este tipo.

Por otra parte, los demás algoritmos tardan un poco más y presentan pequeñas diferencias en dichos resultados. Mergesort (MS), el algoritmo que más tiempo toma de entre este grupo de 4, tarda un poco más por el hecho de utilizar más memoria para crear una nueva lista copia del arreglo dado. Además, el comparar elementos que pueden ser iguales (1 y 1 o 0 y 0) contribuye a que tome un poco más de tiempo en procesar.

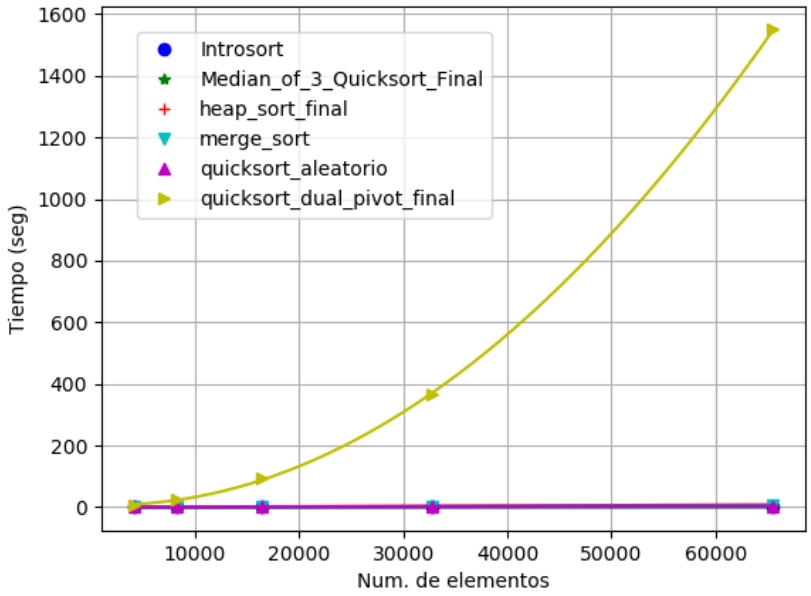
En el caso de Heapsort (HS) y Quicksort with dual pivot (QDP), se presentan rendimientos muy parecidos. La estructura de Heapsort que distingue nodos de un árbol y luego los intercambia sólo ocurre con elementos diferentes, pero la probabilidad de que esto ocurra es sólo del 50% (menor comparado con un arreglo con elementos cualesquiera dentro de un rango real), lo que provoca que funcione de manera bastante rápida. QDP, por su lado, escoge dos pivotes cualesquiera y manda los elementos menores al pivote menor al principio del arreglo y los mayores al pivote mayor al final, sin embargo, si ambos sólo pueden ser 0 o 1, el proceso de comparaciones se hace mucho más rápido.

PRUEBA 4: Arreglo de ceros y unos aleatorios

Realizada en: Computador A

Antes que nada, es importante resaltar el crecimiento cuadrático de los tiempos que toma el algoritmo de Quicksort with three way partition (QTW) en comparación con los demás ejecutados, en función al tamaño de los arreglos. Por la materia de teoría es conocido que para la mayoría de los algoritmos de tipo Quicksort, el peor caso posible en cuanto a las medidas de tiempo es cuando el arreglo ya está ordenado. En otras palabras, las particiones que realiza quicksort dentro de esta prueba están evaluando elementos unos con otros y posiblemente moviéndolos dentro del arreglo, intentando encontrar la combinación correcta. Todo esto conlleva a la realización de demasiadas operaciones que se apilan dentro de la memoria y ralentizan todo el proceso.

- Prueba 4 sin QTW



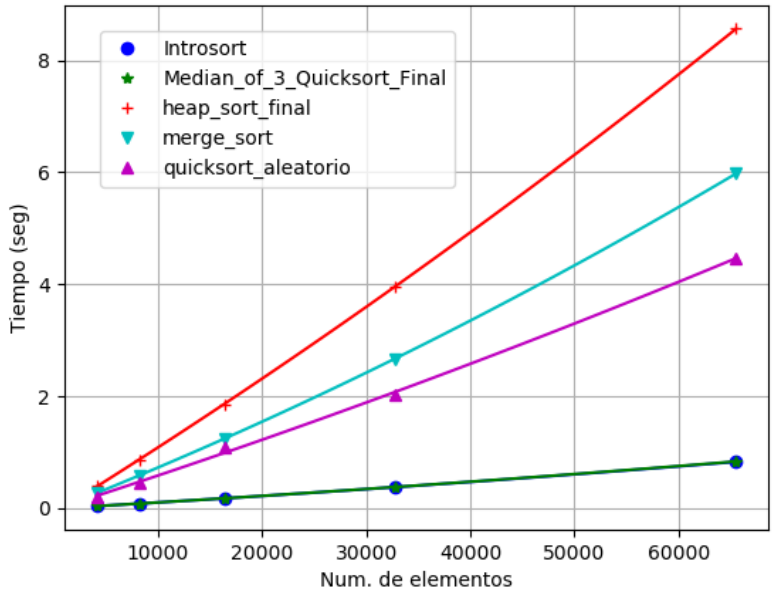
Gráfica 4.1. Prueba 4 sin QTW– Arreglos ordenados

Prueba 4	**Arreglo ordenado sin QS 3-way**					
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS
4096	0,255	0,384	0,192	0,038	0,039	6,079
8192	0,541	0,842	0,417	0,084	0,088	23,53
16384	1,204	1,776	0,853	0,185	0,175	93,232
32768	2,53	3,798	1,899	0,381	0,375	366,444
65536	5,843	8,202	4,029	0,857	0,821	1540,201

Tabla 4.1. Prueba 4 sin QTW – Arreglos ordenados

Ahora, una vez más puede observarse que uno de los algoritmos supera en sobremanera el tiempo de ejecución de todos los demás. Si bien triunfa sobre QTW, es notable que Quicksort Dual Pivot (QDP) que cae en derrota frente a los otros algoritmos, los cuales en la gráfica presentada están muy cerca en rendimiento y todos a la vez alejados en dicho sentido de QDP. La razón de esto es la manera de funcionar de QDP: tomar dos pivotes y evaluar todo lo que está entre ellos a manera de colocar todo lo que es menor al pivote menor al principio y lo que es mayor al pivote mayor al final. Como el arreglo ya está¡ ordenado, esto es, el algoritmo reordena los elementos una y otra vez, pues puede que consiga elementos mayores al pivote correspondiente y los coloca después de elementos mayores a esto, necesitando más llamadas recursivas sobre sí mismo hasta que la situación sea solucionada.

- Prueba 4 sin QTW y QDP



Gráfica 4.2. Prueba 4 sin QTW y QDP– Arreglos ordenados

Prueba 4	**Arreglo ordenado sin QS 3-way y Dual Pivot QS**				
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort
4096	0,281	0,394	0,197	0,039	0,039
8192	0,569	0,862	0,45	0,083	0,083
16384	1,248	1,857	1,08	0,178	0,177
32768	2,675	3,966	2,025	0,379	0,378
65536	5,679	8,564	4,475	0,836	0,83

Tabla 4.2. Prueba 4 sin QTW y QDP – Arreglos ordenados

Dejando de lado QDP y ejecutando nuevamente cliente_ordenamiento, se obtiene una gráfica mucho más ilustrativa del rendimiento de los demás algoritmos, los más eficientes en esta prueba, desarrollándose en menos de 9 segundos para arreglos bastantes grandes. Todos presentan crecimiento lineal o casi lineal, y hay dos que tienen un rendimiento prácticamente idéntico.

En primer lugar destaca Heapsort (HS), que como dentro de su código cambia el primer elemento del arreglo con el Último, necesita de varias comparaciones y cambios dentro del mismo para llevarlo a su forma original, pero aun así requiere un número de

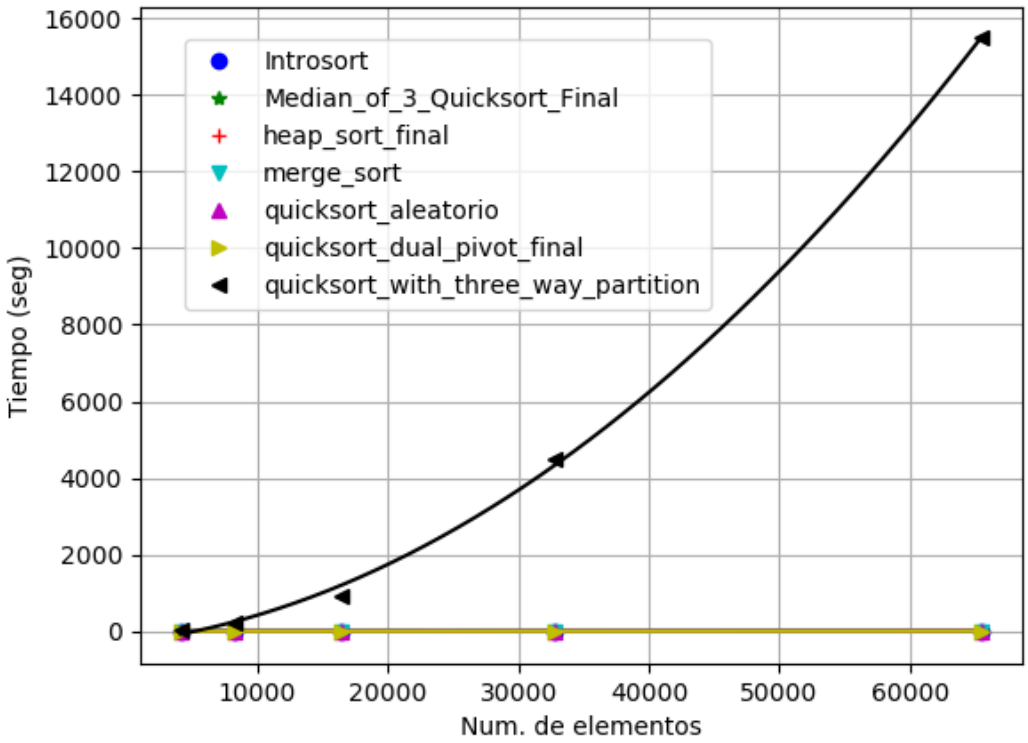
comparaciones y cambios considerablemente menor al de los algoritmos mencionados con anterioridad, pero aun así mayor que los siguientes.

Le siguen Mergesort (MS) y Quicksort aleatorio (QSA), con un tiempo menor a 6 segundos para arreglos de tamaño mayor a 65000, triunfando este Último sobre el primero por casi dos segundos. Por una parte, QSA realiza un número bastante pequeño de comparaciones, pues dentro de su código, en la función 'partition' (definida en la sección QUICKSORT ALEATORIO a partir de la línea 138), se especifica que sólo realizará cambios si el elemento tomado es menor o igual al pivote, y como todo el arreglo está ordenado, sólo se realizarán cambios cuando sean iguales. Por otra parte, MS primero particiona todo el arreglo en pequeñas unidades que luego compara, pero como ya está ordenado, procede a comparar todos los trozos y unirlos, luego los siguientes trozos y unirlos, y así sucesivamente.

Compartiendo el trofeo de algoritmos más eficientes sobre la prueba 4 se encuentran Median-of-3 Quicksort (M3Q) e Introsort (IS). Ambos presentaron rendimientos básicamente idénticos y mejores a todos los demás algoritmos. Su parecido radica en que utilizan las mismas funciones, 'median-of-3' y 'partition-median' (definidas bajo la sección de MEDIAN-OF-3 QUICKSORT a partir de la línea 178). Su buen rendimiento, a su vez, se centra en el uso de estas funciones. En ellas, como el arreglo ya está ordenado, gran parte de las comparaciones son básicamente descartadas y los ciclos se hacen bastante cortos por esta misma razón. Todo lo anterior hace que cualquiera de los dos pueda ser escogido como el más eficiente cuando los arreglos ya están ordenados.

PRUEBA 5: Arreglo de números reales aleatorios entre 0 y 1

Realizada en: Computador A



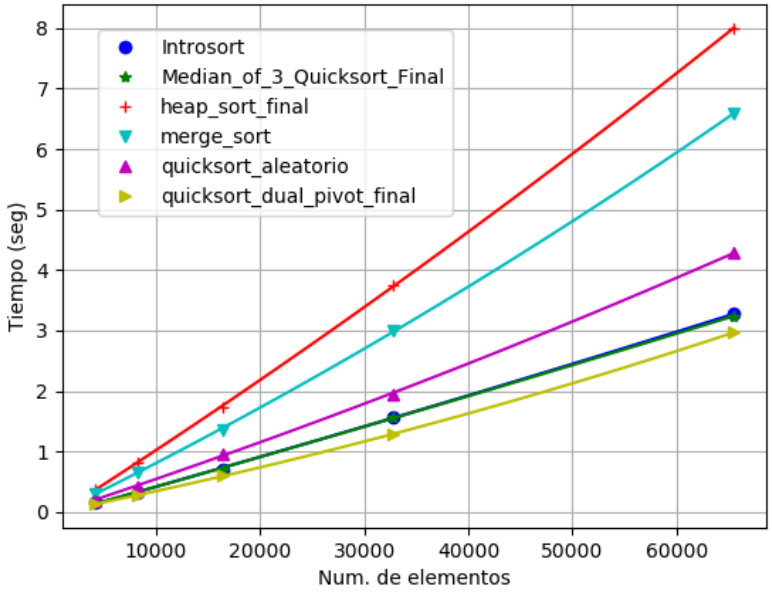
Gráfica 5.1. Prueba 5 – Arreglos de números reales entre 0 y 1

Prueba 5	**Arreglo con numero reales aleatorios entre 0 y 1**						
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS	QS 3-way
4096	0,297	0,37	0,187	0,157	0,157	0,137	54,914
8192	0,65	0,827	0,395	0,322	0,337	0,276	238,108
16384	1,388	1,773	0,865	0,689	0,688	0,624	924,407
32768	3,053	3,79	1,938	1,511	1,494	1,295	4504,868
65536	6,711	8,124	4,229	3,219	3,239	3,06	15494,762

Tabla 5.1. Prueba 5 – Arreglos de números reales entre 0 y 1

Es evidente tanto por la gráfica como la tabla presentada que el algoritmo que toma más tiempo en ejecutarse para esta prueba es Quicksort with three way partition (QTW). Similar a lo ocurrido en la prueba 1, QTW necesita realizar muchas comparaciones y utiliza ciclos dentro de otros ciclos además de llamadas recursivas sobre sí mismo para lograr ordenar el arreglo proporcionado. Todas estas acciones requieren de muchos recursos que Python debe obtener y utilizar, consiguiendo de esta manera un mayor tiempo de ejecución, sobre todo a medida que aumenta el número de elementos del arreglo. Sobre la gráfica, esto se traduce como una curva muy pronunciada.

- Prueba 5 sin QTW



Gráfica 5.2. Prueba 5 sin QTW – Arreglos de números reales entre 0 y 1

Prueba 5	**Arreglo con numero reales aleatorios entre 0 y 1 sin QS 3-way**					
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS
4096	0,299	0,37	0,193	0,153	0,148	0,126
8192	0,647	0,807	0,422	0,315	0,317	0,267
16384	1,36	1,735	0,963	0,703	0,702	0,606
32768	2,993	3,742	1,95	1,568	1,574	1,28
65536	6,593	8,009	4,286	3,237	3,273	2,971

Tabla 5.2. Prueba 5 sin QTW – Arreglos de números reales entre 0 y 1

Ahora bien, realizando la llamada a cliente_ordenamiento omitiendo QTW, es posible observar los comportamientos de los demás algoritmos. Es notable que todos parecieran tener un crecimiento lineal de tiempo vs tamaño del arreglo, y que el hecho de ejecutarse en

un tiempo menor o igual a 8 segundos los convierte en algoritmos eficientes bajo estas condiciones.

Evaluando por separado el rendimiento, primero se debe de señalar varios aspectos de los códigos para entender su comportamiento. En primer lugar, el algoritmo que toma más tiempo dentro de esta segunda ejecución de cliente_ordenamiento es Heapsort (HS). Lo anterior se debe a que dentro de su código necesita realizar varios intercambios de elementos del arreglo cada vez que coloca el Último como el primero, asegurándose de que los nodos hijos y los padres guarden la relación establecida, contribuyendo a que tome más tiempo que todos los demás.

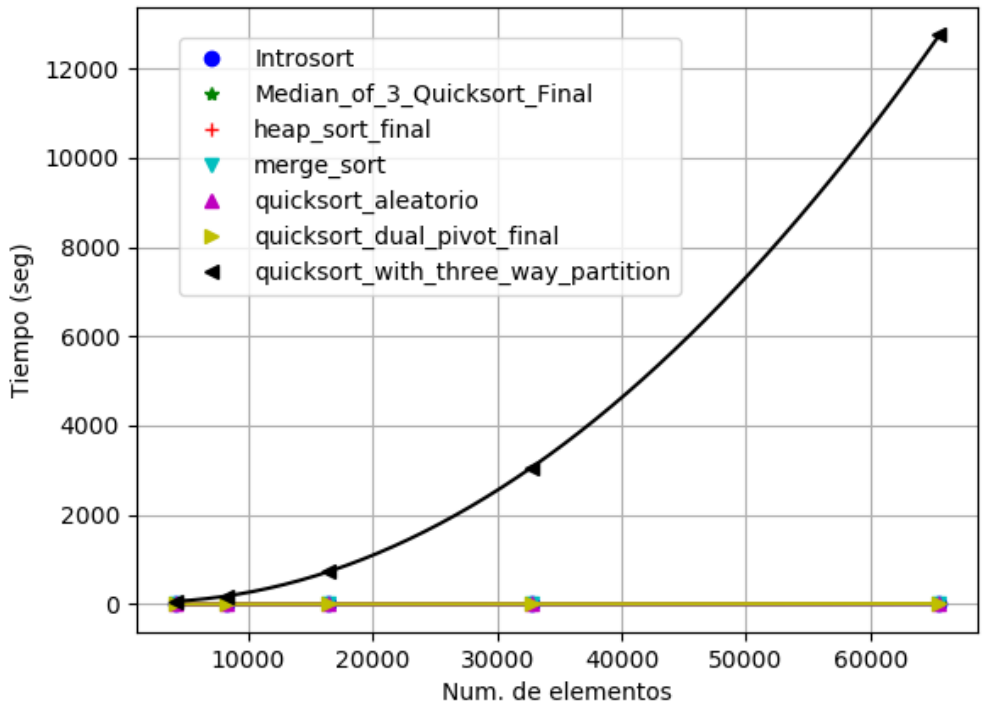
El siguiente en rendimiento menor es Mergesort (MS), que de manera parecida a HS, necesita hacer muchas más comparaciones que se acumulan y ralentizan (sólo con pequeñas diferencias de los demás) todo el proceso. Esto también ocurre con quicksort aleatorio (QSA), pero el randomizar el arreglo inicial le brinda una ligera ventaja con respecto a MS, pues reduce la posibilidad de obtener particiones desbalanceadas que contribuyan a un tiempo mayor de ejecución.

Le siguen Median-of-3 Quicksort (M3Q) e Introsort (IS), los cuales son algoritmos muy parecidos que utilizan algunas funciones en común, en este caso, 'partition_median' y 'Median_of_3' (ambas definidas en la sección de MEDIAN-OF-3 QUICKSORT, a partir de la línea 178), lo cual puede haber contribuido en su desempeoro casi idéntico.

Ahora bien, el algoritmo más eficiente fue sin dudas Quicksort Dual Pivot (QDP). Lo anterior es debido a la manera de trabajar del algoritmo: primero elige dos pivotes y reordena a los elementos del arreglo de manera que los menores al pivote menor están al principio del arreglo, y los mayores al pivote mayor al final, dejando de por medio aquellos que no cumplan con estas especificaciones. El utilizar partes separadas del arreglo contribuye a disminuir los recursos utilizados para las recursiones, y en consecuencia hace de este algoritmo el más rápido de todos en esta prueba.

PRUEBA 6: Arreglo mitad

Realizada en: Computador A



Gráfica 6.1. Prueba 6 – Arreglos mitad

Prueba 6	**Arreglo mitad**						
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS	QS 3-way
4096	0,301	0,441	0,222	0,221	0,255	0,355	45,882
8192	0,65	0,956	0,448	0,519	0,527	0,828	185,089
16384	1,435	2,051	1,04	1,215	1,198	1,95	749,377
32768	3,049	4,442	2,222	2,978	2,923	4,421	2041,533
65536	6,897	9,699	4,864	7,679	7,563	10,523	12748,791

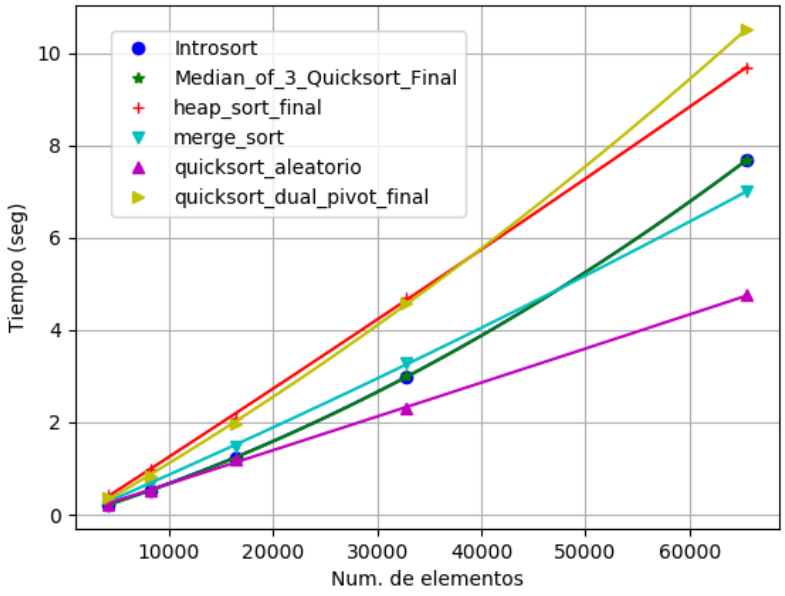
Tabla 6.1. Prueba 6– Arreglos mitad

Una vez efectuada la corrida de los algoritmos, se percibe que QTW fue el algoritmo con peor desempeoro, haciendo imposible el análisis de los tiempos del resto de los algoritmos, por lo que es necesario volver a ejecutar los algoritmos sin el mencionado anteriormente.

Estudiando el tiempo que tardó QTW en concretar el ordenamiento del arreglo, se deduce un análisis análogo a las pruebas anteriores en lo que respecta a los ciclos del código. En relación a la escogencia del pivote, ya que se seleccionan el primer y Último elemento del arreglo y debido a que ambos son 1 (por la forma de la secuencia), todos los elementos se colocan al final y se repite el proceso, lo cual lo hace un algoritmo sumamente lento para este tipo de arreglos.

Debido al mencionado crecimiento de QTW, la gráfica para la siguiente corrida es la mostrada a continuación:

- Prueba 6 sin QTW



Gráfica 6.2. Prueba 6 sin QTW– Arreglos mitad

Prueba 6	Arreglo mitad sin QS 3-way					
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS
4096	0,312	0,444	0,23	0,277	0,229	0,361
8192	0,671	0,971	0,519	0,517	0,521	0,842
16384	1,468	2,103	1,192	1,2	1,23	1,975
32768	3,284	4,707	2,297	3,001	2,996	4,585
65536	7,002	9,689	4,752	7,683	7,685	10,521

Tabla 6.2. Prueba 6 sin QTW– Arreglos mitad

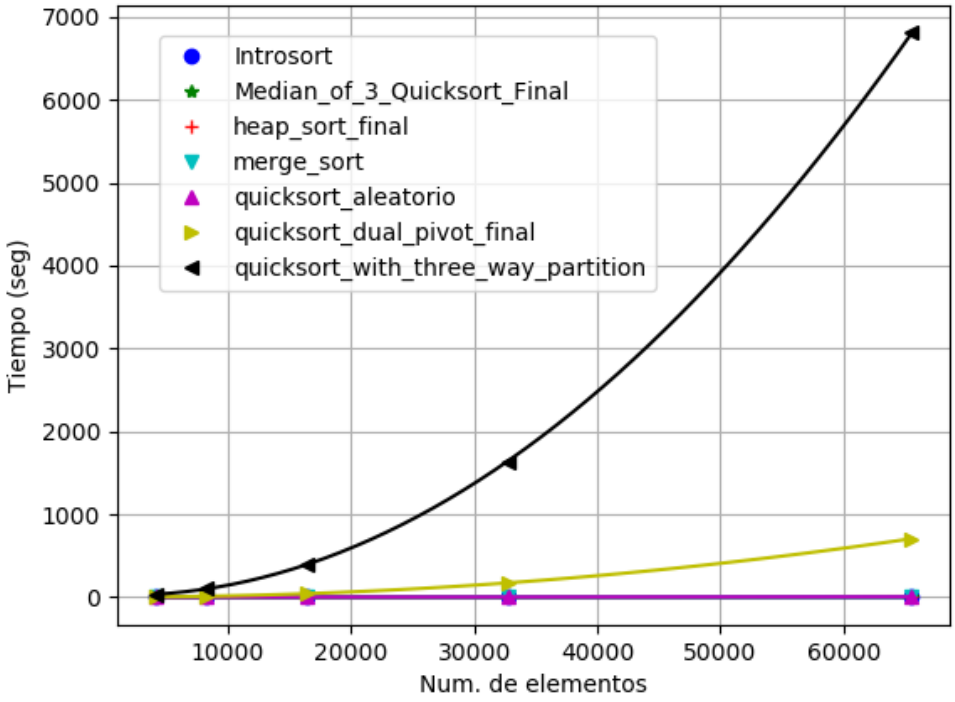
Ahora, se puede notar que HS y QDP poseen un crecimiento y tiempo parecido. Hasta alrededor de 37000 elementos, QDP es ligeramente más eficiente que HS, lo cual se debe a que para menos elementos, como QDP toma el primer y Último elemento (ambos 1) y los coloca al inicio del arreglo, y así sucesivamente con el resto de los elementos, por lo cual la cantidad de comparaciones es mucho menor. Sin embargo, a partir de 40000 elementos, su tiempo de corrida es mayor al de HS, lo cual se debe a la cantidad de elementos que debe organizar y a una mayor cantidad de operaciones. Por su parte, HS establece el heap máximo, lo cual es muy rápido pues algunos elementos se repiten debido a la forma del arreglo inicial. Luego, al momento de reestablecer el heap, no es necesario realizar una gran cantidad de comparaciones entre dichos elementos, por lo que su tiempo es menor al de QDP.

Luego, se compararon los algoritmos IS, M3Q y MS, los cuales tienen tiempos aproximados muy cercanos entre sí. Para IS y M3Q el desempeoro es el mismo, esto se debe a la manera en que se escogen los pivotes en ambos algoritmos, que es muy similar. Al relacionar estos tiempos con MS, se aprecia que los algoritmos anteriores fueron ligeramente más rápidos que MS hasta los 45000 elementos, lo que se debe a la manera de particionar los arreglos de la forma de la secuencia especificada. MS, por otro lado, al particionar el arreglo en mitades y luego unirlos, la primera mitad de la secuencia ya está ordenada, por lo que el algoritmo toma poco tiempo en culminar el ordenamiento.

Finalmente, el algoritmo más rápido fue QSA, debido a que la manera de seleccionar el pivote es aleatoria, por lo que, como se ha mencionado en análisis de las pruebas anteriores, la probabilidad de que las particiones desequilibradas es mucho menor pues el pivote se toma de manera aleatoria.

PRUEBA 7: Arreglo casi ordenado

Realizada en: Computador A



Gráfica 7.1. Prueba 7 – Arreglos casi ordenados

Prueba 7	**Arreglo casi ordenado**						
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS	QS 3-way
4096	0,272	0,385	0,205	0,056	0,054	2,883	26,779
8192	0,582	0,829	0,414	0,113	0,118	11,051	102,382
16384	1,25	1,807	0,88	0,238	0,239	42,688	399,679
32768	2,652	3,943	2,029	0,497	0,498	171,812	1637,928
65536	5,88	8..227	4,281	1,068	1,077	705,452	6804,517

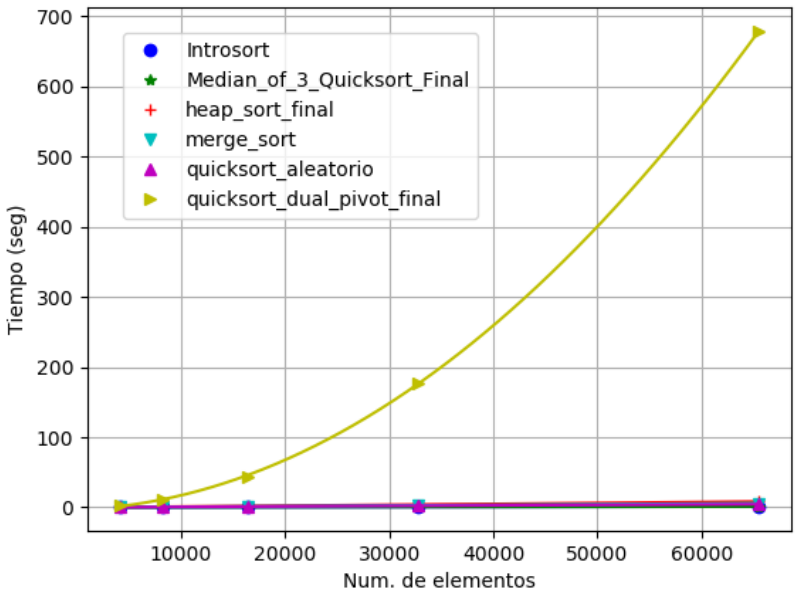
Tabla 7.1. Prueba 7 – Arreglos casi ordenados

Al ejecutar los algoritmos, observar la gráfica y comparar los tiempos mediante la tabla, es claro que el crecimiento de QTW es notablemente mayor a los demás algoritmos, aunque también se evidencia que es posible QDP presente un crecimiento asintótico mucho mayor a la representación del tiempo de los algoritmos que se encuentran en la parte baja de la gráfica.

El tiempo tomado por QTW, como se ha repetido en análisis anteriores, se debe a la forma del código, además de la cantidad de comparaciones que se deben realizar. Ya que el arreglo en esta prueba está casi ordenado y como se seleccionan el primer y Último elemento como pivote, es claro que los arreglos son muy desbalanceados en la mayoría de las llamadas recursivas del algoritmo.

Una vez realizada la prueba sin QTW, se obtuvo la siguiente gráfica:

- Prueba 7 sin QTW



Gráfica 7.2. Prueba 7 sin QTW – Arreglos casi ordenados

Prueba 7	**Arreglo casi ordenado sin QS 3-way**					
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort	Dual pivot QS
4096	0,267	0,381	0,198	0,055	0,056	2,747
8192	0,579	0,886	0,472	0,116	0,121	10,586
16384	1,332	1,806	1,004	0,255	0,247	43,701
32768	2,648	3,844	2,219	0,491	0,489	117,392
65536	5,824	8,249	4,37	1,038	1,103	679,732

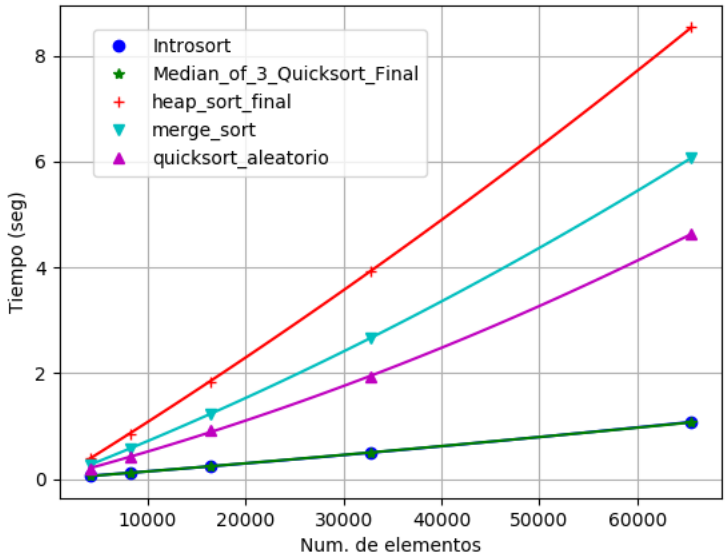
Tabla 7.2. Prueba 7 sin QTW – Arreglos casi ordenados

Como se expresó anteriormente, QDP tomo una mayor cantidad de tiempo al compararlo con el resto de los algoritmos, lo que hace, como en el caso de QTW, imposible analizar las gráficas de los algoritmos restantes. Esto amerita una nueva corrida de los algoritmos sin QTW ni QDP.

Para QDP, la razón por la cual el tiempo que toma es mayor al resto de los algoritmos (excluyendo QTW) se debe a que los pivotes que se consideran son el primero y el Último, que son (con gran probabilidad) el menor y el mayor de los elementos. Luego, al tomarlos pivotes auxiliares y realizar las llamadas recursivas, en cada subarreglo probablemente se hacen particiones de la misma manera desequilibrada, por lo cual el algoritmo tarda más.

La gráfica producida al repetir la prueba con los algoritmos restantes se muestra a continuación:

- Prueba 7 sin QTW y QDP



Gráfica 7.3. Prueba 7 sin QTW y QDP– Arreglos casi ordenados

Prueba 7	**Arreglo casi ordenado sin QS 3-way y Dual Pivot QS**				
N	Mergesort	Heapsort	QS Random	Med-of-3 QS	Introsort
4096	0,263	0,391	0,189	0,054	0,054
8192	0,575	0,855	0,413	0,114	0,117
16384	1,231	1,838	0,92	0,238	0,238
32768	2,662	3,936	1,926	0,5	0,495
65536	6,72	8,536	4,635	1,069	1,077

Tabla 7.3. Prueba 7 sin QTW y QDP– Arreglos casi ordenados

Lo primero que se puede notar es que HS fue el que tomo más tiempo en ordenar el arreglo casi ordenado. Esto se debe a que debe establecer el heap máximo, es decir ordenar todo el arreglo en forma de heap, y cada vez que se coloca un elemento al final se vuelve a establecer la propiedad del heap. Siguiendo con el análisis del algoritmo de MS, se observa que toma un poco de menos tiempo que HS, lo que es debido a que al ir dividiendo en mitades el arreglo, como ya casi está¡ ordenado, al unir las soluciones es mucho más rápido ordenarlo. El siguiente es QSA, que nuevamente tiene un tiempo de corrida óptimo pues el pivote se escoge al azar, por lo cual al particionar el arreglo, lo hace de manera más simétrica. Finalmente, M3Q e IS fueron los algoritmos más óptimos para este tipo de arreglo, con un tiempo menor a dos segundos. La razón para ambos es clara: se escoge la media entre el primer, Último y el elemento de la mitad. Como el arreglo está casi ordenado, es muy probable que se tome uno de los elementos que estará cerca de la mitad del arreglo totalmente ordenado, por lo cual las particiones son sumamente buenas, y los algoritmos son muy rápidos.

En general, los algoritmos de la Última gráfica son bastante óptimos para arreglos casi ordenados.

CONCLUSIONES

Luego de realizar los experimentos anteriores, es evidente que el algoritmo que toma más tiempo en general es QTW, como se observa en las pruebas 1, 2, 5, 6 y 7. Esto se debe a la forma del código y la cantidad de comparaciones que deben hacerse. Además en muchos casos, el pivote se escoge de manera que la partición es desbalanceada. Si bien en la mayoría de los casos es el más ineficiente, para la prueba del arreglo con ceros y unos fue el que tuvo mejor rendimiento. Por otra parte, el algoritmo más eficiente en la mayoría de los casos fue QSA, pues si bien no fue el más rápido siempre, en las pruebas 2, 5, 6 y 7 se evidencia su rapidez. En conclusión, y luego de los análisis anteriores, la rapidez de cada algoritmo queda determinada por la manera en que se escoge el pivote para cada caso (en el caso de los algoritmos basados en quicksort). En cuanto a MS y HS, son algoritmos asintóticamente óptimos que se implementan de la misma manera. Por lo tanto, para resolver diferentes problemas de ordenamiento de arreglos, lo ideal es estudiar la forma en la que se presenta el arreglo inicial para poner seleccionar el algoritmo más eficiente.