Universidad Simón Bólivar
Laboratorio de Algoritmos y Estructuras II
Nombres: Orlando Chaparro # 12-11499
Angel Morante #13-10931

# IINFORME PROYECTO 1

DATOS SOBRE EL EQUIPO DONDE SE REALIZAN LAS PRUEBAS:

MODELO: DELL XPS M1210

CPU: INTEL CORE DUO T2600 (2160 MHz)

MEMORIA RAM: 2 GB

SISTEMA OPERATIVO: UBUNTU 14.06 (DISTRIBUCÒN GNU LINUX)

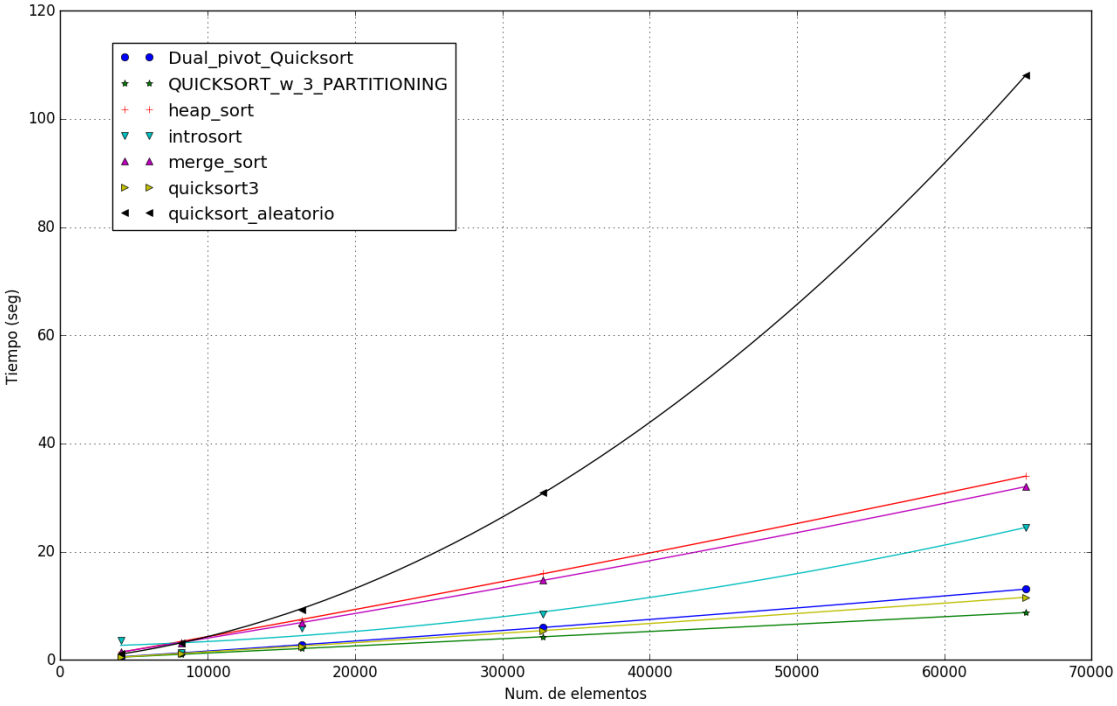
TABLAS PARA LAS PRUEBAS:

CONJUNTOS DE DATOS 1
(“Enteros positivos aleatorios desordenados donde el mayor valor de los enteros es 500”)
#Ejemplo de Llamada:

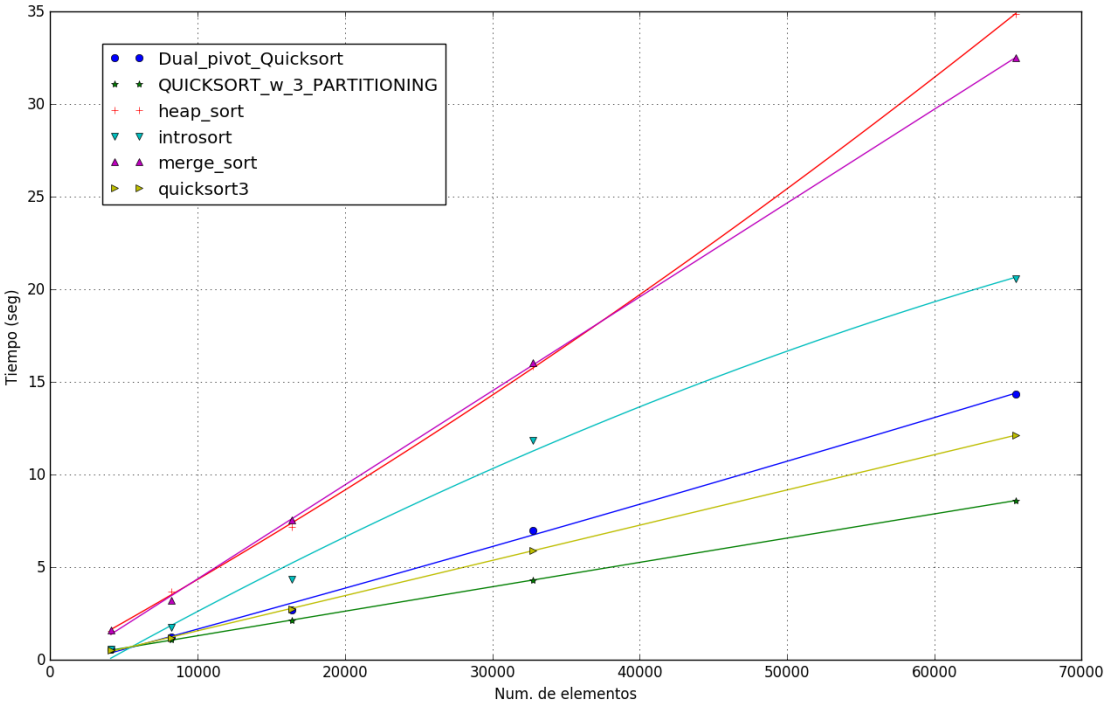
>> python3 cliente_ordenamiento.py -p 1 -g -t 3 4096 8192 16384 32768 65536

- TABLA 1:

| N | MERGESORT | HEAPSORT | QS RANDOM | MED-OF-3 QS | INTROSORT | DUAL PIVOT QS | QS 3-WAY |
|---|---|---|---|---|---|---|---|
| 4096 | 1.498 seg | 1.546 seg | 1.183 seg | 0.582 seg | 3.545 seg | 0.577 seg | 0.504 seg |
| 8192 | 3.190 seg | 3.344 seg | 3.089 seg | 1.131 seg | 1.322 seg | 1.258 seg | 1.021 seg |
| 16384 | 6.853 seg | 7.276 seg | 9.303 seg | 2.538 seg | 5.789 seg | 2.851 seg | 2.100 seg |
| 32768 | 14.735 seg | 16.059 seg | 30.918 seg | 5.445 seg | 8.485 seg | 5.967 seg | 4.272 seg |
| 65536 | 32.022 seg | 33.943 seg | 108.075 seg | 11.559 seg | 24.524 seg | 13.086 seg | 8.743 seg |




PRUEBA SIN FUNCION CUADRATICA:



RESULTADOS PRUEBA 1:

Un arreglo generado con enteros aleatorios es el caso mas común de ordenamiento, donde el desempeño de merge sort y heap sort es similiar en cada prueba, pero quicksort_with_3_partitioning es el algoritmo mas eficiente de todos para cada una de las pruebas.

Para esta prueba los algoritmos más eficientes fueron:
-Quicksort with 3 Partitioning
-Quicksort Med-of-3 QS

Y los algoritmos cuadraticos fueron:
-Quicksort Aleatorio

CONJUNTOS DE DATOS 2
(“Orden Inverso”)

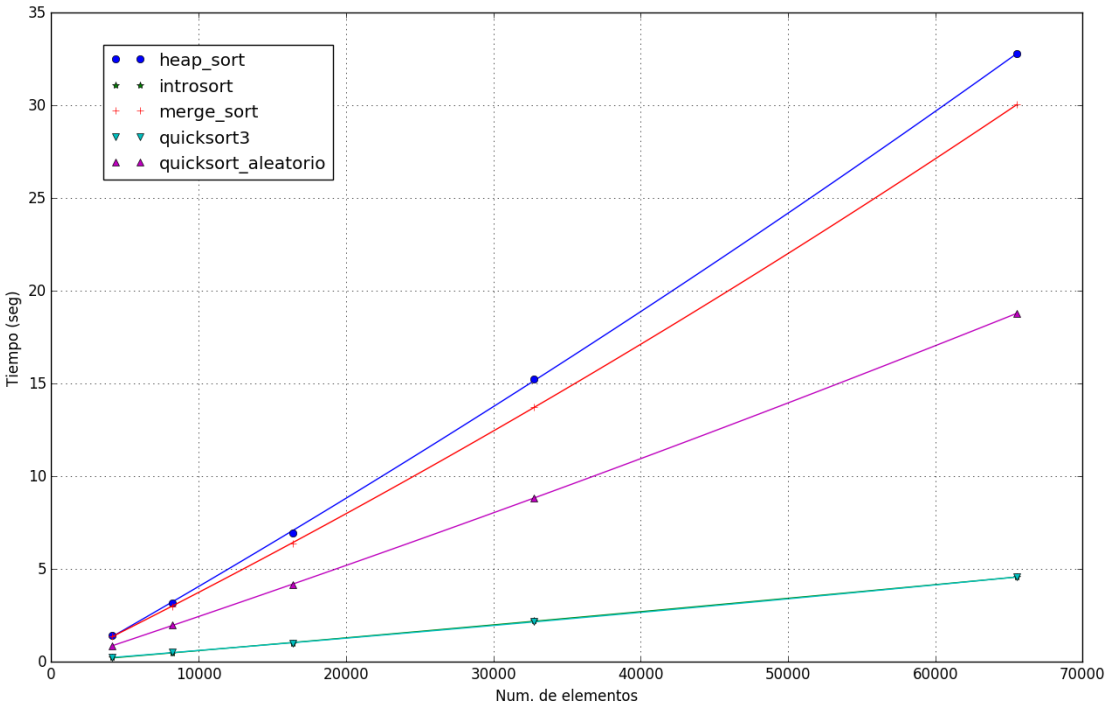
#Ejemplo de Llamada:

>> python3 cliente_ordenamiento.py -p 2 -g -t 3 4096 8192 16384 32768 65536

- TABLA 2:

| N | MERGESORT | HEAPSORT | QS RANDOM | MED-OF-3 QS | INTROSORT | DUAL PIVOT QS | QS 3-WAY |
|---|---|---|---|---|---|---|---|
| 4096 | 1.382 seg | 1.455 seg | 0.834 seg | 0.228 seg | 0.230 seg | 27.332 seg | 28.793 seg |
| 8192 | 2.938 seg | 3.169 seg | 2.017 seg | 0.477 seg | 0.475 seg | 110.135 seg | 113.690 seg |
| 16384 | 6.447 seg | 6.985 seg | 4.125 seg | 1.027 seg | 1.034 seg | 439.038 seg | 454.210 seg |
| 32768 | 13.899 segs | 15.322 segs | 8.795 segs | 2.171 segs | 2.158 segs | 1794.088 segs | 1882.525 segs |
| 65536 | 30.039 segs | 32.784 segs | 18.788 segs | 4.576 segs | 4.573 segs | 7110.739 segs | _ |

PRUEBA SIN FUNCION CUADRATICA




RESULTADOS OBTENIDOS CON LA PRUEBA 2:

EXPLICACION: Al momento de intentar determinar el tiempo de corrida de los algoritmos de ordenamiento, nos encontramos con que las pruebas no se pudieron realizar correctamente, debido a que el procesador (core) se sobrecargó. Para ser más concretos, el error se produjo al intentar correr el algoritmo "Quicksort with 3-way partitioning" al utilizar la cantidad propuesta de 65536 elementos, razón por la cual no obtuvimos la gráfica de todos los algoritmos juntos. Sin embargo, omitiendo este contratiempo determinamos el tiempo de corrida de todos los algoritmos restantes para determinar que aquellos algoritmos que corren en tiempo cuadratico son "Quicksort with 3-way partitioning" y "Dual pivot Quicksort"

Descartando los algoritmos antes mencionados (que corren en tiempo cuadrático), construimos la grafica que se pide en el enunciado del proyecto.

Para esta prueba los algoritmos más eficientes fueron:

-Introsort
-Quicksort Med-of-3 QS

Y los algoritmos cuadraticos fueron:

-Dual pivot Quicksort
-Quicksort with 3 Partitioning

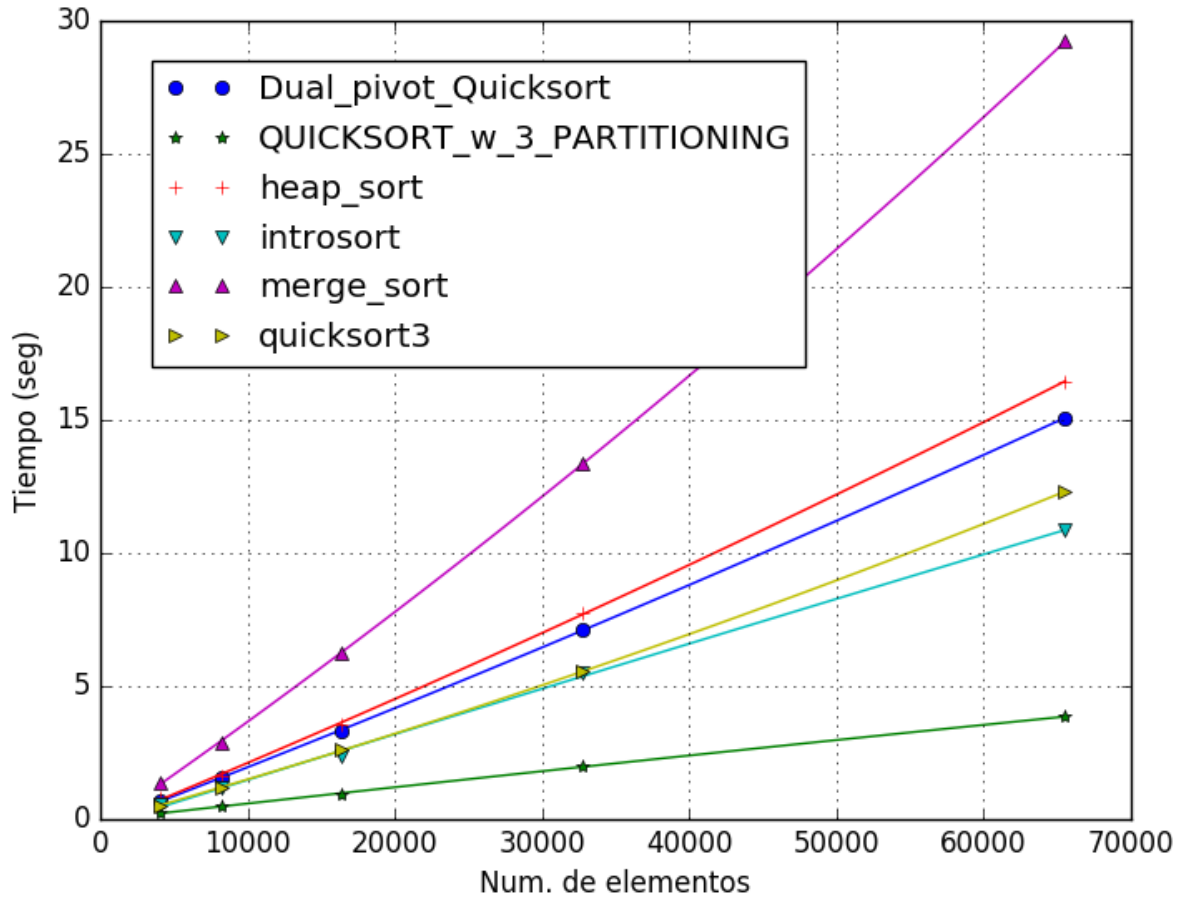
CONJUNTOS DE DATOS 3
(“Arreglo de ceros y unos aleatorios”)
#Ejemplo de Llamada:

>> python3 cliente_ordenamiento.py -p 3 -g -t 3 4096 8192 16384 32768 65536

- TABLA 3:

| N | MERGESORT | HEAPSORT | QS RANDOM | MED-OF-3 QS | INTROSORT | DUAL PIVOT QS | QS 3-WAY |
|---|---|---|---|---|---|---|---|
| 4096 | 1.368 segs | 0.811 segs | 92.639 segs | 0.535 segs | 0.596 segs | 0.695 segs | 0.239 segs |
| 8192 | 2.907 segs | 1.649 segs | 382.124 segs | 1.183 segs | 1.183 segs | 1.495 segs | 0.468 segs |
| 16384 | 6.299 segs | 3.599 segs | 1477.313 segs | 2.617 segs | 2.375 segs | 3.253 segs | 0.946 segs |
| 32768 | 13.639 segs | 7.793 segs | _ | 5.703 segs | 5.643 segs | 7.201 segs | 1.933 segs |
| 65536 | 29.479 segs | 17.265 segs | _ | 12.376 segs | 11.725 segs | 16.191 segs | 4.042 segs |

GRAFICA SIN FUNCIONES CUADRATICAS:



RESULTADOS OBTENIDOS CON LA PRUEBA 3:

EXPLICACION: De igual forma como sucedio con el intento de prueba nº 2, el procesador colapsó al intentar correr el algoritmo "quicksort aleatorio" que para este caso resulto ser de forma cuadratica.

Descartando los algoritmos antes mencionados (que corren en tiempo cuadrático), construimos la grafica que se pide en el enunciado del proyecto.

Para esta prueba los algoritmos más eficientes fueron:

- -Quicksort with 3 Partitioning
- -Introsort
- -Quicksort Med-of-3 QS

Y los algoritmos cuadraticos fueron:

- -Quicksort Aleatorio

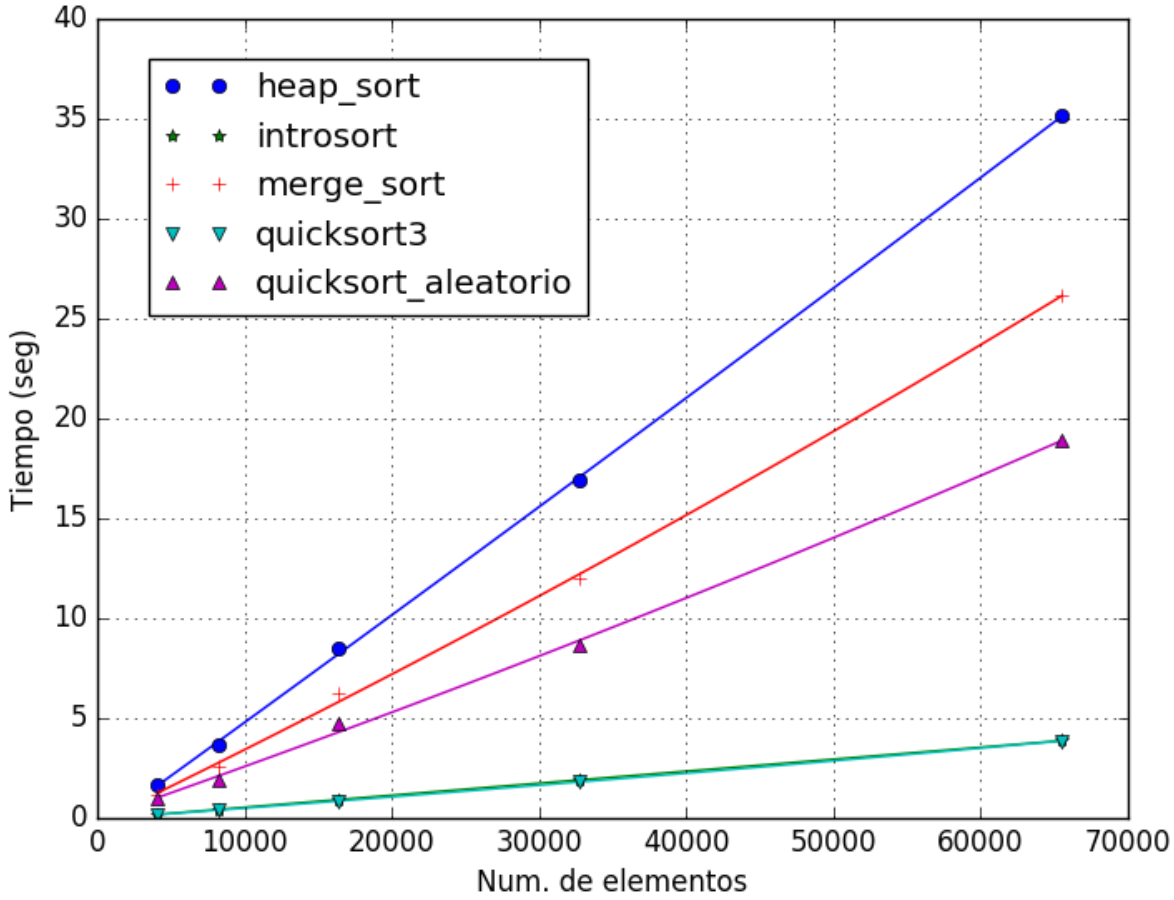
CONJUNTOS DE DATOS 4
(“Enteros positivos ordenados”)

>> python3 cliente_ordenamiento.py -p 4 -g -t 3 4096 8192 16384 32768 65536

- TABLA 4:

| N | MERGESORT | HEAPSORT | QS RANDOM | MED-OF-3 QS | INTROSORT | DUAL PIVOT QS | QS 3-WAY |
|---|---|---|---|---|---|---|---|
| 4096 | 1.181 segs | 1.575 segs | 0.905 segs | 0.183 segs | 0.183 segs | 27.348 segs | 27.141 segs |
| 8192 | 2.556 segs | 3.436 segs | 1.778 segs | 0.391 segs | 0.391 segs | 107.226 segs | 107.389 segs |
| 16384 | 5.474 segs | 7.402 segs | 4.251 segs | 0.834 segs | 0.832 segs | 428.779 segs | 428.243 segs |
| 32768 | 11.902 segs | 16.055 segs | 8.313 segs | 1.781 segs | 1.779 segs | 1718.853 segs | 1717.746 segs |
| 65536 | 26.189 segs | 35.165 segs | 18.942 segs | 3.863 segs | 3.876 segs | 6908.514 segs | |

GRAFICA SIN FUNCIONES CUADRATICAS:




RESULTADOS DE LA PRUEBA:

En el caso de la prueba con la mayor cantidad de elementos se volvio a presentar el problema que se presentó en la prueba dos, donde el proceso no se pudo completar dado que el computador no pudo realizar la tarea. Sin embargo se observa en la gráfica, donde no esta presente Quicksort 3way, dado que corre en tiempo cuadratico, que Quicksort mediana de tres sigue siendo el mas efectivo sin embargo en esta prueba introsort es casi exacto a quicksort mediana de tres.

Para esta prueba los algoritmos más eficientes fueron:

-Introsort
-Quicksort Med-of-3 QS

Y los algoritmos cuadraticos fueron:

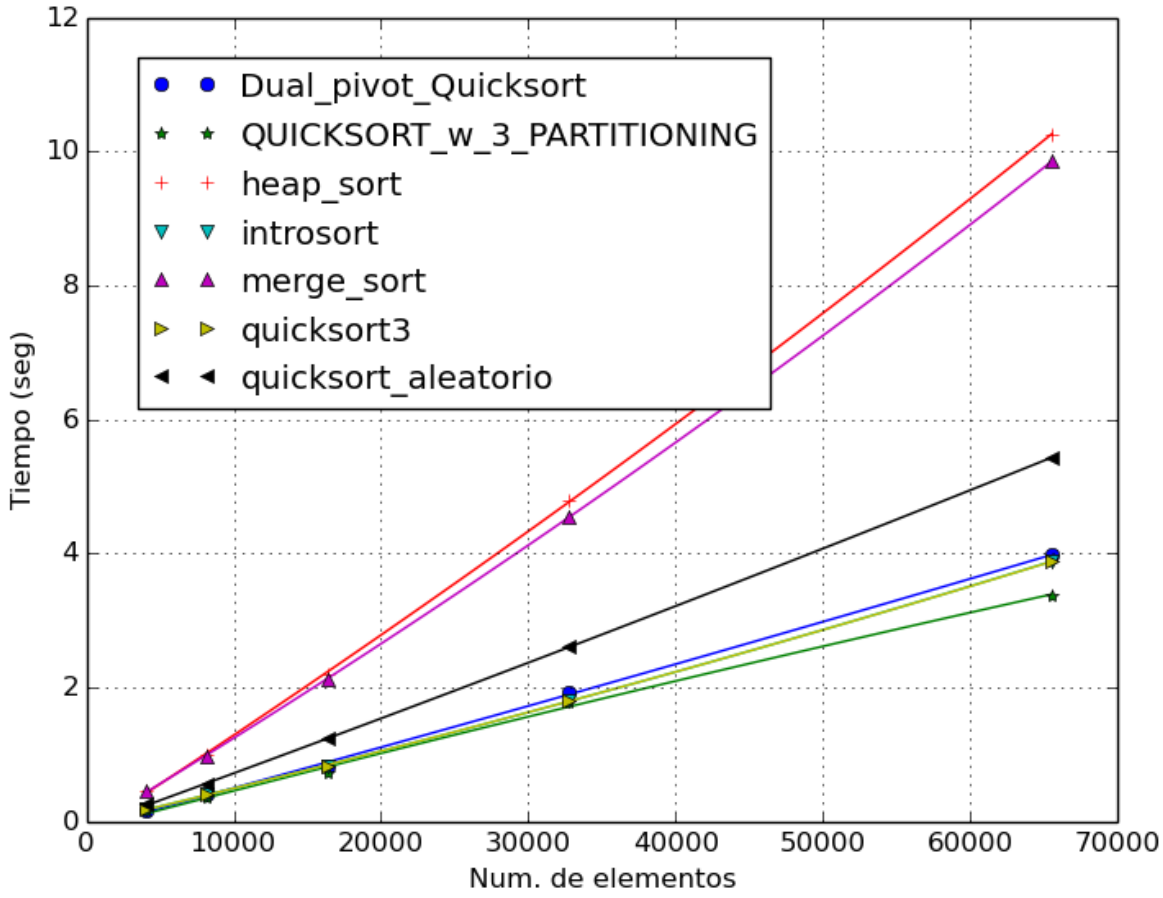
-Quicksort with 3 Partitioning
-Dual Pivot Quicksort

CONJUNTOS DE DATOS 5
(“Reales aleatorios en el intervalo [0,1)”)
#Ejemplo de Llamada:

>> python3 cliente_ordenamiento.py -p 5 -g -t 3 4096 8192 16384 32768 65536

- TABLA 5:

| N | MERGESORT | HEAPSORT | QS RANDOM | MED-OF-3 QS | INTROSORT | DUAL PIVOT QS | QS 3-WAY |
|---|---|---|---|---|---|---|---|
| 4096 | 0.509 segs | 0.516 segs | 0.294 segs | 0.201 segs | 0.197 segs | 0.206 segs | 0.183 segs |
| 8192 | 1.106 segs | 1.132 segs | 0.610 segs | 0.442 segs | 0.430 segs | 0.430 segs | 0.382 segs |
| 16384 | 2.352 segs | 2.475 segs | 1.405 segs | 0.930 segs | 0.929 segs | 0.884 segs | 0.828 segs |
| 32768 | 5.186 segs | 5.366 segs | 2.811 segs | 2.020 segs | 2.013 segs | 1.942 segs | 1.793 segs |
| 65536 | 11.041 segs | 11.571 segs | 6.193 segs | 4.406 segs | 4.419 segs | 4.332 segs | 3.812 segs |




RESULTADOS OBTENIDOS CON LA PRUEBA 5:

EXPLICACION: Debido a que al correr los algoritmos de ordenamiento con los arreglos correspondientes a la prueba 5 no hubo ningun algoritmo que corriera en tiempo cuadratico, no se realizo la grafica en la que se supone deberian estar unicamente los algorimos que no corran en tiempo cuadratico.

Para esta prueba los algoritmos más eficientes fueron:

-Quicksort with 3 Partitioning
-Dual Pivot Quicksort
-Quicksort Med-of-3 QS

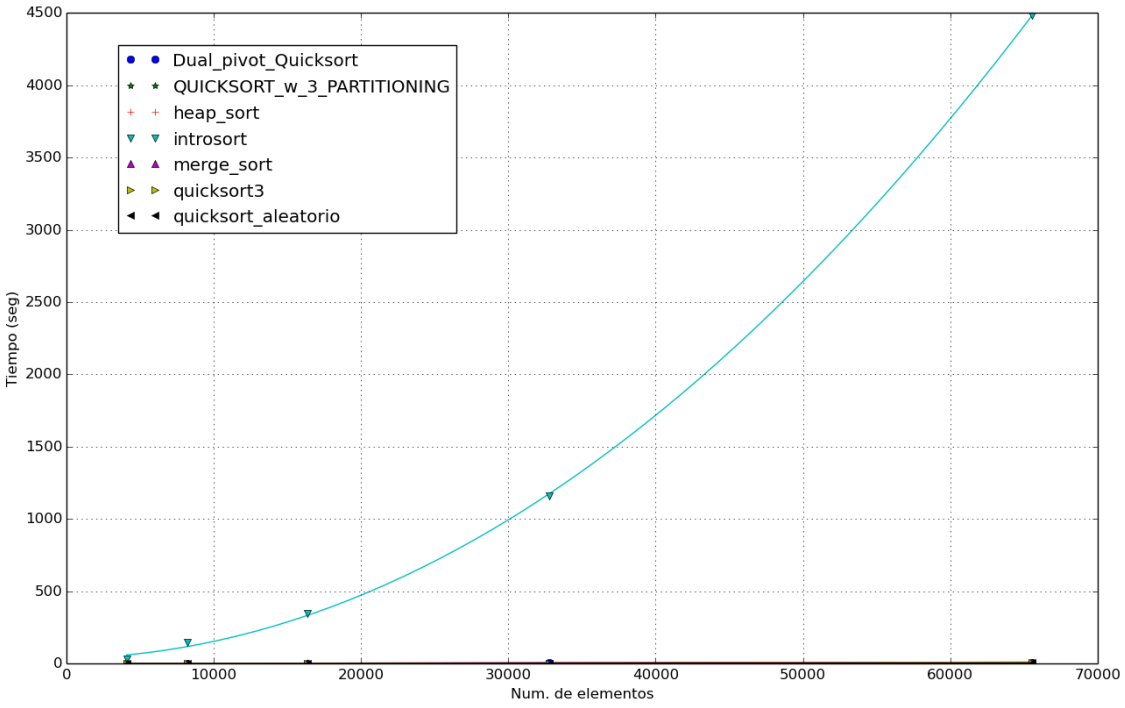
Y no se registraron algoritmos cuadraticos.

CONJUNTOS DE DATOS 6
(“Arreglo de enteros positivos aleatorios de tamaño N, donde los elementos estan ordenados desde 0 a N/2 y luego de N/2 hasta 0)
#Ejemplo de Llamada:

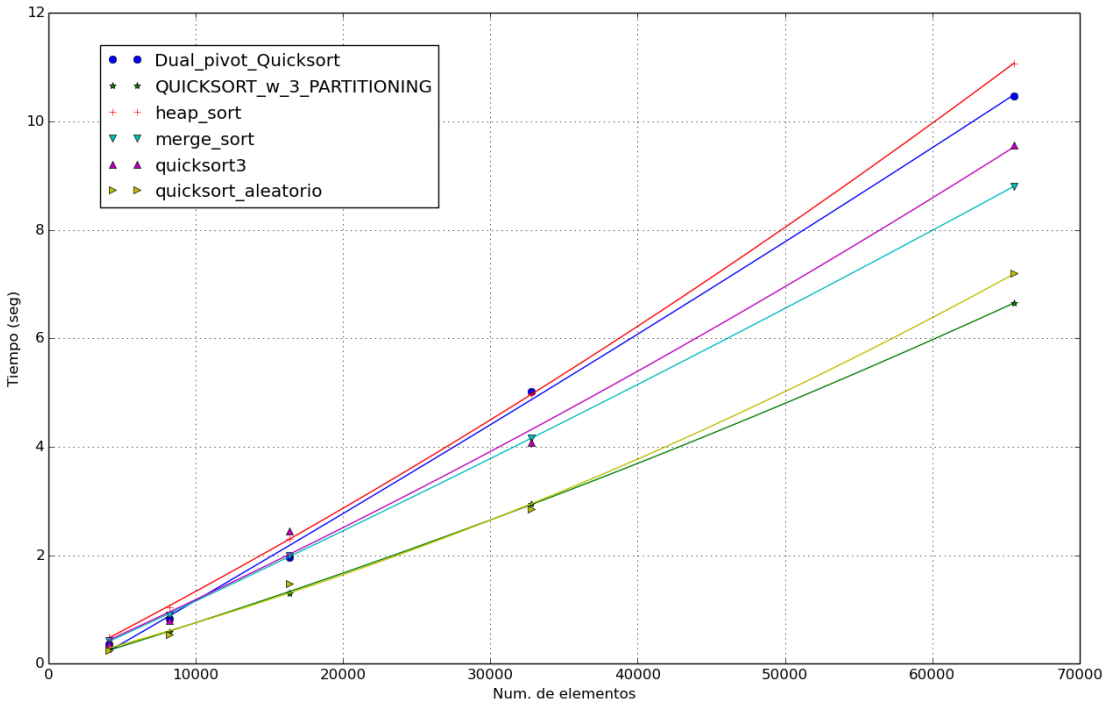
>> python3 cliente_ordenamiento.py -p 6 -g -t 3 4096 8192 16384 32768 65536

- TABLA 6:

| N | MERGESORT | HEAPSORT | QS RANDOM | MED-OF-3 QS | INTROSORT | DUAL PIVOT QS | QS 3-WAY |
|---|---|---|---|---|---|---|---|
| 4096 | 0.425 segs | 0.499 segs | 0.274 segs | 0.374 segs | 17.113 segs | 0.366 segs | 0.267 segs |
| 8192 | 0.904 segs | 1.090 segs | 0.552 segs | 0.840 segs | 70.247 segs | 0.842 segs | 0.600 segs |
| 16384 | 1.948 segs | 2.324 segs | 1.231 segs | 1.897 segs | 279.298 segs | 1.974 segs | 1.328 segs |
| 32768 | 4.178 segs | 5.062 segs | 2.852 segs | 4.292 segs | 1148.949 segs | 4.552 segs | 2.979 segs |
| 65536 | 8.801 segs | 10.842 segs | 7.200 segs | 9.561 segs | 4482.336 segs | 10.430 segs | 6.505 segs |



GRAFICA SIN FUNCIONES CUADRATICAS:




Para esta prueba los algoritmos más eficientes fueron:

-Quicksort with 3 Partitioning

-Quicksort Aleatorio

Y los algoritmos cuadráticos fueron:

-Introsort

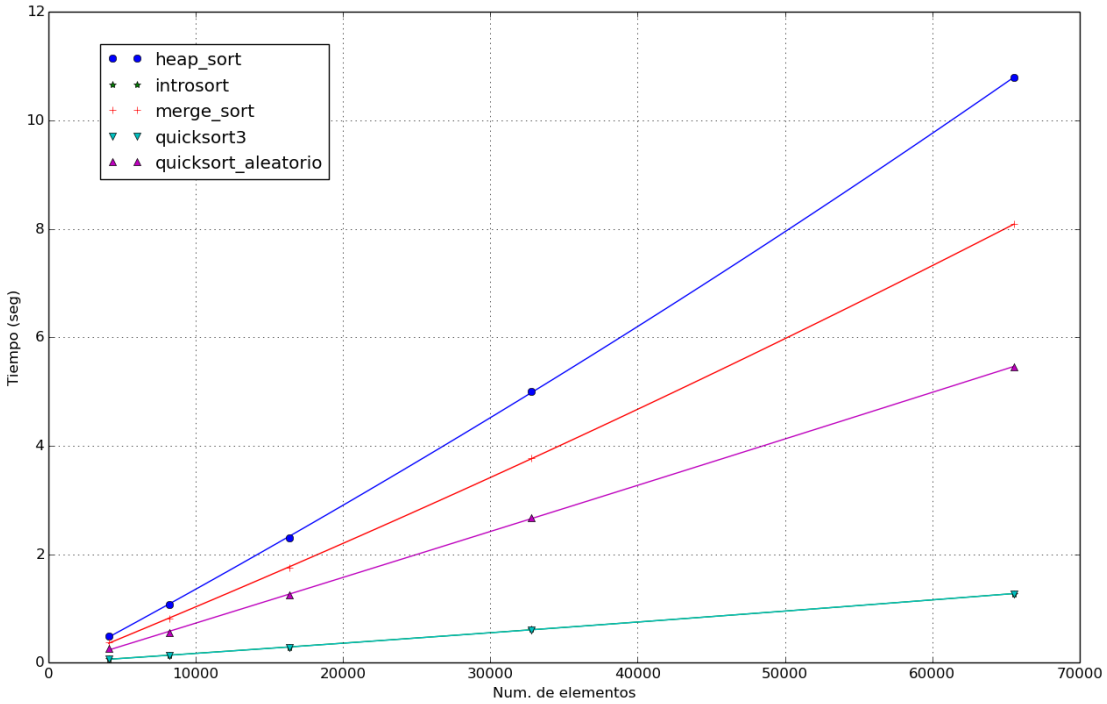
CONJUNTOS DE DATOS 7
(“Casi ordenado Dado un conjunto ordenado de N elementos de tipo entero, se escogen al azar n/4 pares de elementos que se encuentran separados 4 lugares, entonces se intercambian los pares.”)

#Ejemplo de Llamada:

>> python3 cliente_ordenamiento.py -p 7 -g -t 3 4096 8192 16384 32768 65536

- TABLA 7:

| N | MERGESORT | HEAPSORT | QS RANDOM | MED-OF-3 QS | INTROSORT | DUAL PIVOT QS | QS 3-WAY |
|---|---|---|---|---|---|---|---|
| 4096 | 0.383 segs | 0.523 segs | 0.263 segs | 0.068 segs | 0.079 segs | 6.173 segs | 6.472 segs |
| 8192 | 0.814 segs | 1.067 segs | 1.002 segs | 0.133 segs | 0.132 segs | 25.408 segs | 28.361 segs |
| 16384 | 1.737 segs | 2.319 segs | 1.206 segs | 0.283 segs | 0.282 segs | 103.543 segs | 100.710 segs |
| 32768 | 3.773 segs | 4.940 segs | 2.650 segs | 0.588 segs | 0.597 segs | 392.813 segs | 410.093 segs |
| 65536 | 7.994 segs | 10.336 segs | 5.328 segs | 1.224 segs | 1.232 segs | 1559.346 segs | _ |




RESULTADOS OBTENIDOS CON LA PRUEBA 7:

EXPLICACION: Análogo al caso (2)

Para esta prueba los algoritmos más eficientes fueron:

-Introsort
-Quicksort Med-of-3 QS

Algoritmos que corren en tiempo cuadratico:

- Dual Pivot Quicksort
- Quicksort with 3 Partitioning

Conclusiones generales:

Aunque se evaluaron los algoritmos en sus peores y mejores casos para arreglos de diferentes tamaños, queda claro luego de todas las pruebas que Quicksort Mediana de tres es el mas eficiente y óptimo, sin embargo introsort presente un comportamiento similiar, pero la constanto escondido en Quicksort mediana de tres es un poco menor a introsort, por lo que dicho algoritmos es mejor para mayor optimalidad.