ALGORITMOS Y COSTO COMPUTACIONAL

Luis Felipe Sante Taipe, Maryori Lizeth Hilares Angelo, Rodrigo Santiesteban Pachari
Escuela Profesional de Ciencias de la Computación, Universidad Nacional de San Agustín
Arequipa, Perú
lsantet@unsa.edu.pe
mhilaresa@unsa.edu.pe
rsantisteban@unsa.edu.pe

Abstract
In order to give a better understanding of the Sorting Algorithms a comparison will be made between the different algorithms and their computational costs, and how efficient they end up being, and their behavior when compiled in different Programming Languages such as C/C++, Python and Java, for a better understanding the comparison will be made in a line graph as they increase their times as the number of random data increases, one more than others where the efficiency of these will be seen.
Keywords*: bubble sort, insertion sort, heap sort, selection sort, merge sort, quick sort, shell sort, efficient, computational cost,*

Resumen
A fin de dar un mejor entendimiento sobre los Algoritmos de Ordenación se dará un comparación entre los diversos algoritmos y sus costos computacionales, y cuan eficiente estos terminan siendo, y su comportamiento al momento de ser compilados en diferentes Lenguajes de Programación como lo son C/C++, Python y Java, para una mejor comprensión se hará la comparación en un gráfico de líneas como estos aumentan sus tiempos a medida que el aumenta el número de datos aleatorios, uno más que otros donde se verá la eficiencia de estos.
Palabras clave*: bubble sort, insertion sort, heap sort, selection sort, merge sort, quick sort, shell sort, eficiente, costo computacional,*

I. Introducción

Debido a que las estructuras de datos son utilizadas para almacenar información, para poder recuperar esa información de manera eficiente es deseable que aquella esté ordenada. Existen varios métodos para ordenar las diferentes estructuras de datos básicas.

En general los métodos de ordenamiento no son utilizados con frecuencia, en algunos casos sólo una vez. Hay métodos muy simples de implementar que son útiles en los casos en dónde el número de elementos a ordenar no es muy grande (ej, menos de 500 elementos). Por otro lado hay métodos sofisticados, más difíciles de implementar pero que son más eficientes en cuestión de tiempo de ejecución.

Implementar estos métodos, no suele ser tan complicado.Sin embargo debemos tener muy en cuenta el lenguaje en que será implementado, por ejemplo, C++ es un lenguaje de alto nivel pero también puede comportarse como un lenguaje de bajo nivel, es por ello que se puede aprovechar esta ventaja para un mejor rendimiento de los algoritmos. Por otro lado, implementar estos algoritmos en Java es más complicado ya que al ser un lenguaje de Programación Orientado a Objetos en su totalidad, se

requiere la creación de objetos para poder implementar estos algoritmos, sin embargo todo ello no quiere decir que el tiempo de ejecución y complejidad se incrementen, ya que al ser POO se convierte en algo más eficaz. En cambio en Python, al ser este un lenguaje multiplataforma, puede ser compilado tanto en Linux como en Windows.

Los objetivos que se tiene con este artículo son:

- A pesar de que algunos algoritmos sean muy eficientes, siempre va acompañado de la arquitectura del computador para notar si existe un mejoramiento en el rendimiento.
- Dar a conocer que como de igual manera la arquitectura interviene mucho, queremos dar a conocer que los resultados pueden mejorar depende del compilador que se esté usando
- Las verdaderas diferencias entre el costo computacional entre los algoritmos se nota a partir de un vector de gran tamaño(N), ya que si se hacen pruebas con un N = 10, se diría que a pesar de que un algoritmo sea de complejidad O(n^2) y O(n log n) no hay diferencias.

En el presente trabajo se realizará el análisis de los diversos algoritmos como lo son Bubble Sort, Insertion Sort, Heap Sort, Selection Sort, Merge Sort, Quick Sort, Shellsort, su comportamiento en los diferentes lenguajes de programación[II], luego de la evaluación se darán las conclusiones para ver si se llegaron a completar los objetivos previstos en este artículo[III], adicionamos una pequeña discusión sobre porque a veces podemos preferir un algoritmo sobre otro aunque uno tengo mejor eficiencia al momento de ordenar.

II. Algoritmos de Ordenación y Complejidad Computacional

En las matemáticas y la computación un algoritmo de ordenamiento es un algoritmo que pone elementos de una lista o un vector en una secuencia dada por una relación de orden, es decir, el resultado de salida ha de ser una permutación o reordenamiento de la entrada que satisfaga la relación de orden dada.

Podemos tocar el tema de complejidad computacional, estos experimentos de complejidad computacional podemos hacerlos en el mejor caso, peor de los casos y caso promedio, en términos del tamaño del vector(N). Para esto se usa el concepto de orden de una función y se usa la notación O(n). El mejor comportamiento para ordenar es O(n log n). Los algoritmos más simples son cuadráticos, es decir $O(n^2)$.

Existen demasiados algoritmos de ordenación pero esta vez se hará la evaluación de 7 algoritmos de ordenamiento: Bubble Sort, Insertion Sort, Heap Sort, Selection Sort, Merge Sort, Quick Sort, Shellsort.

Se realizó experimentos en vectores con elementos aleatorios de tamaño 100000 ya que al probar con vectores de gran tamaño se puede visualizar la gran diferencia que hay entre ciertos algoritmos de complejidad O(n) y O(n log n), como lo son Bubble Sort y Merge Sort.

Realizamos experimentos este experimento en 3 lenguajes de programación los cuales son C/C++ , Python y Java, cabe resaltar que estos 3 lenguajes son de alto nivel rescatando que C/C++ también tiene la capacidad de que el programador tenga acceso a una programación de bajo nivel, algo que nos ayudará demasiado al momento de obtener resultados.

- Bubble Sort: El ordenamiento burbuja funciona revisando cada elemento del vector de elementos, si encuentra un elemento mal ubicado entonces hará un intercambio y si sigue estando en un lugar

equivocado seguirá intercambiando hasta que el vector de elementos esté ordenado, tiene una complejidad de O(n^2).

- Heap Sort: Es una técnica de clasificación basada en comparaciones basada en la estructura de datos de montón binario. Es similar al Selection Sort donde primero encontramos el elemento mínimo y colocamos el elemento mínimo al principio. Repetimos el mismo proceso para el resto de elementos, tiene una complejidad de O(n log n).

- Insertion Sort: Es un algoritmo de clasificación simple que funciona de manera similar a la forma en que clasifica las cartas de juego en sus manos. La matriz se divide virtualmente en una parte ordenada y otra sin clasificar. Los valores de la parte sin clasificar se seleccionan y se colocan en la posición correcta en la parte clasificada y tiene una complejidad de O(n^2).

- Selection Sort: Ordena una vector encontrando repetidamente el elemento mínimo (considerando el orden ascendente) de la parte no ordenada y colocándolo al principio. El algoritmo mantiene dos submatrices en una matriz determinada, primero se tiene un subarreglo que ya está ordenado, luego se tiene un subarreglo restante que no está clasificado. En cada iteración del ordenamiento por selección, el elemento mínimo (considerando el orden ascendente) del subarreglo sin clasificar se elige y se mueve al subarreglo ordenado, tiene una complejidad de O(n^2).

- Shellsort: Tiene este nombre gracias a su creador Donald Shell, está es una variación del Insertion Sort(movimiento de elementos adelante), en el caso del ShellSort se realiza muchos más movimientos ya que se quiere hacer el intercambio con elementos que se encuentran mucho más a la izquierda, este tiene una complejidad de O(n log^2 n).

- Merge Sort: Es un tipo de algoritmo divide and conquer, el objetivo de este algoritmo es dividir al vector en partes iguales y luego esas mitades las vuelve a dividir, luego une mitades pero ya de manera ordenada, tiene una complejidad O(n log n) en el mejor de los casos y en el peor O(n^2).

- Quicksort: Este es otro tipo de algoritmo divide and conquer, elige un valor pivote en la mayoría de los casos se recomienda usar el último elemento del vector como el elemento pivote y a partir de este empezar a fraccionar el vector, tiene una complejidad de O(n log n)

Algoritmo	Complejidad	Método
Bubble Sort	O(n^2)	Intercambio
Heap Sort	O(n log n)	Selección
Insertion Sort	O(n^2)	Inserción
Selection Sort	O(n^2)	Selección
Shellsort	O(n log^2 n)	Inserción

Merge Sort	O(n log n)	Mezcla
Quick Sort	Promedio: O(n log n) Peor caso: O(n²)	Partición

Tabla 1. Complejidad de Algoritmos de Ordenación

A. C/C++

Hicimos una prueba en el lenguaje C/C++, se realizaron pruebas en un vector de elementos aleatorios de tamaño 100000(0 <= N = 100000); podemos inferir de Imagen 1 los algoritmos de Bubble Sort, Insertion Sort y Selection Sort son los más pesados por lo que al momento de compilar estos algoritmos podemos encontrarnos con la sorpresa que estos demoren en acabar, ya que estos algoritmos son de complejidad cuadrática en el caso del Bubble Sort, Insertion Sort y Selection Sort-O(n^2), si queremos ver cuál es el algoritmo más pesado entre estos tres vemos que es Bubble Sort la razón de esto es que para ordenar estos datos se ordenan uno por uno hasta que encuentre un lugar correcto en el orden del vector.

Por otro lado los demás algoritmos como Heap Sort-O(n log n), Shell Sort-O(n log^2 n), Merge Sort-O(n log n), Quick Sort -O(log n), son los algoritmos más eficientes ya que al analizar 100000 datos estos no superan los 40000 nanosegundos, la razón es porque en el caso de Heap Sort es un algoritmo basado en montículos, es decir, lo podemos definir como un árbol binario completo y al tener estas características la complejidad se reduce a una logarítmica; en el caso de Shell Sort se considera la generalización del algoritmo de Bubble Sort o un algoritmo Insertion Sort optimizado, en el caso del Merge Sort es un algoritmo recursivo por lo que la complejidad del tiempo se puede expresar en T (n) = 2T (n / 2) + θ (n), si se resuelve llegamos a una complejidad logarítmica, con el Quick Sort de igual manera es un algoritmo recursivo se obtiene la siguiente ecuación recursiva T (n) = T (k) + T (nk-1) + \ θ (n)

Nanosegundos	BubbleSort	HeapSort	InsertionSort	SelectionSort	ShellSort	MergeSort	QuickSort
10000	449219	2678	69159	131039	1985	1575	1642
20000	1,43E+06	6066	282810	528107	4675	3347	3464
30000	3,61E+06	9399	715448	1,34E+06	6919	5310	5676
40000	6,23E+06	12517	1,48E+06	2,38E+06	9982	7035	9750
50000	9,88E+06	16485	1,87E+06	3,76E+06	13050	9403	14325
60000	2,15E+07	30038	3,08E+06	5,71E+06	19285	14916	15584
70000	1,96E+07	30806	3,79E+06	7,32E+06	19003	13480	14860
80000	2,69E+07	34030	5,02E+06	9,16E+06	22608	17551	17898
90000	3,34E+07	32043	6,89E+06	1,10E+07	26914	17473	19086
100000	4,26E+07	36323	8,12E+06	1,40E+07	29327	19406	21212

Imagen 1. Costo Computacional de los Algoritmos de Ordenación

Los algoritmos de Bubble Sort, Insertion Sort, Selection Sort al ser los algoritmos más pesados tomen mucha más fuerza para crecer, tanto es su peso que opacan a los 4 algoritmos más eficientes, como notamos en Imagen 2 donde el algoritmo que más pesa es el de Bubble Sort.

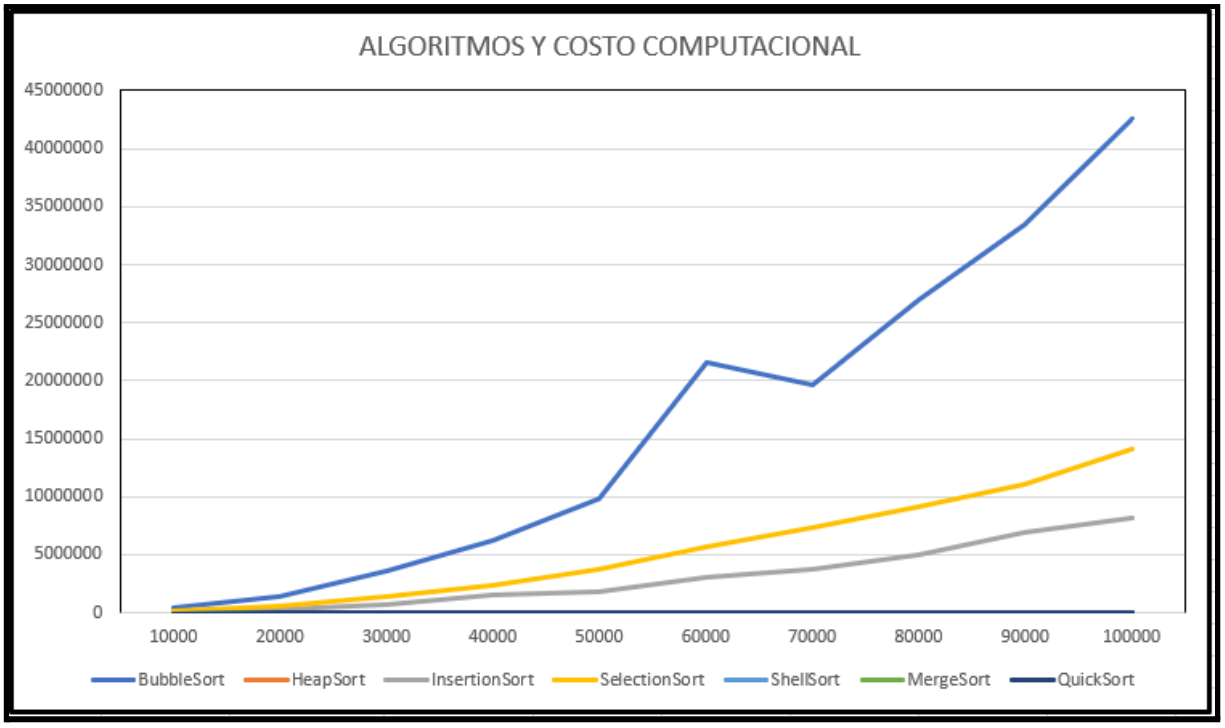

Imagen 2. Gráfica de líneas de los Algoritmos de Ordenación

Para analizar mejor los algoritmos que no se notan en Imagen 2, haremos un análisis independiente de estos 4 algoritmos(Heap Sort, Shell Sort, Merge Sort, Quick Sort), y veremos que el algoritmo más eficiente es el de Merge Sort - Imagen 3, pero este siempre es acompañado de muy cerca por el Quick Sort, esto se debe a que estos algoritmos se basan en recursividad y es un algoritmo de "Divide and Conquer". Este análisis será mucho más notable en Imagen 4, donde se notará que aunque existan algoritmos muy eficientes, siempre existe uno mejor que los demás.

Nanosegundos	HeapSort	ShellSort	MergeSort	QuickSort
10000	2678	1985	1575	1642
20000	6066	4675	3347	3464
30000	9399	6919	5310	5676
40000	12517	9982	7035	9750
50000	16485	13050	9403	14325
60000	30038	19285	14916	15584
70000	30806	19003	13480	14860
80000	34030	22608	17551	17898
90000	32043	26914	17473	19086
100000	36323	29327	19406	21212

Imagen 3. Costo Computacional de los Algoritmos de Ordenación

- Merge Sort es el algoritmo más eficiente que los demás algoritmos
- Heap Sort aunque se base en un algoritmo de montículos es el más pesado de los 4
- Quick Sort y Merge Sort al basarse en algoritmos recursivos estos son los más eficientes ya que estos se basan en el particionamiento de los vectores

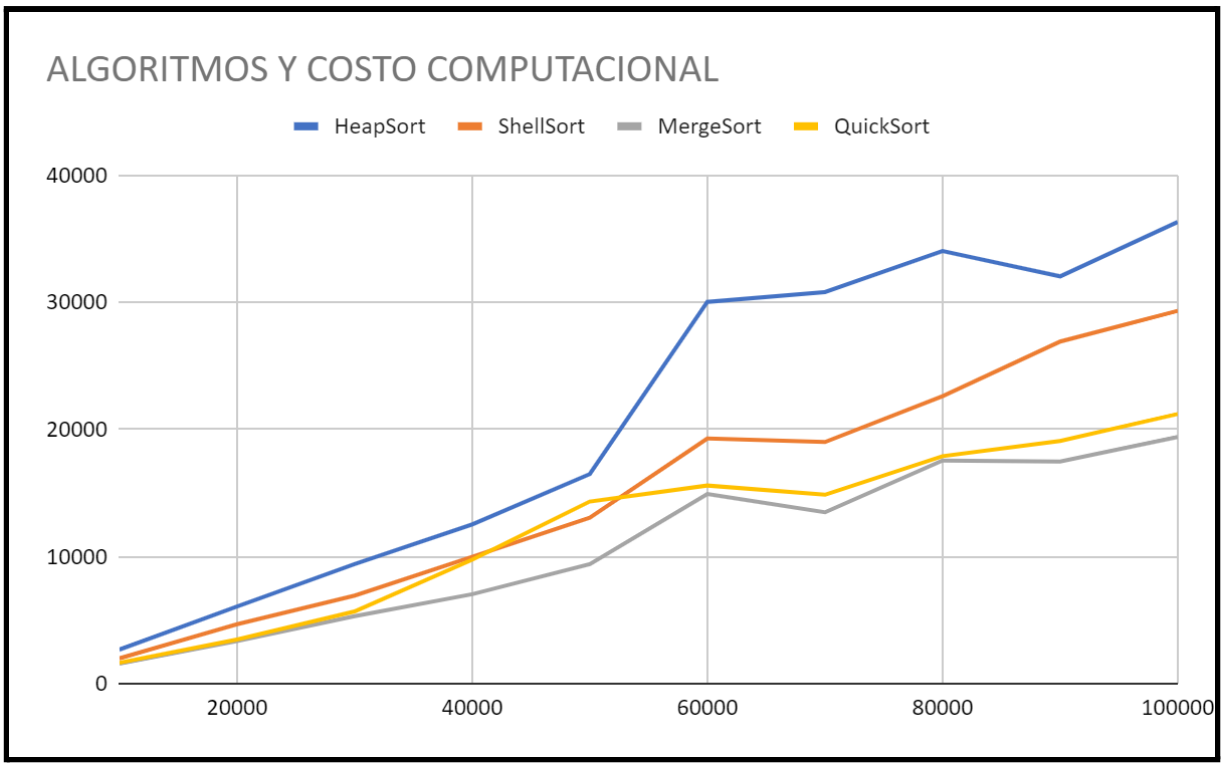

Imagen 4. Gráfica de líneas de los Algoritmos de Ordenación

B. Python

Además de probar dichos algoritmos en c++, probamos ejecutarlos en el lenguaje python, siguiendo el mismo criterio de evaluación, siendo el siguiente gráfico el resultado de los datos obtenidos:

Imagen 5. Costo Computacional de los Algoritmos de Ordenación

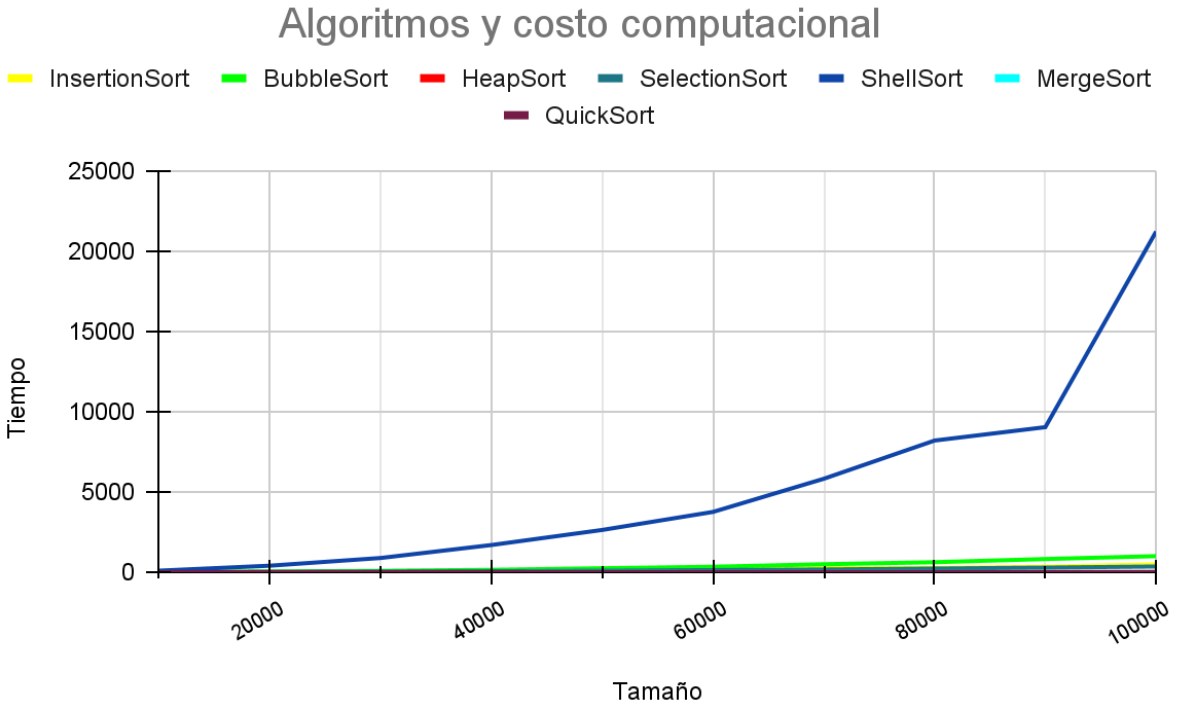

Imagen 6. Gráfica de líneas de los Algoritmos de Ordenación

Como se puede observar el algoritmo más pesado es ShellSort, ya que tarma mucho más en ejecutarse en el lenguaje python. Esto resulta bastante extraño, ya que su complejidad es mucho menor a otros algoritmos.
Para poder observar mejor los desempeños de los demás algoritmos se hizo el siguiente acercamiento:

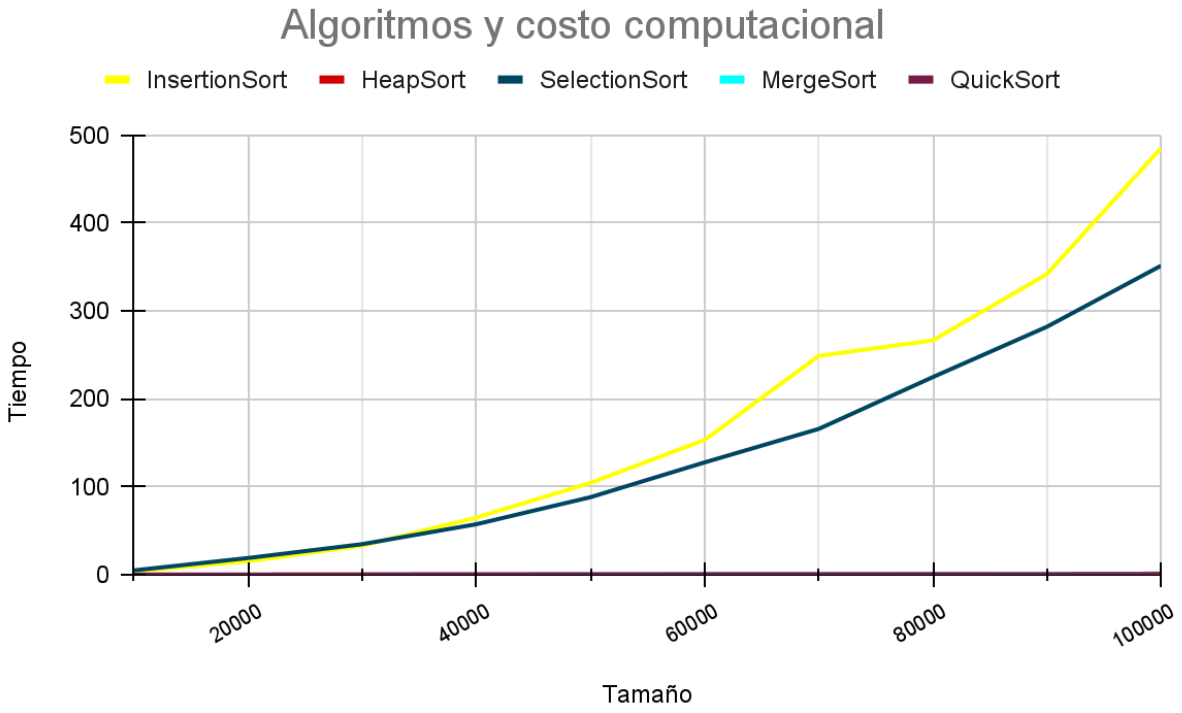

Imagen 7. Gráfica de líneas de los Algoritmos de Ordenación

Como podemos observar el segundo algoritmo con peor desempeño y el segundo mas tardado fue bubble sort, pero para mejor vista de los demas desempeños fue retirado.El tercer mas tardado fue insertion sort seguido de selection sort.

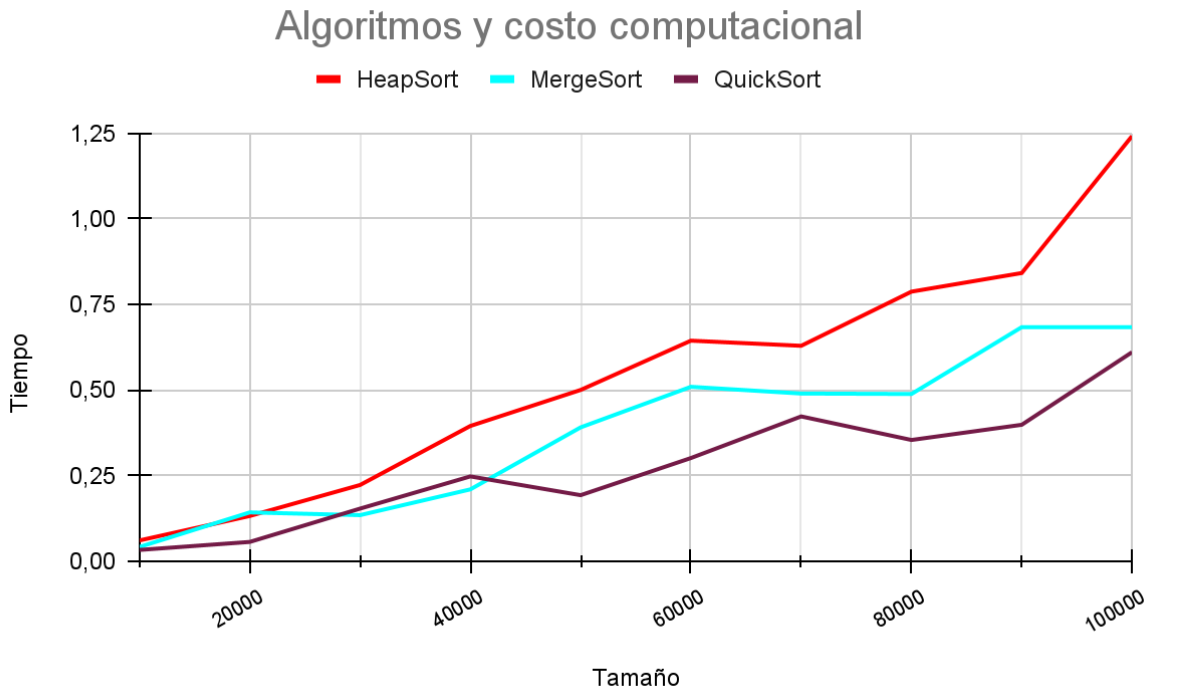

Imagen 8. Gráfica de líneas de los Algoritmos de Ordenación

Finalmente podemos decir que los más rápidos, como se esperaba, fueron QuickSort y Merge Sort.

C. Java

Hicimos el experimento en el lenguaje de Java con los mismos datos con los que se experimentó con C++ y con Python, con n datos donde 10000<=n<=100000. Se experimentó con los diferentes algoritmos de ordenamiento.
A continuación se verá una tabla que el programa en Java genera en formato .csv.
En esta tabla se verá cuando tomó en nanosegundos para ordenarse cada arreglo de diferentes tamaños de n:

n:	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
BubbleSort	148435600	4.5E+08	1.1E+09	2E+09	3E+09	4.49E+09	6.1E+09	7.98E+09	1.009E+10	1.2E+10
HeapSort	1837300	2204200	4057100	5E+06	4E+06	5511500	6471900	7503500	8537100	1.5E+07
InsertionSort	11533400	2.9E+07	7.3E+07	1E+08	2E+08	2.78E+08	4E+08	4.89E+08	657896800	7.8E+08
SelectionSort	43222700	2.2E+08	2.1E+08	4E+08	7E+08	8.57E+08	1.2E+09	1.5E+09	1.915E+09	2.4E+09
ShellSort	3139500	1817800	2785800	4E+06	5E+06	5872600	6911500	8326800	9458400	1.1E+07
MergeSort	2207300	2623600	3544000	5E+06	7E+06	5256300	7658600	9332700	11333000	1.3E+07
QuickSort	1566900	1145400	1785400	2E+06	3E+06	3700000	6949000	4957100	5518000	6433400

Imagen 11. Tabla de costos de algoritmos en lenguaje JAVA

Se puede apreciar que los algoritmos que toman más tiempo son InsertionSort, BubbleSort y SelectionSort, esto es algo lógico dado que sus complejidades son cuadráticas. A continuación se verá la tabla generada por el programa, solo que ahora se verá de forma gráfica para apreciar las diferencias entre los algoritmos de ordenamiento:

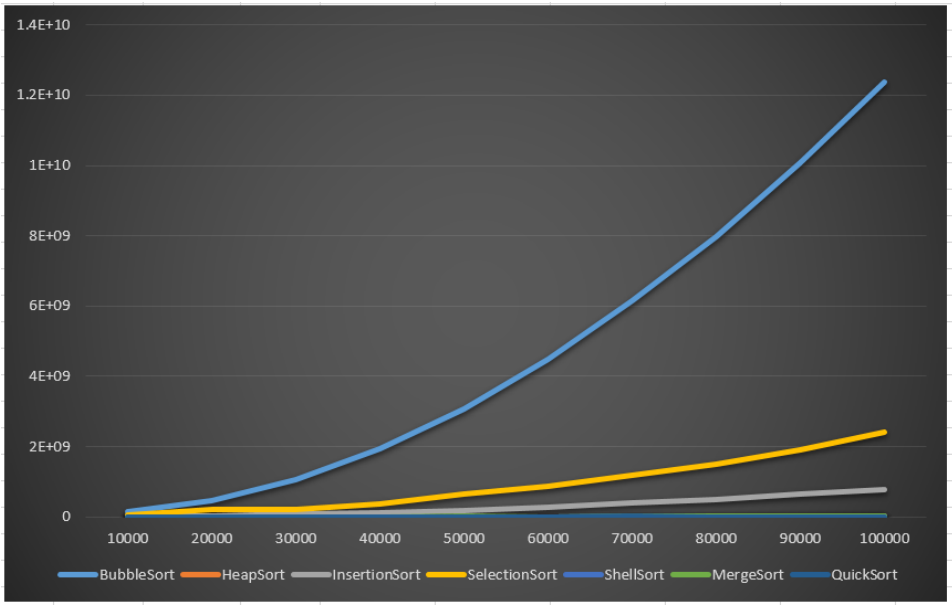

Imagen 12. Gráfica de algoritmos en lenguaje JAVA

Se pueden apreciar ciertas diferencias entre el InsertionSort, SelectionSort y BubbleSort a pesar de tener la misma complejidad. MergeSort, QuickSort, HeapSort y ShellSort tienen una gráfica muy similar, debido a que sus complejidades están en O(n*log(n)) en sus casos promedios.
Para ser más precisos con los algoritmos y que se visualicen mejor, se quitarán de la gráfica InsertionSort, BubbleSort y SelectionSort:

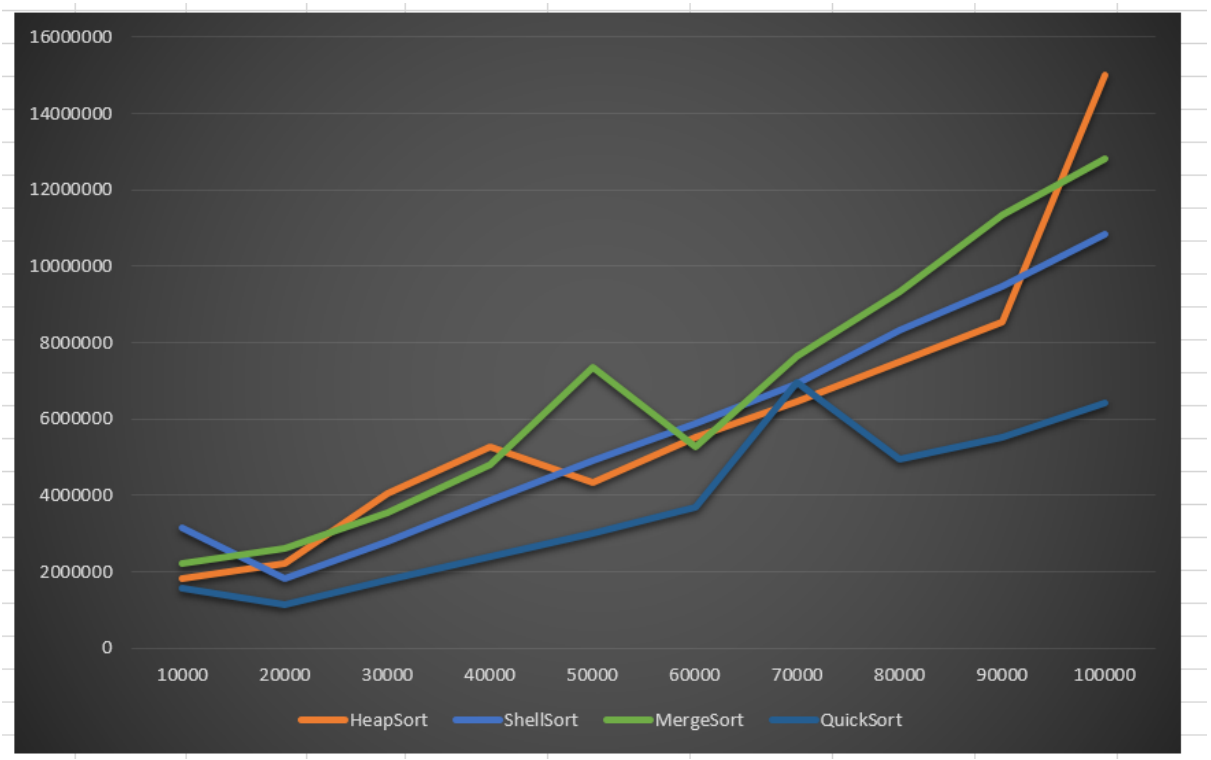

Imagen 13. Gráfica de algoritmos HeapSort, ShellSort, MergeSort y QuickSort en lenguaje JAVA

Las gráficas ahora son más parejas, sin embargo se nota una cierta ventaja para el algoritmo de QuickSort, que al parecer tiende a demorar un poco menos que los demás algoritmos.

III. Conclusiones

- A pesar de tener la misma complejidad $O(n^2)$, el algoritmo Insertion Sort tiene una clara ventaja sobre el algoritmo Bubble Sort, esta ventaja es que el primer algoritmo hace menos comparaciones que el segundo, por lo tanto, Insert Sort será más rápido que Bubble Sort en la mayoría de casos. Esto no implica que la complejidad entre ambos algoritmos cambie, sigue siendo la misma.
- El algoritmo de Selection Sort, a pesar de tener la misma complejidad $O(n^2)$ que los algoritmos de Bubble Sort e Insertion Sort, tiene una gráfica que se visualiza y se sitúa al medio de la gráfica de estos dos algoritmos, es también por el número de comparaciones y asignaciones que se tiene en este algoritmo.
- Aunque dos algoritmos tengan la misma complejidad O, no quiere decir que necesariamente al ejecutarse demorarán el mismo tiempo.

- Aunque no se aprecie en las gráficas, para valores muy pequeños de n, los algoritmos con complejidad cuadrática son mejores que los algoritmos con complejidad O(nlog(n)), debido a que se hacen menos comparaciones, evidentemente para valores muy grandes, esto cambia.
- Los algoritmos con complejidad O(nlog(n)) no son estables, ya que se puede apreciar que para algunos valores de n más grandes demoran menos que para ciertos valores de n más pequeños, por lo que las gráficas no están de forma proporcional a n.
- C++ es más rápido que JAVA y Python, debido a que en estos dos últimos se trabaja con objetos, mientras que C++ trabaja con punteros y es un lenguaje de más bajo nivel.
- No existe un mejor algoritmo de ordenamiento que siempre nos va a beneficiar, ya que todo depende del tamaño de n, en algunos casos conviene más usar InsertionSort que QuickSort o MergeSort.

IV. Discusión

Existen diversos algoritmos para ordenar una cantidad de información. Cada algoritmo tiene sus ventajas y desventajas además de que son ideales para aplicarse en un determinado contexto. Esto quiere decir que aquel algoritmo que resulta ser más óptimo en cuanto a una cantidad de datos mínima no siempre será el más óptimo si de miles o millones de datos se tratara. Saber identificar cuándo aplicar cada algoritmo supone una verdadera ventaja a la hora de programar. Además de ello, saber aplicarlo en el lenguaje adecuado hará una verdadera diferencia.

V. Bibliografía

- Algoritmo de Ordenamiento. (06 de octubre de 2021). En wikipedia. https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento
- HeapSort. GeeksForGeeks. Recuperado el 12 de octubre de 2021 de https://www.geeksforgeeks.org/heap-sort/
- InsertionSort. GeeksForGeeks. Recuperado el 12 de octubre de 2021 de https://www.geeksforgeeks.org/insertion-sort/
- SelectionSort. GeeksForGeeks. Recuperado el 12 de octubre de 2021 de https://www.geeksforgeeks.org/selection-sort/
- ShellSort. GeeksForGeeks. Recuperado el 12 de octubre de 2021 de https://www.geeksforgeeks.org/shellsort/
- MergeSort. GeeksForGeeks. Recuperado el 12 de octubre de 2021 de https://www.geeksforgeeks.org/merge-sort/
- Quicksort. GeeksForGeeks. Recuperado el 12 de octubre de 2021 de https://www.geeksforgeeks.org/quick-sort/
- BubbleSort. GeeksForGeeks. Recuperado el 12 de octubre de 2021 de https://www.geeksforgeeks.org/bubble-sort/

VI. Repositorio de trabajo

https://github.com/RodrigoJesusSantistebanPachari/ExperimentoEDA