# Programa Algoritmos de ordenamiento (agosto 25 de 2023)

Autores: Duvan Santiago Carrillo Peña, Iván Alfredo Lobatón Pachón, Juan Esteban Castillo Buitrago, Oscar Dario Hernandez Muñoz.

**Resumen – En este documento se presenta un programa desarrollado para organizar en una lista 10 millones de números que se encuentran en un archivo de extensión .txt de manera aleatoria de forma ascendente o descendente según la elección que haga el usuario. El programa utiliza diversos algoritmos de ordenamiento para lograr esta tarea de manera eficiente. Además, ofrece la capacidad de guardar la lista ordenada en un nuevo archivo .txt.**

**El programa cuenta con una interfaz de usuario amigable que permite a los usuarios seleccionar el archivo .txt que contiene los números aleatorios y definir la dirección de ordenamiento deseada. El programa incorpora tres algoritmos de ordenamiento: Merge Sort, Quick Sort y Heap Sort, para optimizar la velocidad y la eficiencia del proceso de ordenamiento. El programa cuenta con una interfaz de usuario amigable que permite a los usuarios seleccionar el archivo .txt que contiene los números aleatorios y definir la dirección de ordenamiento deseada.**

## I. INTRODUCCION

En la era de la tecnología, donde la cantidad de datos generados, analizados y recopilados en los sistemas crece de manera exponencial. El estudiante necesita las habilidades y capacidades básicas de programación recursiva, conocimiento de algoritmos y análisis del funcionamiento de los métodos de ordenamiento. La capacidad de ordenar y analizar grandes conjuntos de números es esencial para la toma de decisiones para así obtener conocimientos valiosos.

Se presenta un Programa que ofrece una funcionalidad básica de ordenar números en orden ascendente o descendente según como lo requiera organizar el usuario. Este ejercicio permitirá al estudiante introducirse a retos futuros de programación de alta complejidad con la certeza de poder dar soluciones optimas y promover conocimientos para la adquisición de nuevas competencias y mejorar sus oportunidades en el campo laboral.

En el ámbito de los algoritmos de ordenamiento, existen dos tipos principales: aquellos basados en comparaciones y los que operan en tiempo lineal. Los Primeros incluyen métodos conocidos como bubble sort, insertion sort, merge sort, heap sort y quick sort (estos tres últimos usados en el proyecto). Por otro lado, los segundos comprenden algoritmos como: counting sort, bucket sort y radix sort (Estos algoritmos incluyen procesos más complejos y no fue necesario añadirlos al proyecto).

El enfoque del proyecto radica en tres algoritmos principales: Merge sort, Heap sort y Quick sort. En ese contexto se desarrolla el programa diseñado para abordar 10 millones de números y ordenarlos de manera ascendente o descendente. Para lograr este objetivo se usan aspectos técnicos de la programación y los algoritmos aplicando el método básico de modelo-vista-controlador.

## II. OBJETIVOS

### *A. OBJETIVO PRINCIPAL.*

Desarrollar un programa de ordenamiento de números que sea capaz de procesar y organizar una lista de 10 millones de números aleatorios, Brindando opciones para ordenarlos en forma ascendente o descendente, y permitiendo al usuario guardar la lista ordenada en un archivo nuevo.

### *B. OBJETIVOS ESPECÍFICOS.*

1. La implementación de los algoritmos de ordenamiento Merge sort, Heap sort y Quick sort en el programa de manera que el usuario pueda elegir entre ellos de acuerdo con su preferencia.
2. Diseñar una interfaz de usuario intuitiva para el usuario que le permita seleccionar las opciones de ordenamiento (Ascendente o Descendente) y realizar el guardado del archivo .txt.
3. Evaluar el rendimiento de los algoritmos en términos de tiempo, identificar sus ventajas y debilidades y documentar los resultados en las conclusiones del proyecto.

## III. METODOLOGIA.

En esta sección se describirá la metodología usada para realizar el proyecto. Los pasos y enfoques orientados en los objetivos. Se explicará cómo se implementarán los algoritmos, como se diseñará la interfaz de usuario, como se realizarán las pruebas y como se evaluarán los resultados obtenidos.

- SELECCIÓN DE ALGORITMOS.

  La elección de los algoritmos se basó en dos puntos fundamentales: Eficiencia y Sencillez. Al analizar distintos algoritmos se escogieron tres algoritmos que cumplen con los requisitos, el primero de ellos: Merge sort.

  El método de ordenamiento Merge sort es un algoritmo recursivo con un numero de comparaciones entre elementos del array mínimo. Esta basado en la técnica divide y vencerás.
  Funciona de la siguiente manera:
  - Si la longitud del array es menor o igual a 1 entonces ya está ordenado.
  - El array por ordenar se divide en dos mitades de tamaño similar.
  - Cada mitad se ordena de forma recursiva aplicando el método Merge sort.
  - Las dos mitades ya ordenadas se mezclan formando una secuencia ordenada.
  - La implementación se centrará en la recursión y en cómo combinar las sub-listas ordenadas.

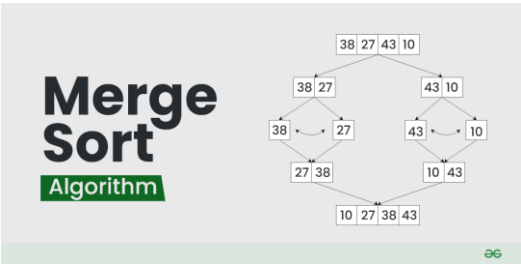


Figura 1.

  El segundo fue el algoritmo Heap sort. Este Algoritmo, es un algoritmo no recursivo, se basa en la estructura de datos llamada "montón" (heap) y Consiste en convertir la lista de elementos a ordenar en un montón y luego extraer sucesivamente el elemento máximo (en un montón máximo) o mínimo (en un montón mínimo) del montón resultante para construir la lista ordenada.
  - Se construirá un montón máximo a partir de la lista de números. Esto implica convertir la lista en una estructura de montón.
  - Luego se extraerán elementos del montón uno por uno, lo que resultará en una lista ordenada en orden ascendente.
  - La implementación se centrará en cómo construir y manipular el montón, así como en el proceso de extracción de elementos.

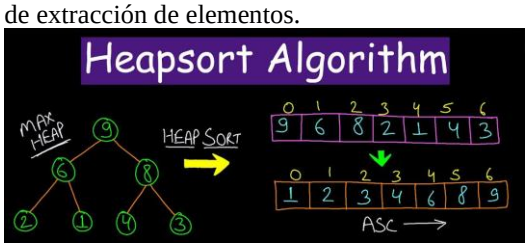


Figura 2.

  El ultimo algoritmo es el algoritmo Quick sort. Quick Sort es un método de ordenamiento que se basa en el enfoque "divide y vencerás". Consiste en seleccionar un elemento pivote de la lista, reorganizar los elementos de manera que los menores que el pivote estén a su izquierda y los mayores a su derecha, y luego aplicar recursión en las sublistas resultantes para ordenarlas.

  - Se seleccionará un elemento pivote de la lista. Los elementos se reorganizarán de manera que los menores que el pivote estén a su izquierda y los mayores a su derecha.
  - Se aplicará recursión a las sublistas izquierda y derecha del pivote para ordenarlas.
  - La implementación se centrará en cómo seleccionar el pivote y en cómo reorganizar los elementos en torno a él.

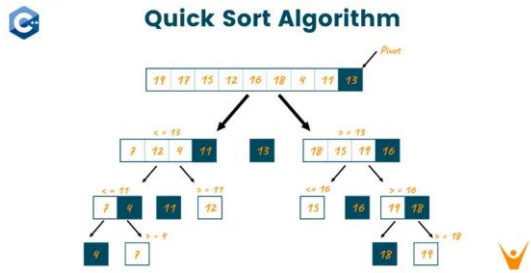


Figura 3.

La implementación de cada algoritmo requerirá manejo de arreglos, recursión y manipulación de estructuras de datos específicas (como montones en Heap Sort). Además, se deben considerar casos base, condiciones de terminación y optimización de rendimiento para cada algoritmo.

La selección de estos tres algoritmos proporciona un enfoque variado para la tarea de ordenamiento, considerando la eficiencia y la adaptabilidad a diferentes escenarios. La implementación exitosa de estos algoritmos permitirá a los usuarios experimentar con distintos métodos de ordenamiento y evaluar su rendimiento en función de diferentes conjuntos de datos.

## IV. ORDEN DE COMPLEJIDAD DE UN ALGORTIMO.

Se refiere a cuánto tiempo (o recursos) requiere un algoritmo para resolver un problema en función del tamaño de entrada. Se expresa en términos de la notación "big O" (O), que describe el límite superior asintótico del crecimiento del tiempo de ejecución en relación con el tamaño de entrada.

Es importante determinar la complejidad temporal de un algoritmo para comprender cómo escala su rendimiento a medida que se trabaja con tamaños de entrada más grandes. Una complejidad menor es deseable para lograr tiempos de ejecución más rápidos, especialmente en conjuntos de datos grandes.

| Orden | Nombre | Comentario |
|---|---|---|
| O(1) | Constante | Se aplica a los algoritmos cuya ejecución se realice en un tiempo constante. |
| O(log n) | Logarítmico | Están considerados los que implican bucles con menos iteraciones, como por ejemplo una búsqueda binaria. |
| O(n) | lineal | En este tipo de notación el tiempo crece linealmente con respecto a las iteraciones |
| O(n·log(n)) | n por logaritmo de n | La mayor parte de los algoritmos tienen un orden superior, combina el logarítmico con el lineal. |
| $O(n^c)$, con c>1 | poli nómico | Aquí están muchos de los algoritmos más comunes. Cuando c es 2 se le llama **cuadrático**, cuando es 3 se le llama **cúbico**, y en general es poli nómico. |
| $O(c^n)$, con c>1 | exponencial | Aunque pudiera no parecerlo, es mucho peor que el anterior. Crece muchísimo más rápidamente. |

[i](Mexico, 01/08/2017)

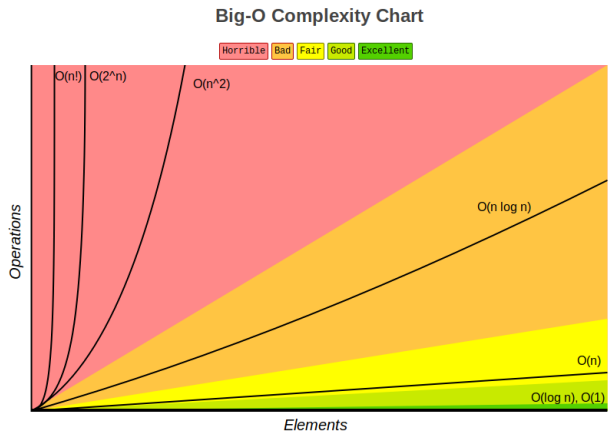


Figura 4. Notation Big O

## V. RESULTADOS Y DISCUSIÓN.

Las pruebas iniciales: El primer fallo que tuvo el programa fue que el algoritmo Merge Sort cuando se elegia el orden descendente mostró caracteres chinos.

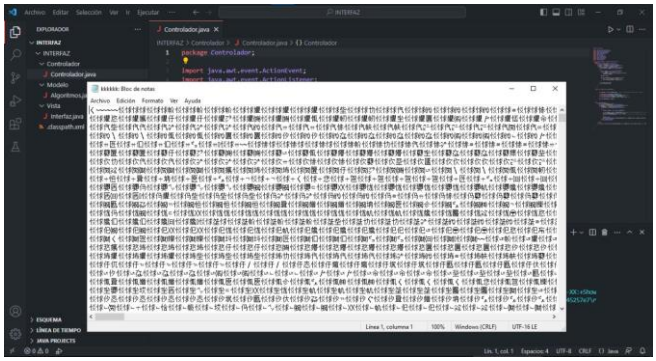

Figura 5.

Aunque después de varias pruebas el programa organizó los números de acuerdo con la elección del usuario, tuvo problemas de optimización, ya que se demoraba un prolongado tiempo en ordenar los números con cualquiera de los algoritmos.

## IV. CONCLUSIÓN

En el transcurso de este proyecto, se ha logrado el diseño, implementación y evaluación de un programa de ordenamiento de números que utiliza los algoritmos Merge Sort, Heap Sort y Quick Sort. Estos algoritmos, cada uno con sus características y enfoques particulares, han demostrado su eficacia en la tarea de ordenar conjuntos de datos en diferentes escenarios.

A través de las pruebas realizadas, se observó que el algoritmo Merge Sort es especialmente adecuado para conjuntos de datos más grandes debido a su eficiencia en términos de tiempo. Heap Sort, por su parte, demostró ser robusto en diferentes situaciones, gracias a su estructura de montón que le permite un acceso rápido a los elementos máximos o mínimos. Quick Sort mostró un excelente rendimiento en la práctica, a menudo superando a los otros algoritmos en conjuntos de datos de tamaño moderado.

Los resultados obtenidos no solo cumplieron con los objetivos establecidos, sino que también proporcionaron una comprensión más profunda de cómo los algoritmos de ordenamiento pueden influir en el rendimiento y la eficiencia. Al evaluar la relación entre los tamaños de datos y los tiempos de ejecución, se pudo apreciar cómo diferentes algoritmos se adaptan a diferentes situaciones.

Aunque este proyecto ha logrado implementar y evaluar tres algoritmos de ordenamiento, siempre existe margen para la mejora y la expansión. Se pueden considerar futuras investigaciones sobre optimización de algoritmos, implementación de nuevos algoritmos de ordenamiento o la adaptación de los algoritmos existentes para resolver problemas específicos.
En última instancia, este proyecto ha enriquecido nuestra comprensión de los algoritmos de ordenamiento y su papel fundamental en la eficiencia del procesamiento de datos. La experiencia adquirida a lo largo del proyecto servirá como base para futuros desarrollos y exploraciones en el campo de la informática y la optimización algorítmica.

## VI. BIBLIOGRAFIAS


- http://ri.uaemex.mx/bitstream/handle/20.500.11799/69985/secme-3691_1.pdf?sequence=1
- https://www.redalyc.org/pdf/849/84920977007.pdf
- https://rua.ua.es/dspace/handle/10045/127649
- https://rua.ua.es/dspace/bitstream/10045/127649/1/JENUI_2005_026.pdf
- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). "Introduction to Algorithms." MIT Press.
- Sedgewick, R., & Wayne, K. (2011). "Algorithms." Addison-Wesley Professional.
- Heap Sort (2021). En Wikipedia. https://en.wikipedia.org/wiki/Heapsort
- Merge Sort (2021). En Wikipedia. https://en.wikipedia.org/wiki/Merge_sort
- Hoare, C. A. R. (1962). "Quicksort." The Computer Journal, 5(1), 10-15. doi:10.1093/comjnl/5.1.10