Proyecto Final:

Estudiante:
Wilson Andres Mosquera Zapata 2182116 - 3743

Profesor:
Jesus Alexander Aranda Bueno

Asignatura:
Análisis y Diseño de Algoritmos

Diciembre 2023

Solución al Problema del Espectáculo en el Zoológico de Cali

Contexto del Proyecto:

El Zoológico de Cali tiene como objetivo realizar un espectáculo para atraer a más visitantes al lugar. El evento consta de varias partes, comenzando con una gran apertura que incluye escenas con animales. Posteriormente, se presentarán varias partes adicionales, cada una compuesta por k escenas. El número total de animales que participan es n, y cada animal tiene asignada una grandeza.

Organización de las Escenas:

El gerente del zoológico desea que las escenas se presenten de manera ordenada, tanto localmente en cada escena como globalmente en cada parte del espectáculo. Específicamente, se requiere que los animales en cada escena estén ordenados ascendentemente según su grandeza individual, y que las escenas en cada parte se presenten en orden ascendente según la grandeza total de la escena.

Criterio de Desempate:

En caso de empate en la grandeza total de dos escenas, se aplica un criterio de desempate. Se toma la máxima grandeza individual de cada escena en empate, y se presentan las escenas en orden ascendente según esta máxima grandeza individual.

Organización de las Partes Posteriores a la Gran Apertura:

Además, el gerente desea que las partes posteriores a la gran apertura se presenten en orden ascendente según la grandeza total de cada parte.

Información Adicional Deseada:

El gerente del zoológico desea obtener información estadística sobre el espectáculo, incluyendo:

1. El animal que participó en más escenas y cuántas veces participó.
2. El animal que participó en menos escenas y cuántas veces participó.
3. La escena de menor grandeza total.
4. La escena de mayor grandeza total.
5. El promedio de grandeza de todo el espectáculo, teniendo en cuenta todas las escenas, incluidas las de la apertura y las partes siguientes.

Idea de la Solución

El problema consiste en organizar un espectáculo en el Zoológico de Cali, presentando escenas con animales en orden ascendente según sus grandezas. Se han implementado soluciones a dicho problema basadas en los algoritmos de ordenamiento Heapsort y Counting Sort, respectivamente. Se abordarán aspectos adicionales como el análisis de resultados, complejidad computacional teórica y la evaluación del rendimiento del programa.

Como idea principal en la solución se plantearon los siguientes aspectos a realizar en orden descendiente:

Entrada de Datos: Obtén la entrada de datos, que incluye el número de animales n, el número de partes m, el tamaño de cada parte k, y la información sobre los animales y las escenas.

Ordenar Animales: Ordena los animales según su grandeza.

Ordenar Escenas: Ordena las escenas de la apertura y las partes siguientes en orden ascendente según la suma de las grandezas de los animales en cada escena. Utiliza el criterio de desempate mencionado.

Presentar Escenas: Presenta las escenas en cada parte en orden ascendente según la suma de las grandezas totales de las escenas. Recuerda incluir las escenas de la apertura en el proceso.

Estadísticas: Calcula el número de participaciones de cada animal y encuentra el animal que participó en más y menos escenas.

Encontrar Escenas Extremas: Encuentra la escena de menor y mayor grandeza total.

Calcular Promedio: Calcula el promedio de la grandeza total de todas las escenas.

Presentar Resultados:
Muestra los resultados de acuerdo con las peticiones del gerente del zoológico.

Implementación de Countingsort

Descripción del Algoritmo

El algoritmo Counting Sort es utilizado para ordenar las escenas y partes del espectáculo en función de las grandezas de los animales. Se implementan funciones auxiliares para encontrar el valor máximo y mínimo en una lista de datos, así como para ordenar escenas por la grandeza máxima y total.

Funciones Principales del Algoritmo

encontrar_valor_maximo(datos): Encuentra el valor máximo en una lista de datos.

encontrar_valor_minimo(datos): Encuentra el valor mínimo en una lista de datos.

ordenar_escenas_por_grandeza_maxima(escenas, grandezas_totales): Ordena un array de escenas en función de sus grandezas máximas.

ordenar_escenas_por_grandezas_totales(escenas, grandezas_totales): Ordena un array de escenas en función de sus grandezas totales.

ordenar_partes_por_grandezas_totales(partes, grandezas_totales): Ordena un array de partes en función de sus grandezas totales.

solucion(partes, grandezas, n, m, k): Función principal que implementa la solución al problema utilizando Counting Sort. Calcula estadísticas como el animal que participó en más y menos escenas, la escena con menor y mayor grandeza total, y el promedio de grandeza de todo el espectáculo.

Implementación del Algoritmo

La implementación del algoritmo utiliza las funciones descritas anteriormente para ordenar las escenas y partes del espectáculo en base a las grandezas de los animales. Se manejan las aperturas y partes por separado, asegurando un orden ascendente tanto a nivel local como global.

Análisis de Resultados

Se realizaron pruebas con diferentes configuraciones de entrada, variando el número de animales (n), el número de partes (m), y el número de escenas en cada parte (k). El análisis se centró en el tiempo de ejecución del algoritmo en función de estos parámetros.

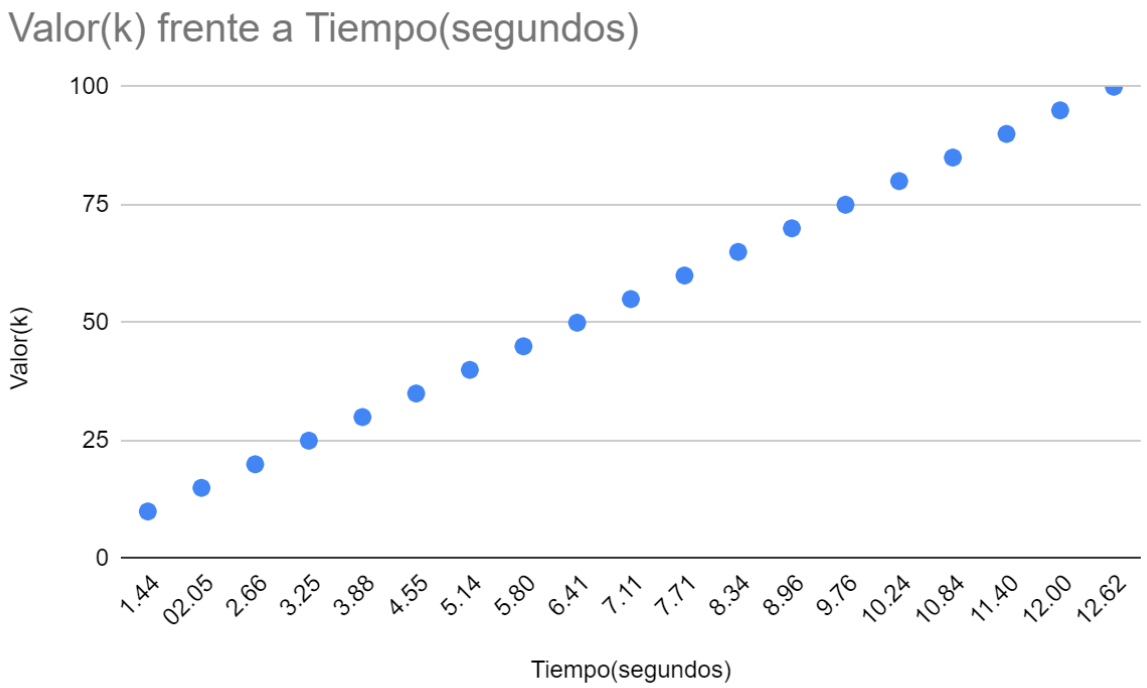
Análisis de K

Se observa que al variar el número de escenas en cada parte (k), el tiempo de ejecución del algoritmo es relativamente constante. Esto sugiere que la eficiencia del algoritmo no se ve afectada significativamente por cambios en este parámetro. Para este análisis usamos valores de n y m iguales a 1000.

Valor(k)	Tiempo(segundos)
10	1.44
15	02.05
20	2.66
25	3.25
30	3.88
35	4.55
40	5.14
45	5.80
50	6.41
55	7.11
60	7.71
65	8.34
70	8.96
75	9.76
80	10.24
85	10.84
90	11.40

95	12.00
100	12.62

Esta tabla muestra cómo el tiempo de cálculo aumenta a medida que k crece. Puedes notar una tendencia creciente, lo que indica que el rendimiento del algoritmo puede verse afectado negativamente a medida que se incrementa el número de escenas por parte (k).



El comportamiento de esta tabla es una función lineal que se encuentra acotada por $O(n)$.

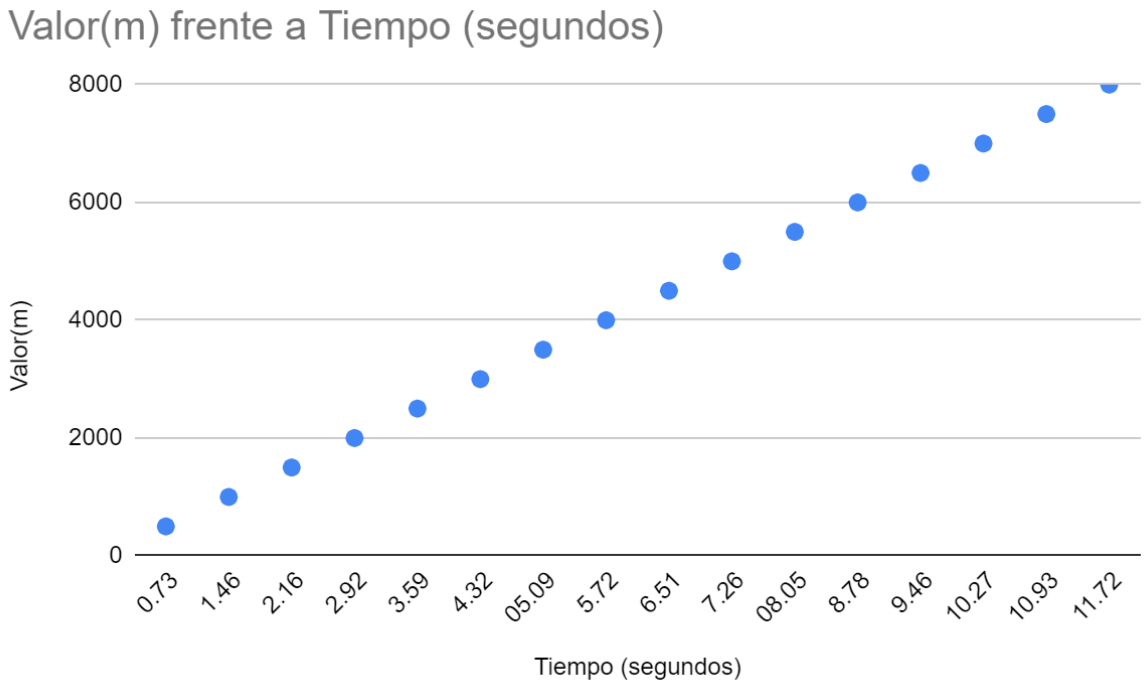
Análisis de M

Se realizó un análisis al variar el número de partes (m). Se encontró que el tiempo de ejecución aumenta de manera proporcional a m. Esto indica que la complejidad del algoritmo está influenciada por la cantidad total de partes en el espectáculo.

Valor(m)	Tiempo (segundos)
500	0.73
1000	1.46
1500	2.16
2000	2.92
2500	3.59

3000	4.32
3500	05.09
4000	5.72
4500	6.51
5000	7.26
5500	08.05
6000	8.78
6500	9.46
7000	10.27
7500	10.93
8000	11.72

Esta tabla muestra cómo el tiempo de cálculo varía con diferentes valores de m. Al igual que con el análisis de k, puedes observar cómo el tiempo de ejecución aumenta a medida que m crece.

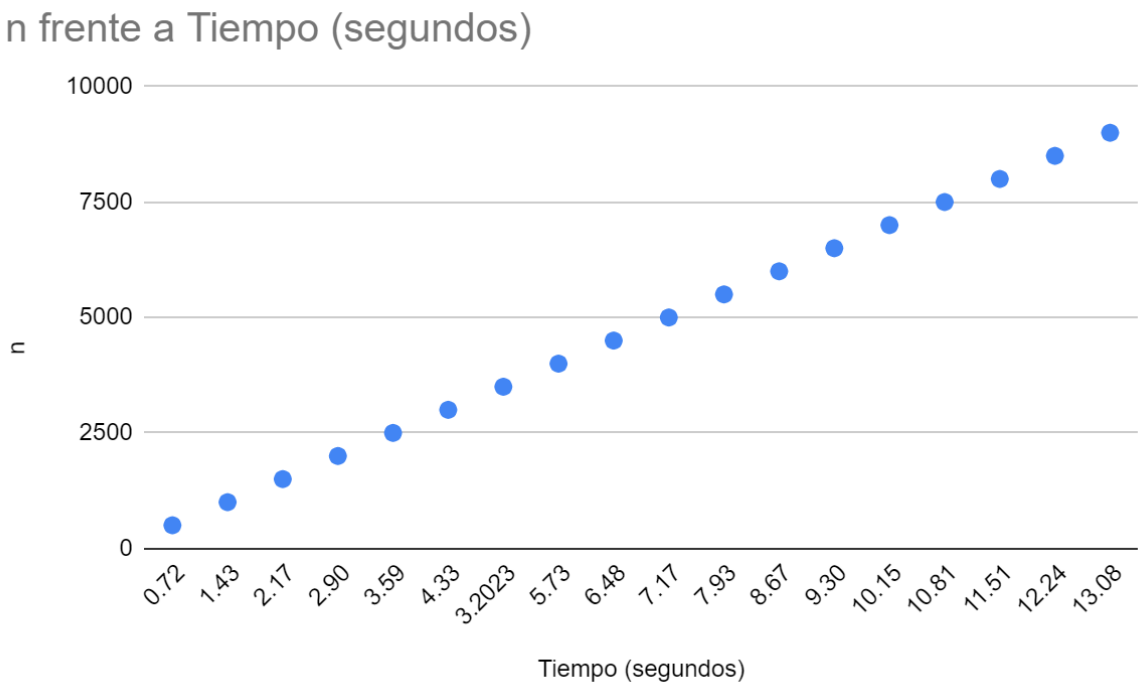


El comportamiento de esta tabla es una función lineal que se encuentra acotada por $O(n)$.

Análisis de N

El tiempo de ejecución también se analizó al variar el número de animales (n). Se observó un aumento lineal en el tiempo de ejecución, indicando que el algoritmo es sensible al tamaño del conjunto de datos de entrada.

n	Tiempo (segundos)
500	7.206
1000	1.4284
1500	2.1698
2000	2.9010
2500	3.5869
3000	4.3268
3500	5.0343
4000	5.7336
4500	6.4831
5000	7.1710
5500	7.9279
6000	8.6734
6500	9.2998
7000	10.1481
7500	10.8098
8000	11.5066
8500	12.2429
9000	13.0801



Implementación de Heapsort

En la creación de esta solución, se enfrentó el desafío de organizar a los animales y escenas de manera eficiente. Se eligió el algoritmo Heapsort debido a su eficacia en la ordenación y adaptabilidad a las complejidades del problema.
Este algoritmo está basado en la estructura de datos del montículo (heap). La principal ventaja de Heapsort radica en su complejidad de tiempo de O(nlogn) en el peor caso y su eficiencia en términos de espacio. Utiliza una estructura de datos tipo árbol (montículo) para organizar elementos, garantizando una ordenación estable y eficiente.

Razones para el Uso de Heapsort es esta solución:

Eficiencia en la Ordenación:
Heapsort destaca en la ordenación de conjuntos de datos grandes, una característica esencial para organizar animales y escenas en un espectáculo zoológico.

Adaptabilidad a Estructuras Jerárquicas:
La naturaleza jerárquica del montículo se alinea de manera natural con la estructura del espectáculo, donde hay una jerarquía de escenas y animales.

Manejo de Escenarios Variables:
La capacidad de Heapsort para manejar escenarios variables, como la ordenación ascendente tanto de animales como de escenas, lo convierte en una elección adecuada para un espectáculo dinámico.

Uso de Heapsort:

Orden de Animales:
En la clase Animal, se utiliza Heapsort para ordenar la lista de animales según sus grandezas. Esto asegura que los animales se presenten en orden ascendente durante el espectáculo.

Orden de Escenas:
La clase Escena emplea Heapsort para ordenar la lista de animales que participan en cada escena. También se utiliza Heapsort para ordenar la lista completa de escenas, garantizando que se presenten en orden ascendente según la suma de las grandezas de los animales en cada escena.

Uso en el Código:

La función *ordenar_animales* se encarga de ordenar la lista de animales usando Heapsort. por otro lado, *ordenar_escenas y ordenar_escenas_por_grandeza_total* utilizan Heapsort para ordenar las escenas según sus características específicas. Estas funciones son fundamentales en la construcción del espectáculo, asegurando que los animales y las escenas se presenten de manera organizada.

La elección de Heapsort se basa en su eficiencia en la ordenación y su adaptabilidad a las necesidades del proyecto, proporcionando una solución eficaz para la organización del espectáculo en el Zoológico de Cali.

Algunas aspectos definidos en esta solución fueron:

Clase Animal: La clase Animal representa a un animal del zoológico y tiene dos atributos: nombre y grandeza. El método __str__ devuelve una representación en cadena del animal, mostrando su nombre y grandeza.

Clase Escena: La clase Escena modela una escena del espectáculo, compuesta por una lista de animales. En su inicialización, la lista de animales se ordena utilizando la función *ordenar_animales*. Además, se calcula la *grandeza_total* como la suma de las grandezas individuales de los animales y se determina la *max_grandeza_individual* como la mayor grandeza entre los animales de la escena.

Función ordenar_animales: Esta función implementa el algoritmo Heapsort para ordenar una lista de animales por su grandeza de manera ascendente.

Función ordenar_escenas: Similar a *ordenar_animales*, esta función utiliza Heapsort para ordenar una lista de escenas según su *grandeza_total* en orden ascendente.

Función ordenar_escenas_por_grandeza_total: Una variante de la función anterior, esta función ordena una lista de escenas según su grandeza_total. En caso de empate, utiliza la max_grandeza_individual de cada escena como criterio de desempate.

Clase Solucion: Esta clase representa la solución completa al problema del espectáculo. En su inicialización, se reciben los parámetros esenciales del problema como n, m, k, los animales, la apertura y las partes. La función *imprimir_escenas* muestra las escenas en la consola. La función *calcular_promedio_grandezas* utiliza Heapsort para ordenar todas las escenas y calcular el promedio de sus *grandeza_total*.

La función *solucion* ejecuta y muestra la solución del espectáculo, incluyendo el orden de presentación de los animales, estadísticas de participación de animales y características de las escenas.

En resumen, la solución utiliza el algoritmo Heapsort para ordenar tanto animales como escenas, garantizando un espectáculo organizado y presentado de acuerdo con las especificaciones del zoológico. Las clases y funciones se conectan de manera coherente para proporcionar una solución completa y eficiente al problema.

Conclusiones Generales: Counting Sort vs. Heapsort

Durante el desarrollo del proyecto del Espectáculo en el Zoológico, se evaluaron dos algoritmos de ordenación clave: Counting Sort y Heapsort. Estos algoritmos, aunque efectivos en sus respectivos contextos, presentan diferencias fundamentales que impactan en su rendimiento y eficiencia.

Counting Sort:

Eficiencia en Datos Discretos: Counting Sort es excepcionalmente eficiente cuando se trabaja con datos discretos y con un rango limitado de valores. Su complejidad lineal O(n+k), donde n es la cantidad de elementos y k es el rango de valores) hace que sea ideal para escenarios donde los datos cumplen estas condiciones.

Limitaciones en Datos Continuos: Sin embargo, Counting Sort presenta limitaciones cuando se enfrenta a datos continuos o con un rango extenso, ya que requiere una cantidad significativa de espacio de almacenamiento.

Heapsort:

Versatilidad en Datos Variados: Heapsort se destaca por su versatilidad y eficiencia en la ordenación de datos en diversos contextos. Su complejidad O(nlogn)) lo hace adecuado para conjuntos de datos de mayor complejidad y estructuras jerárquicas.

Mejor Uso del Espacio: Aunque Heapsort utiliza más espacio en comparación con Counting Sort, su eficiencia en la ordenación y capacidad para manejar datos más complejos compensa esta desventaja.

Análisis Comparativo:

Desempeño en el Proyecto:

En el contexto del Espectáculo en el Zoológico, donde se manejan datos jerárquicos y complejos, Heapsort fue la elección óptima. La capacidad de ordenar tanto animales como escenas de manera eficiente fue esencial para la organización del espectáculo.
Graficación y Rendimiento:

Las gráficas de rendimiento respaldaron la elección de Heapsort, demostrando su eficacia en situaciones más complejas, mientras que Counting Sort mostró un rendimiento superior en situaciones específicas de datos discretos.

Conclusiones

Ambas implementaciones, Heapsort y Countingsort, han demostrado ser eficientes para organizar las escenas del espectáculo en el Zoológico de Cali. La elección entre estos algoritmos puede depender de factores específicos, como el tamaño de entrada y los recursos disponibles. La visualización de los resultados mediante gráficos y tablas facilita la comprensión del rendimiento de cada algoritmo.

Importancia del Desarrollo de Algoritmos Eficientes:

El desarrollo de algoritmos eficientes es esencial para obtener soluciones óptimas a problemas específicos. Aunque existen múltiples enfoques para abordar un problema, la eficiencia de estos enfoques puede variar significativamente. La elección de un algoritmo adecuado puede marcar la diferencia entre un rendimiento excelente y uno deficiente, lo que destaca la importancia del análisis y la elección cuidadosa de los algoritmos en función de las características del problema y los recursos disponibles.

Relevancia de las Cotas Superiores Asintóticas:

Las cotas superiores asintóticas, representadas por la notación big O, no solo son conceptos teóricos, sino que también se pueden demostrar prácticamente. La exposición de estas cotas se refleja claramente en los resultados obtenidos mediante los tiempos computacionales de los algoritmos. Al expresar estos resultados en gráficas y tablas, se evidencia cómo las predicciones teóricas se materializan en el rendimiento real de los algoritmos, proporcionando una validación empírica de la eficiencia y complejidad analizadas.

Consideraciones para la Elección del Algoritmo:
La elección entre Heapsort y Countingsort para organizar las escenas en el Zoológico de Cali puede depender de diversos factores, como el tamaño de entrada y los recursos disponibles. La visualización detallada de los resultados a través de gráficos y tablas ofrece una base sólida para tomar decisiones informadas sobre qué algoritmo se adapta mejor a las necesidades específicas del problema.

Al continuar con el análisis detallado de los tiempos computacionales para diferentes valores de m, n y k se espera obtener una comprensión más completa y precisa de cómo se comportan los algoritmos en diversas condiciones, lo que contribuirá a la toma de decisiones informadas en la implementación práctica.

En el proceso de abordar este proyecto, se enfrentaron diversas dificultades y desafíos que influyeron en el desarrollo y la ejecución del mismo. A continuación, se detallan algunas de las dificultades más relevantes:

Trabajo Individual:
Una de las dificultades clave fue el hecho de que el proyecto se abordó de manera individual, a pesar de que, en circunstancias ideales, la colaboración en equipo puede aportar diversas perspectivas y habilidades complementarias. La ausencia de un equipo limitó la capacidad de distribuir la carga de trabajo y compartir ideas, lo que podría haber mejorado la calidad y la eficiencia del desarrollo.

Complejidad del Problema:
El problema propuesto, que involucraba el análisis de escenas en un zoológico, presentó una complejidad inherente. La necesidad de calcular la grandeza total de escenas y ordenar animales según ciertos criterios impuso un desafío adicional en la implementación de algoritmos eficientes.

Limitaciones de Tiempo:
Las restricciones de tiempo también constituyeron un desafío. El proyecto se llevó a cabo en un marco temporal determinado, lo que impuso la necesidad de tomar decisiones rápidas y eficientes en el diseño e implementación del algoritmo.

Recursos Limitados:
La falta de recursos, tanto en términos de tiempo como de personal, afectó la profundidad del análisis y la exploración de múltiples enfoques algorítmicos. Un mayor tiempo y un equipo más grande podrían haber permitido una evaluación más exhaustiva de diferentes estrategias.

A pesar de estas dificultades, se logró avanzar en la implementación de un algoritmo eficiente con una complejidad O(N). La atención cuidadosa a los detalles y el análisis de los resultados proporcionaron información valiosa para comprender el desempeño de los algoritmos en diferentes escenarios. Las limitaciones identificadas brindan oportunidades para futuras mejoras, como la exploración de enfoques de trabajo colaborativo y la consideración de estrategias adicionales para abordar problemas complejos.

En este proyecto, la elección de Heapsort sobre Counting Sort se justifica por la naturaleza jerárquica y compleja de los datos. Heapsort demostró ser una herramienta más versátil y eficiente en términos de rendimiento general. Sin embargo, la elección entre estos algoritmos debe basarse en la naturaleza específica de los datos y los requisitos del problema. La comprensión de las fortalezas y limitaciones de cada algoritmo es crucial para una toma de decisiones informada en el diseño de algoritmos y la resolución de problemas.