1. **Reseña Histórica**

Inicio de los algoritmos de ordenamiento:

- En los primeros días de la computación, cuando los sistemas solo podían manejar tareas simples, se usaban algoritmos como Bubble Sort y Selection Sort.

Aparición de algoritmos más eficientes:

- Con la mejora del poder de procesamiento, surgieron algoritmos más potentes como Merge Sort (años 40) y QuickSort (años 60).
- Estos algoritmos ofrecieron una eficiencia cercana a O(n log n), lo cual representó un gran avance en el procesamiento de grandes volúmenes de información.

Adaptación a datos del mundo real:

- En la práctica, los datos muchas veces no están completamente desordenados.
- Para aprovechar esto, se creó Timsort en 2002, un algoritmo diseñado para trabajar con datos parcialmente ordenados, como los que encontramos en redes sociales o sistemas personalizados.

Características y ventajas de Timsort:

- Es un algoritmo híbrido, que combina:
    - La velocidad de Merge Sort.
    - La eficiencia en casos simples de Insertion Sort.

Aplicaciones actuales:

- Timsort se utiliza hoy en día en lenguajes como Python y Java.
- También está presente en plataformas como TikTok, donde se necesita procesar grandes cantidades de datos en tiempo real para ofrecer contenido personalizado.

2. Teoría Brindada por el estudiante

Con el objetivo de analizar el desempeño del algoritmo Timsort, se estableció una metodología basada en la recolección sistemática de datos referentes al tiempo de ejecución sobre conjuntos de datos de distintos tamaños. Esta estrategia permitirá observar el comportamiento del algoritmo bajo diversas condiciones de carga y evaluar su eficiencia en

términos cuantitativos. Para facilitar el tratamiento de la información, los tiempos obtenidos se agruparán en intervalos previamente definidos, lo cual posibilitará su análisis estadístico a través de medidas como la media, la mediana, la moda y la desviación estándar.

El enfoque del estudio se centrará exclusivamente en la relación entre el tamaño del conjunto de entrada y el rendimiento temporal del algoritmo. A partir de los datos obtenidos, se buscará identificar tendencias centrales que indiquen comportamientos típicos del algoritmo, así como niveles de dispersión que evidencien variabilidad en los resultados. Este análisis permitirá no solo describir el comportamiento de Timsort, sino también realizar inferencias que podrían ser útiles para optimizar su implementación.

En este contexto, la estadística se convierte en una herramienta clave para sustentar con evidencia empírica las conclusiones del estudio, asegurando un enfoque riguroso y fundamentado en datos reales.

3. Hipótesis

Tipo Investigador

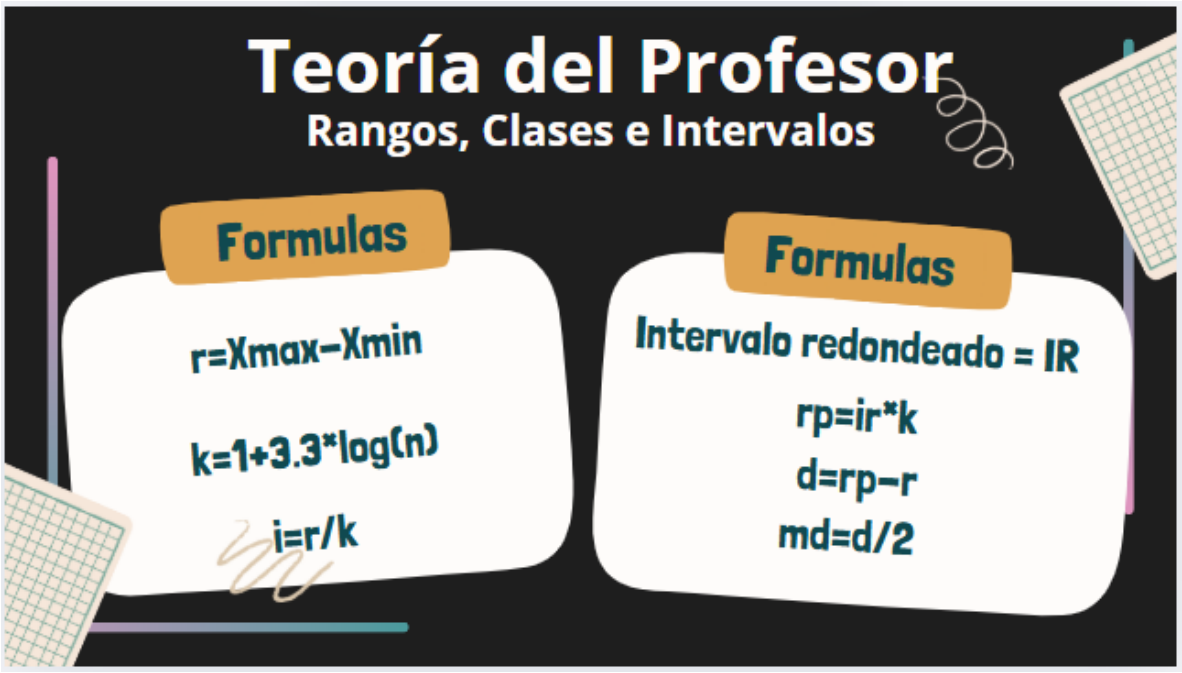
La hipótesis del investigador plantea que el algoritmo de ordenamiento Timsort mantiene su eficiencia computacional al procesar grandes volúmenes de datos típicos en entornos TIC, incluso cuando estos presentan distribuciones aleatorias o parcialmente ordenadas.

Tipo Nula

La hipótesis nula plantea que el algoritmo de ordenamiento Timsort experimenta una degradación significativa de eficiencia a medida que el tamaño de los datos aumenta, lo que contradice su supuesta complejidad óptima en escenarios reales de grandes volúmenes de información.

4. Teoría del Profesor

Rango, Clases e Intervalos: Definición de intervalos sobre los tiempos de ejecución.



En primer lugar, utilizando la siguiente formula, calculamos el Rango de los datos:

$$r=Xmax-Xmin$$

Esta operación nos permite medir la diferencia entre el mayor y el menor valor observado.

Después de esto, determinamos el número de clases utilizando la fórmula de Sturges:

$$k=1+3.3*\log(n)$$

Donde "n" representa el número total de datos recolectados. Esta formula nos proporciona una adecuada cantidad de categorías para así agrupar los datos de manera organizada.

Como tercer paso, calculamos el tamaño del intervalo mediante la formula mostrada en pantalla:

$$i=r/k$$

Cuyo valor nos indica la amplitud de cada intervalo o clase.

Por consiguiente, con el tamaño del intervalo y el número de clases, aplicando la siguiente operación, obtenemos el rango propuesto

$$rp=ir*k$$

El rango propuesto busca ajustar correctamente los límites de los datos para que no queden valores sin clasificar.

Seguidamente, calculamos la diferencia de rangos con esta fórmula:

$$d=rp-r$$

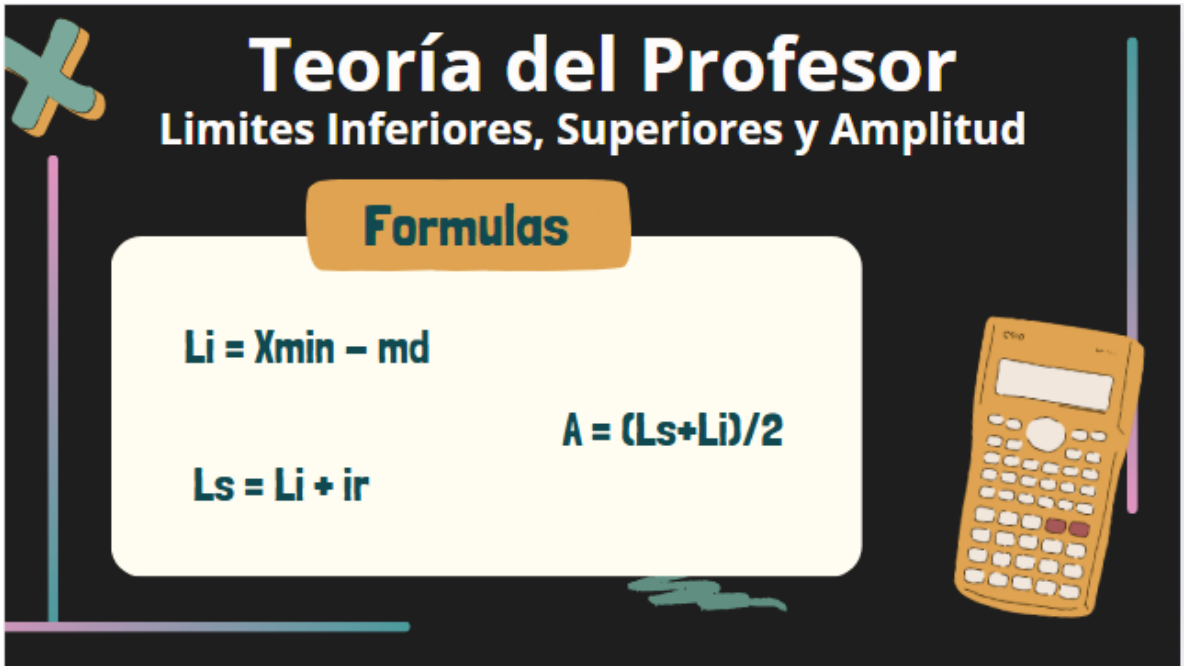
Donde esta diferencia, nos indica el pequeño desfase que puede surgir tras los ajustes realizados.

Luego de esto, calculamos la diferencia media utilizando la siguiente expresión:

$$md=d/2$$

La finalidad de esta operación es distribuir el ajuste de manera equilibrada en los extremos de la distribución.

Límites Inferiores, Superiores y Amplitud: Determinación precisa de los límites para cada clase.

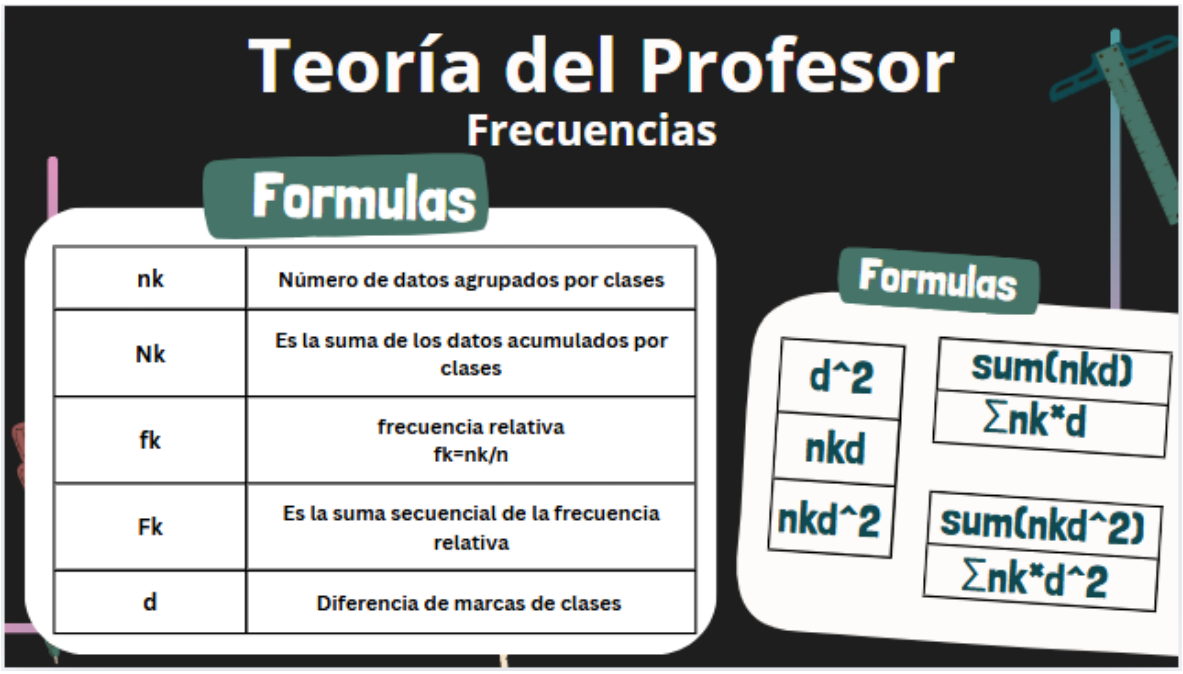


Posteriormente, procedemos al cálculo de los límites de los intervalos:

$Li = Xmin - md$ $Ls = Li + ir$

De esta manera, se construyen los intervalos de forma precisa.

Frecuencias: Cálculo del número de observaciones por intervalo.



Una vez organizados los intervalos, procedemos a calcular distintas frecuencias:

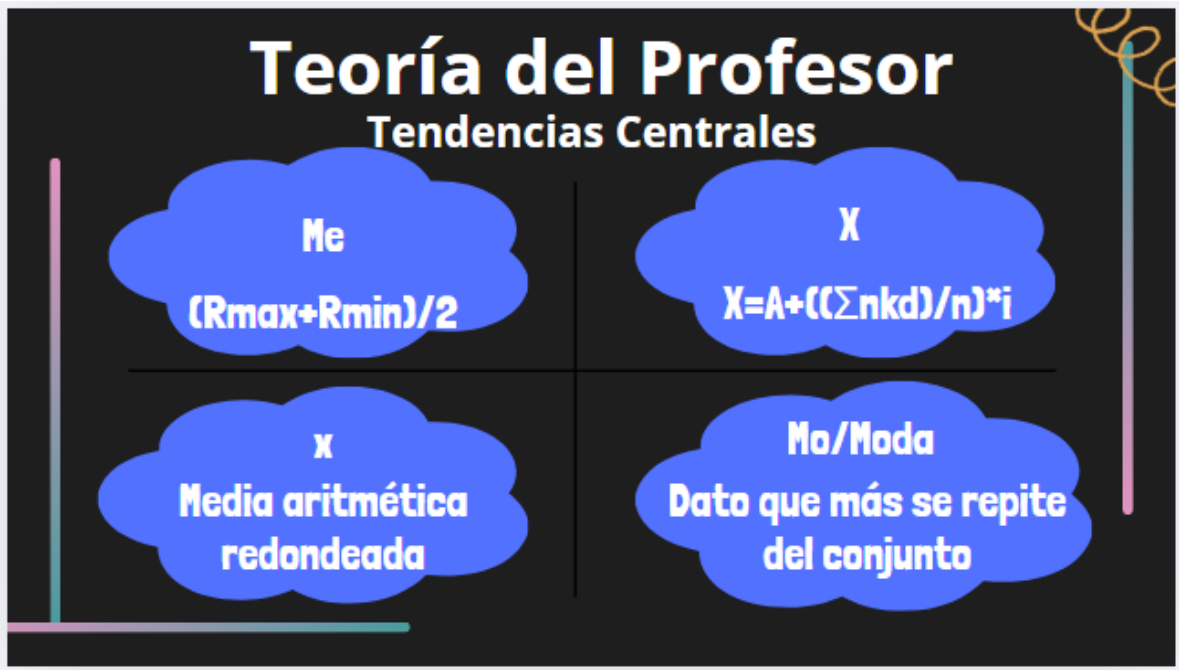
Frecuencia absoluta: Número de datos en cada clase.

Frecuencia absoluta acumulada: Suma progresiva de los nk.

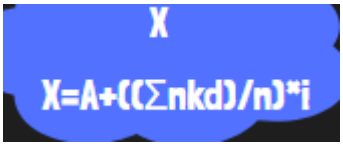
Frecuencia relativa: Proporción de datos en cada clase respecto al total

Frecuencia relativa acumulada: Suma progresiva de las frecuencias relativas.

Tendencias Centrales: Obtención de la media, mediana y moda de los datos.



Ahora, para descubrir la ubicación central de los datos, calculamos la media, que vendría siendo el promedio de los datos y se vería representado por esta formula



Donde A, es la amplitud, nkd el producto entre la frecuencia absoluta y la distancia al punto medio, e i el intervalo.

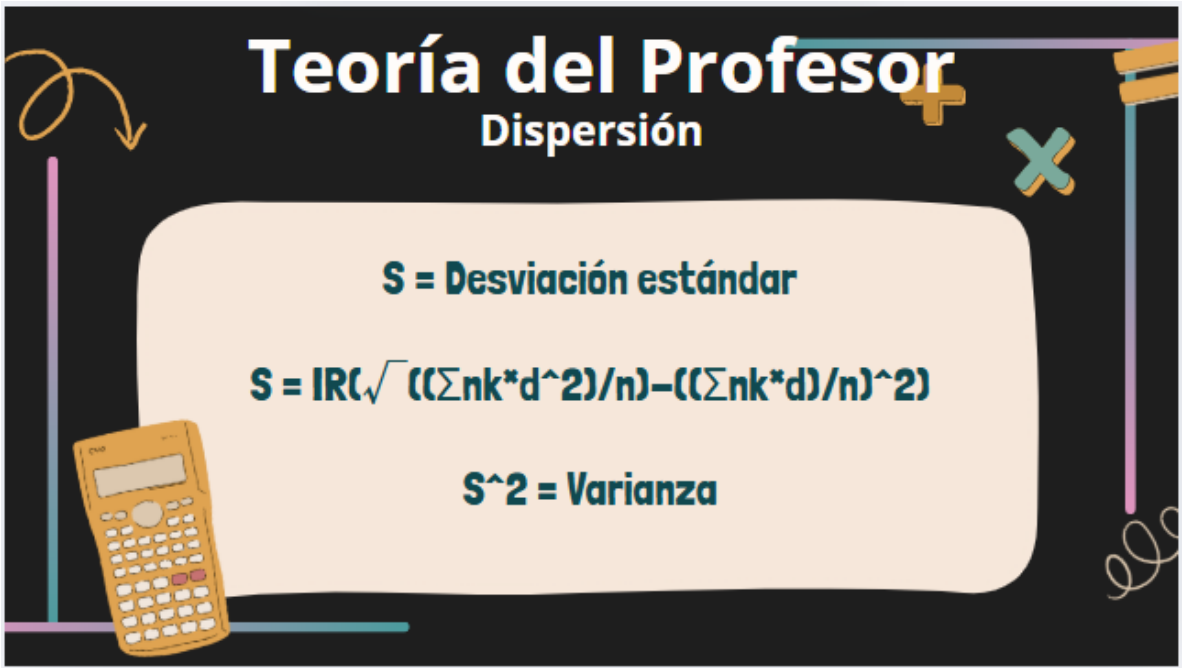
La Mediana, por su parte vendría siendo:

El valor central que divide el conjunto en dos partes iguales.

Mientras que la Moda sería:

El valor o clase con mayor frecuencia de aparición.

Dispersión: Evaluación de la variabilidad mediante desviación estándar y rango.



La dispersión mide qué tan alejados están los datos respecto a la media.

Procedemos a calcular la varianza y la desviación estándar utilizando esta expresión:

$$S = IR(\sqrt{((\sum nk*d^2)/n)-((\sum nk*d)/n)^2})$$

Dónde:

∑nkd es la suma de los productos entre la frecuencia y el dato,

∑(nkd)^2 es la suma de los cuadrados de estos productos.

Estos valores permiten interpretar de forma cuantitativa la variabilidad de la muestra.

5. Planteamiento del Problema

En la actualidad, el crecimiento exponencial del volumen de datos procesados en entornos tecnológicos y científicos ha generado la necesidad de contar con algoritmos de ordenamiento que sean no solo precisos, sino también altamente eficientes en términos de tiempo y recursos computacionales. Esta exigencia se vuelve aún más relevante en el contexto de las Tecnologías de la Información y la Comunicación (TIC), donde la rapidez en

el procesamiento de grandes conjuntos de datos puede marcar una diferencia significativa en el rendimiento global de un sistema.

En este marco, surge la necesidad de evaluar el comportamiento y la eficiencia de algoritmos de ordenamiento modernos, siendo **Timsort** uno de los más utilizados en lenguajes de programacizxón como Python y Java debido a su capacidad para combinar las ventajas de algoritmos clásicos como Merge Sort e Insertion Sort. Sin embargo, su desempeño puede variar dependiendo del tamaño y la estructura de los datos de entrada, lo cual plantea interrogantes sobre su eficacia real en distintos escenarios de aplicación.

El presente estudio se propone analizar estadísticamente el rendimiento de Timsort al ordenar conjuntos de datos de diversos tamaños, con el fin de identificar patrones de comportamiento, detectar posibles cuellos de botella y evaluar su estabilidad en términos de tiempo de ejecución. Esta evaluación será fundamental para determinar en qué condiciones Timsort representa una solución óptima y cuándo podría ser necesario considerar alternativas más adecuadas.

Al aplicar técnicas estadísticas sobre los tiempos de ejecución recolectados, se busca aportar evidencia empírica que respalde decisiones de diseño y desarrollo en contextos TIC, donde el uso eficiente de recursos es prioritario.

6. Soluciones a los Problemas

Muestra de Datos:

N = 30 mediciones.

Tamaños de conjuntos de datos: 1000, 5000, 10000, 50000, 100000.

6 mediciones por cada tamaño.

Tabla Ordenada:

Organización de los datos por tamaño y tiempo de ejecución.

Tabla de Rango, Clases e Intervalos:

Cálculo de rango total y división en clases equitativas.

Tabla de Frecuencias:

Representación de la frecuencia absoluta y relativa.

Tabla de Tendencias Centrales:

Media, mediana y moda para los distintos tamaños de datos.

Tabla de Dispersión:

Desviación estándar y amplitud de los tiempos de ejecución.

7. Histograma

Se construyó un histograma combinado donde se representan las frecuencias de los tiempos de ejecución de Timsort para los diferentes tamaños de datos analizados.

Eje X: Tiempo de ejecución.

Eje Y: Frecuencia de aparición.

Colores diferenciados para cada tamaño de conjunto de datos.

8. Conclusiones

- Eficiencia bajo diferentes cargas de datos: Timsort presenta un comportamiento eficiente incluso bajo diferentes niveles de carga, con tiempos de ejecución muy bajos (del orden de microsegundos y milisegundos), especialmente en conjuntos pequeños, lo que evidencia su capacidad de adaptación a distintos volúmenes de datos.
- Baja dispersión y resultados consistentes: El análisis estadístico demostró baja dispersión en los datos, lo cual indica que el rendimiento de Timsort es predecible y estable, sin variaciones significativas entre ejecuciones con entradas similares.
- Confirmación de la hipótesis del investigador: Los resultados muestran que Timsort no solo es eficiente en teoría, sino también en la práctica, como se evidencia en la consistencia de los tiempos recolectados y la aceptación de la hipótesis del investigador. Esto valida su uso en aplicaciones que requieren procesamiento en tiempo real.

Existe estabilidad en las medidas de tendencia central con baja dispersión.

Se acepta la hipótesis del investigador: Timsort mantiene eficiencia incluso con aumentos de tamaño en los datos.

9. Recomendaciones

- **Aplicación práctica de Timsort en entornos críticos:** Timsort es una excelente opción para sistemas que requieren eficiencia y estabilidad en el ordenamiento de datos, como redes sociales, motores de búsqueda y plataformas de comercio

electrónico. Su consistencia en los tiempos de ejecución lo hace ideal para escenarios donde el tiempo de respuesta es esencial.

- **Ampliar el tamaño de la muestra en futuros estudios:** Se recomienda aumentar la cantidad de datos recolectados, especialmente para conjuntos grandes, para asegurar resultados más sólidos. Esto permitirá detectar posibles anomalías y mejorar la representatividad de las conclusiones estadísticas.
- **Comparación con otros algoritmos de ordenamiento:** Evaluar el desempeño de Timsort frente a algoritmos como QuickSort o HeapSort permitiría contextualizar sus ventajas reales. Estas comparaciones pueden orientar decisiones más informadas sobre cuál algoritmo usar en función del tipo de carga de datos.
- **Evaluar impacto en diferentes arquitecturas de hardware:** Sería relevante analizar cómo varía el rendimiento de Timsort según el tipo de hardware utilizado (por ejemplo, procesadores modernos, móviles o servidores). Esto aportaría una perspectiva más completa sobre su escalabilidad y adaptabilidad a distintos entornos tecnológicos.

10. Resumen Final

Evolución histórica hacia algoritmos de ordenamiento más eficientes.

Análisis estadístico aplicado a Timsort.

Validación de su alta eficiencia en entornos TIC.

Ronda de Preguntas

(Espacio reservado para preguntas del público o profesor)