**UNIVERSIDAD**

**AMERICANA**

## **Facultad de Ingeniería y Arquitectura (FIA)**



# **Algoritmos y Estructuras de Datos**



**“Proyecto de investigación como trabajo final de curso”**

# **Integrantes**

* Harvey Mateo González Jiménez
* Elías Fidel Flores Bermúdez
* Carlos Fernando Acuña Rosales

# **Docente**

# Silvia Gigdalia López Ticay

*Managua, Nicaragua*

### *25 de junio del 2025*

### **Introducción**

El presente trabajo de investigación se enfoca en el estudio y análisis de dos algoritmos fundamentales en ciencias de la computación: el algoritmo de ordenamiento por inserción (*Insertion Sort*) y el algoritmo de búsqueda binaria. Se busca comprender su funcionamiento, eficiencia y aplicabilidad en distintos escenarios, con el objetivo de fortalecer la toma de decisiones en el diseño de soluciones computacionales eficientes.

### **Planteamiento del problema**

En el desarrollo de software, especialmente en aplicaciones que manejan grandes volúmenes de datos, la elección de algoritmos adecuados para ordenar y buscar información puede impactar significativamente el rendimiento. Este estudio se centra en evaluar la eficiencia de *Insertion Sort* y búsqueda binaria como opciones viables para conjuntos de datos pequeños a medianos, analizando su comportamiento en términos de tiempo de ejecución y uso de memoria.

### **Objetivo general**

Analizar la eficiencia computacional del algoritmo de ordenamiento por inserción y el algoritmo de búsqueda binaria mediante su implementación, evaluación y comparación con diferentes tamaños de entrada.

### **Objetivos específicos**

* Implementar el algoritmo de ordenamiento por inserción.
* Implementar el algoritmo de búsqueda binaria.
* Evaluar el desempeño de ambos algoritmos en distintos escenarios.
* Realizar un análisis a priori y a posteriori de los algoritmos seleccionados.
* Determinar la eficiencia en términos de tiempo y memoria.

### **Metodología**

**1. Diseño de la investigación:** Experimental, con pruebas controladas de los algoritmos sobre conjuntos de datos definidos.

**2. Enfoque:** Cuantitativo, midiendo tiempo de ejecución y consumo de memoria.

**3. Alcance:** Descriptivo y explicativo, enfocándose en el análisis del comportamiento de los algoritmos.

**4. Procedimiento:**

* Codificación de los algoritmos en Python.
* Generación de datos de prueba de distintos tamaños.
* Ejecución y recolección de métricas.
* Análisis comparativo con gráficas y tablas.

### **Marco conceptual**

* **Algoritmo:** Conjunto de instrucciones definidas para resolver un problema.
* **Ordenamiento por inserción:** Algoritmo que construye la lista ordenada uno a uno, insertando cada elemento en la posición correcta.
* **Búsqueda binaria:** Técnica eficiente que divide la lista ordenada en mitades para encontrar un valor.
* **Análisis a priori:** Evaluación teórica del rendimiento.
* **Análisis a posteriori:** Evaluación práctica mediante ejecución.
* **Notación Big O:** Herramienta para expresar la complejidad algorítmica.

### **Implementación del algoritmo**

Se implementaron los algoritmos de Insertion Sort y Búsqueda Binaria en Python. El algoritmo de ordenamiento por inserción fue utilizado para ordenar listas numéricas y también listas de palabras ingresadas por el usuario. Por su parte, la búsqueda binaria fue aplicada sobre listas previamente ordenadas para encontrar eficientemente un elemento específico.

A continuación, se destacan dos escenarios prácticos implementados:

* **Simulación de llegada de datos en tiempo real:** Se utilizó Insertion Sort para ordenar progresivamente cada nuevo dato ingresado por el usuario. Esta simulación refleja un contexto en el que los datos no están disponibles de una sola vez, sino que llegan secuencialmente.
* **Diccionario digital con búsqueda de palabras:** Se implementó una herramienta donde el usuario ingresa una lista arbitraria de palabras, que luego son ordenadas alfabéticamente utilizando Insertion Sort. A continuación, se ejecuta una búsqueda binaria para localizar una palabra específica. Esta implementación permite manejar una cantidad indefinida de palabras, demostrando la capacidad de ambos algoritmos de escalar en aplicaciones reales de texto.

### **Análisis a Priori**

**Eficiencia espacial:**

* **Insertion Sort:** No necesita espacio extra. Ordena los datos en el mismo lugar donde están, sin crear nuevas listas.
* **Búsqueda binaria:** También trabaja con la misma lista sin crear copias adicionales.

**Eficiencia temporal:**

**Insertion Sort:**

* En el mejor de los casos (si los datos ya están en orden), funciona muy rápido.
* En el peor de los casos (si los datos están completamente desordenados), se vuelve lento, porque tiene que comparar y mover muchos elementos.
* En general, no es el mejor para listas grandes, pero sí es bueno para listas pequeñas o casi ordenadas.

**Búsqueda binaria:**

* Siempre es muy rápida, porque en cada paso descarta la mitad de la lista.
* Eso sí, solo funciona si la lista ya está ordenada.

**Contextos de eficiencia:**

* **Insertion Sort** es útil cuando los datos se van ingresando poco a poco, como en tiempo real, o cuando la lista no es muy grande.
* **Búsqueda binaria** es ideal cuando tenemos que buscar algo en una lista grande que ya está ordenada, como una palabra en un diccionario digital.

### **Análisis a Posteriori**

**Mejor caso:**

* **Insertion Sort:** Si los datos ya están ordenados, casi no tiene que hacer trabajo extra.
* **Búsqueda binaria:** Si la palabra que buscamos está justo en el medio de la lista, la encuentra enseguida

**Caso promedio:**

* En condiciones normales, ambos algoritmos se comportan bastante bien con listas de tamaño pequeño o medio.
* Si los datos están medio ordenados, **Insertion Sort** no se ralentiza tanto.

**Peor caso:**

* **Insertion Sort:** Si los datos están completamente al revés, necesita mucho más tiempo porque tiene que mover muchos elementos.
* **Búsqueda binaria:** Si la palabra no está en la lista, igual tiene que revisar varias veces antes de saberlo, aunque sigue siendo rápida.

### **Resultados esperados**

Presentación de gráficas y tablas que muestren el tiempo de ejecución en distintos tamaños de datos. Se analizarán los puntos de eficiencia y las limitaciones prácticas observadas.

### **Conclusiones**

Se concluirá cuál de los dos algoritmos presenta mejor rendimiento bajo qué condiciones, y se destacarán sus posibles aplicaciones óptimas. También se ofrecerán recomendaciones sobre cuándo preferir estas técnicas frente a alternativas más complejas.

### **Referencias bibliográficas**

* <https://di-algo-monica.blogspot.com/2012/05/analisis-priori-y-prueba-posteriori_23.html>
* <https://www.infor.uva.es/~cvaca/asigs/doceda/tema1-2021.pdf>
* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.