

|  |
| --- |
| Trabajo Integrador - Propuesta de Investigación Cátedra de Programación I |
|  |
| 20 junio  Investigación aplicada en Python  Creado por: Jose Ricardo Miranda y Bruno Mendiberry Comision 17 UTN San Nicolas -Tecnicatura en Programacion Profesor: Cinthia Rigoni-Bruselario, Sebastián |

# “Comparación práctica de algoritmos de búsqueda y ordenamiento con árboles en Python: eficiencia y aplicabilidad”

|  |
| --- |
| Descripción breve Desarrollar un programa en Python que implemente distintos algoritmos de ordenamiento (como Quicksort, Mergesort) y búsqueda (como búsqueda binaria, BFS y DFS en árboles), y permita comparar sus rendimientos en tiempo y espacio, usando estructuras como árboles binarios o AVL. |
| *Este enfoque cubre:*  *Algoritmos de Búsqueda y Ordenamiento*  *Estructuras de datos avanzadas*  *Análisis de eficiencia* |
| Temas a investigar y profundizar  a) Algoritmos de búsqueda y ordenamiento  Búsqueda secuencial vs binaria  BFS (Breadth-First Search) y DFS (Depth-First Search) en árboles  Ordenamientos: BubbleSort, InsertionSort, MergeSort, QuickSort, HeapSort  b) Análisis de eficiencia  Complejidad temporal y espacial (Big O)  Pruebas empíricas con timeit, memory\_profiler  c) Estructuras de datos avanzadas  Árboles binarios, AVL o B-trees  Comparación con listas y diccionarios  Casos de uso (sistemas de archivos, bases de datos, juegos)  1. Introducción  Este trabajo se centra en el análisis y la comparación de distintos algoritmos clásicos de búsqueda y ordenamiento, aplicados sobre estructuras de datos avanzadas como árboles binarios y AVL, utilizando el lenguaje de programación Python. La elección de este tema se basa en su importancia dentro del campo de la programación, ya que estas técnicas son fundamentales para optimizar el rendimiento de aplicaciones que manejan grandes volúmenes de información.  El objetivo general es llevar a cabo una implementación funcional y documentada de estos algoritmos, evaluando su eficiencia a través de pruebas prácticas. Esto permite no solo comprender su funcionamiento teórico, sino también observar su comportamiento real en distintos escenarios. La relevancia del tema radica en que el manejo eficiente de datos es una habilidad clave para cualquier desarrollador de software.  2. Marco Teórico  El desarrollo teórico de este trabajo se encuentra detallado en un documento complementario titulado “Marco Teórico Trabajo Integrador.docx”. En él se describen en profundidad los algoritmos de ordenamiento (BubbleSort, MergeSort, QuickSort), búsqueda (lineal, binaria, BFS y DFS), y las estructuras de datos avanzadas como el árbol binario y el árbol AVL. También se analiza la eficiencia computacional mediante la notación Big-O y se presentan comparaciones de rendimiento entre algoritmos. Este marco teórico se basa en bibliografía académica, documentación oficial de Python y recursos especializados.  3. Caso Práctico  Se implementó un programa en Python con estructura modular, que permite al usuario:  - Ordenar listas utilizando diferentes algoritmos (BubbleSort, MergeSort, QuickSort).  - Realizar búsquedas con los métodos lineal y binario.  - Insertar y recorrer elementos en un árbol AVL.  - Comparar tiempos de ejecución de los algoritmos mediante una utilidad de medición.  El código está comentado, indentado correctamente y organizado en carpetas. Se utilizaron estructuras como listas, árboles binarios y AVL. El archivo principal (main.py) funciona como menú de navegación y fue diseñado sin el uso de la instrucción 'break', utilizando estructuras de control más limpias.    *Figura 1. Codigo Menu Principal”*    *Figura 2. Ejecucion del Menu Principal*    *Figura 3. Resultado de ejecución opción 1*    *Figura 4. Resultado de ejecución opción 2*    *Figura 5. Resultado de ejecución opción 3*    *Figura 6. Resultado de ejecución opción 4*    *Figura 7. Resultado de ejecución de una opcion que no esta disponible, vuelve a pedir que ingrese una opcion.*    *Figura 8. Resultado de ejecución de una opcion de salida.*  4. Metodología Utilizada  El desarrollo del trabajo se llevó a cabo en varias etapas bien definidas:  - Se inició con una investigación bibliográfica basada en libros, documentación oficial de Python y recursos académicos.  - Luego, se diseñó la estructura del proyecto modularizando el código en carpetas según la funcionalidad: ordenamiento, búsqueda, árboles y utilidades.  - Se implementaron los algoritmos seleccionados (BubbleSort, MergeSort, QuickSort, búsqueda lineal y binaria, y árboles AVL) en Python, verificando su correcto funcionamiento.  - Se realizaron pruebas de rendimiento con datos aleatorios generados, usando la biblioteca 'time' para medir tiempos de ejecución.  - Se utilizó Git para control de versiones, organizando el código en un repositorio accesible.  - Finalmente, se documentó todo el proceso y se preparó una presentación en video para exponer los resultados.  Las herramientas utilizadas incluyeron: VS Code como entorno de desarrollo, Python 3.11.9, bibliotecas estándar (time, random), y la terminal integrada para ejecutar pruebas.  5. Resultados Obtenidos  Los algoritmos fueron probados utilizando listas de 1000 elementos generadas aleatoriamente. Los tiempos de ejecución evidenciaron una gran diferencia entre ellos:  - BubbleSort fue el más lento, confirmando su complejidad O(n²).  - MergeSort y QuickSort mostraron una eficiencia considerablemente mayor, especialmente este último en promedio.  - En las búsquedas, la búsqueda binaria fue mucho más eficiente que la lineal, especialmente en listas grandes y ordenadas.  - La estructura de árbol AVL funcionó correctamente insertando y recorriendo valores, aunque no se implementó aún la lógica de rotación para balanceo completo.  Se corrigieron errores de importación al organizar el proyecto, y se evitó el uso de instrucciones como 'break' para mejorar la claridad del flujo del programa.  El código fue probado en múltiples ejecuciones y validado mediante salidas en consola.  6. Conclusiones  El trabajo permitió aplicar conocimientos sobre estructuras de datos y algoritmos en un proyecto práctico, observando su comportamiento real. Se comprendió la diferencia de eficiencia entre métodos de ordenamiento y búsqueda, y la utilidad de estructuras como los árboles AVL. Además, se reforzaron habilidades de programación modular, control de versiones con Git, y documentación técnica. Como mejora futura, se propone implementar rotaciones AVL completas y agregar visualización gráfica con matplotlib o interfaces interactivas.  7. Bibliografía  • Hetland, M. L. (2010). Python Algorithms: Mastering Basic Algorithms in the Python Language. Apress.  • Bhargava, A. (2016). Grokking Algorithms: An illustrated guide for programmers and other curious people. Manning Publications.  • Miller, B. N., & Ranum, D. L. (2022). Problem Solving with Algorithms and Data Structures using Python. Runestone Interactive. https://runestone.academy/  • GeeksForGeeks. (s.f.). Data Structures and Algorithms. https://www.geeksforgeeks.org/data-structures/  • Python Software Foundation. (s.f.). The Python Tutorial. https://docs.python.org/3/  • Universidad de Alicante. (s.f.). Estructuras de Datos y Algoritmos en Python. <https://www.dlsi.ua.es/asignaturas/eda/temas.html>  8. Anexos  A continuación, se listan los elementos complementarios que acompañan al trabajo práctico:  - Capturas del programa en ejecución: se incluyen imágenes del menú principal, ejecución de algoritmos de ordenamiento y búsqueda, y recorrido del árbol AVL.  - Enlace al video explicativo: https://youtu.be/EWKGsLqU45E  - Archivo ZIP con el código fuente completo: se adjunta el proyecto organizado modularmente en carpetas, con todos los módulos de Python desarrollados.    - Documento Word con el marco teórico detallado.  - Comparación de tiempos de ejecución entre algoritmos, basada en pruebas prácticas incluidas en el módulo de medición. |