

Título: Algoritmos de Búsqueda y Ordenamiento en Python

Alumnos:

Matías Germán Corzo - matimcorzo@gmail.com

Lucas Couchot – lucas.couchot@gmail.com

Materia: Programación 1

Profesor/a: Ariel Enferrel

Fecha de Entrega: 08/06/2025

Contenido

1. Introducción	3
2. Marco Teórico	4
3. Caso Práctico	7
4. Metodología Utilizada	9
5. Resultados Obtenidos	10
6. Conclusiones	15
7. Bibliografía	16
8. Anexos	17

1. Introducción

El tema seleccionado para desarrollar fue el de algoritmos de búsqueda y ordenamiento.

Poder buscar u ordenar es fundamental para cualquier sistema que vaya a trabajar con un volumen de datos considerable ya que resuelve muchos problemas que cualquier usuario pueda tener en el día a día de su organización.

Elegimos esta temática ya que consideramos que son una herramienta fundamental en el desarrollo de sistemas de software y que muy probablemente se requieran en la gran mayoría de nuestros proyectos como futuros profesionales.

Como objetivo principal de este trabajo tenemos el de mostrar cómo, con una aplicación práctica relativamente sencilla, estas herramientas podrían ser de utilidad en un software real.

2. Marco Teórico

Búsqueda Lineal

La búsqueda lineal es un algoritmo que localiza un valor concreto dentro de una lista comprobando cada elemento uno a uno. Empieza por el primer elemento, lo compara con el objetivo y sigue moviéndose por la lista hasta que encuentra el objetivo o llega al final de la lista. Es un algoritmo sencillo e intuitivo.

La búsqueda lineal no necesita que los datos estén ordenados para funcionar, por lo que se utiliza principalmente en conjuntos de datos sin ordenar. Esto lo hace útil en situaciones en las que no es práctico ordenar, o cuando trabajas con datos en bruto. Sin embargo, esta ventaja tiene un coste: no es tan eficaz como otros algoritmos que requieren datos preclasificados.

La búsqueda lineal es ideal en situaciones en las que se trabaja con conjuntos de datos relativamente pequeños o cuando ordenar los datos no es factible.

Búsqueda Binaria

La búsqueda binaria es un potente algoritmo diseñado para encontrar de forma eficiente un valor en un conjunto de datos ordenado. La idea central de la búsqueda binaria es sencilla: En lugar de comprobar cada elemento del conjunto de datos uno a uno, como en una búsqueda lineal, la búsqueda binaria reduce el intervalo de búsqueda a la mitad con cada paso, lo que hace que el proceso sea mucho más rápido.

Así es como funciona:

- Empieza comparando el valor objetivo con el elemento medio del conjunto de datos. El índice del valor medio se calcula mediante la fórmula $\text{medio} = (\text{bajo} + \text{alto})/2$, donde bajo es el índice del primer elemento del intervalo de búsqueda actual y alto es el índice del último elemento.

- Compara el valor medio con el objetivo. Si el valor objetivo es igual al elemento del medio, has encontrado el índice y la búsqueda ha terminado. Si el valor objetivo es menor que el elemento del medio, la búsqueda continúa en la mitad izquierda del conjunto de datos. Si el valor objetivo es mayor, la búsqueda continúa en la mitad derecha del conjunto de datos.
- Repite los pasos 1-2. El intervalo de búsqueda se reduce continuamente a la mitad en cada paso. Repite el proceso hasta encontrar el valor objetivo o hasta que el intervalo de búsqueda quede vacío.

Este proceso de reducción a la mitad es lo que hace que la búsqueda binaria sea tan eficiente. Sin embargo, es importante tener en cuenta que el conjunto de datos debe estar ordenado para que la búsqueda binaria funcione correctamente. Si los datos no están ordenados, el algoritmo no funcionará como es debido.

Ordenamiento con Bubble Sort

El algoritmo Bubble Sort es una técnica de clasificación basada en la comparación que compara repetidamente elementos adyacentes y los intercambia si están en el orden incorrecto.

Deriva su nombre de cómo los elementos más pequeños "burbujean" en la parte superior de la lista, similar a las burbujas que se elevan en un líquido.

El algoritmo procede iterativamente hasta que se ordena toda la lista.

Comprendamos los pasos clave involucrados en el algoritmo Bubble Sort:

- Comenzando desde el principio de la lista, compare cada par de elementos adyacentes.
- Si los elementos están en el orden incorrecto (p. ej., el elemento actual es mayor que el siguiente elemento en orden ascendente), cámbielos.
- Pase al siguiente par de elementos y repita el proceso de comparación e intercambio.

- Continúe este proceso hasta que no se necesiten más intercambios, lo que indica que la lista está ordenada.

El algoritmo Bubble Sort, con su sencilla implementación, proporciona una comprensión básica de las técnicas de clasificación. Ordena eficientemente una lista dada de números a través de comparaciones e intercambios repetidos. Aunque puede que no sea el algoritmo más eficiente para grandes conjuntos de datos, tiene importancia en escenarios de pequeña escala o casi ordenados.

Ordenamiento con Quick Sort

El ordenamiento rápido usa dividir y conquistar para obtener las mismas ventajas que el ordenamiento por mezcla, pero sin utilizar almacenamiento adicional. Sin embargo, es posible que la lista no se divida por la mitad. Cuando esto sucede, veremos que el desempeño disminuye.

Un ordenamiento rápido primero selecciona un valor, que se denomina el valor pivote. Aunque hay muchas formas diferentes de elegir el valor pivote, simplemente usaremos el primer ítem de la lista. El papel del valor pivote es ayudar a dividir la lista. La posición real a la que pertenece el valor pivote en la lista final ordenada, comúnmente denominado punto de división, se utilizará para dividir la lista para las llamadas posteriores a la función de ordenamiento rápido.

Este algoritmo es más rápido que bubble sort en la mayoría de los casos.

3. Caso Práctico

El problema que seleccionamos para desarrollar nuestro trabajo fue el de representar una base de estudiantes con sus notas.

La idea es proveer al usuario la posibilidad de cargar los estudiantes con sus notas y darle la posibilidad de imprimir un listado ordenado en base a sus notas y de realizar una búsqueda por nombre.

Creemos que Bubble Sort para ordenamiento sería ideal para este problema por su sencillez y debido a que no vamos a manejar un gran volumen de datos. Si tuviéramos que abordar este sistema para el mundo real podríamos optar por Quick Sort que en general es más rápido.

Para no terminar con un trabajo muy sencillo decidimos implementar ambos algoritmos de ordenamiento para poder compararlos entre sí.

Para ilustrar los algoritmos de búsqueda optamos por implementar la búsqueda lineal. Este algoritmo es más que aceptable para este problema.

En caso de tener un volumen de estudiantes mayor podríamos haber optado por una búsqueda binaria, ordenando previamente la lista.

Código que implementa los algoritmos que seleccionamos:

```
def bubble_sort_por_nota(lista):  
    """  
    Ordena la lista de estudiantes de menor a mayor nota utilizando el algoritmo Bubble Sort.  
    Devuelve una nueva lista ordenada.  
    """  
    n = len(lista)  
    for i in range(n):  
        for j in range(0, n - i - 1):  
            if lista[j]["nota"] > lista[j + 1]["nota"]:  
                lista[j], lista[j + 1] = lista[j + 1], lista[j]  
    return lista
```

```
def bubble_sort_por_nota(lista):  
    """  
    Ordena la lista de estudiantes de menor a mayor nota utilizando el algoritmo Bubble Sort.  
    Devuelve una nueva lista ordenada.  
    """  
    n = len(lista)  
    for i in range(n):  
        for j in range(0, n - i - 1):  
            if lista[j]["nota"] > lista[j + 1]["nota"]:  
                lista[j], lista[j + 1] = lista[j + 1], lista[j]  
    return lista  
  
def quick_sort_por_nota(lista):  
    """  
    Ordena la lista de estudiantes de menor a mayor nota utilizando el algoritmo Quick Sort.  
    Devuelve una nueva lista ordenada.  
    """  
    if len(lista) <= 1:  
        return lista  
    else:  
        pivote = lista[0]  
        menores = [x for x in lista[1:] if x["nota"] <= pivote["nota"]]  
        mayores = [x for x in lista[1:] if x["nota"] > pivote["nota"]]  
        return quick_sort_por_nota(menores) + [pivote] + quick_sort_por_nota(mayores)
```


4. Metodología Utilizada

Con la finalidad de desarrollar este trabajo profundizamos los temas principalmente basándonos en el contenido brindado por la cátedra el cual nos permitió comprender el propósito de los algoritmos y nos permitió elegir qué usar para nuestro problema. Encontramos de gran utilidad principalmente el notebook provisto con ejemplos y explicaciones del funcionamiento de los diferentes algoritmos.

Para implementar el código hicimos uso del entorno Visual Studio Code, utilizando las extensiones oficiales de Python.

Hicimos uso de la librería “timeit” para medir los tiempos de ejecución de algunos algoritmos.

Finalmente, para poder realizar las pruebas ejecutamos varios escenarios con diferente cantidad de estudiantes y diversos valores para sus notas, los cuales se analizarán en el apartado de Resultados Obtenidos.

División de tareas:

Matías Corzo	<ul style="list-style-type: none">- Algoritmo de búsqueda lineal- Algoritmo de ordenamiento bubble sort- Función de carga de estudiantes
Lucas Couchot	<ul style="list-style-type: none">- Algoritmo de ordenamiento quick sort- Selector de acciones en base a entrada de teclado con comparativa de tiempos

5. Resultados Obtenidos

Repositorio en GitHub:

<https://github.com/CorzoMatias/UTN-TUPaD-progra-integracion>

Casos de prueba

Nota: En el apartado de anexos se podrá encontrar capturas con los resultados de ejecución retornados por la aplicación.

Con respecto a la búsqueda podemos mostrar un caso exitoso (estudiante encontrado) y uno no exitoso (estudiante no encontrado).

Caso 1: Búsqueda exitosa por ingreso de texto “Matías” para los datos de entrada:

Nombre	Nota
Martín	5
Matías	3
Lucas	8

Resultados: Alumno encontrado.

Caso 2: Búsqueda fallida por ingreso de texto “Roberto” para los datos de entrada:

Nombre	Nota
Martín	5
Matías	3
Lucas	8

Resultados: Alumno no encontrado.

A continuación ejecutamos tres casos para comparar el rendimiento de los algoritmos de ordenamiento Bubble Sort y Quick Sort ante distintos datos iniciales:

Caso 3:

Nombre	Nota
Paula	10
Pedro	8
Carlos	7
Juan	6
Andrea	5.5
Roberto	5
Matías	4
Ana	3
Lucas	2
Martín	1.5
Sol	1

Resultados:

Tiempo Bubble Sort	Tiempo Quick Sort
3.74e-5 segundos	3.25e-5 segundos

En este caso alimentamos a los algoritmos con una lista previamente ordenada de mayor a menor. Ambos algoritmos se comportan de manera similar, siendo bubble sort ligeramente más lento.

Caso 4:

Nombre	Nota
Paula	1
Pedro	1.5
Carlos	2
Juan	3
Andrea	4
Roberto	5
Matías	5.5
Ana	6
Lucas	7
Martín	8
Sol	10

Resultados:

Tiempo Bubble Sort	Tiempo Quick Sort
2.23e-5 segundos	4.47e-5 segundos

En este caso alimentamos a los algoritmos con una lista ya ordenada de menor a mayor por lo que el Bubble Sort se comporta mejor que el Quick Sort lo que es esperable.

Caso 5:

Nombre	Nota
Paula	8
Pedro	5.5
Carlos	3
Juan	10
Andrea	1
Roberto	4
Matías	6
Ana	2
Lucas	5
Martín	1.5
Sol	7

Resultados:

Tiempo Bubble Sort	Tiempo Quick Sort
2.42e-5 segundos	2.24e-5 segundos

Este escenario parte de una lista desordenada. Podría pensarse en la prueba más “real”. En este caso podemos ver que el Bubble Sort se comporta ligeramente peor que el Quick Sort aunque la diferencia no es realmente considerable.

6. Conclusiones

En la realización de este trabajo aprendimos la importancia de estudiar algoritmos debido a su utilidad para resolver problemas comunes que se puedan presentar en cualquier sistema real.

Un punto importante que notamos es que cuanto más información tenga que manejar nuestro sistema, más relevantes pasan a ser las decisiones de diseño sobre la implementación del mismo en cuanto a algoritmos de ordenamiento y búsqueda.

Por otro lado, en sistemas sencillos o con poco volumen de información podría optarse por implementaciones más sencillas que luego podrían ir actualizándose según los requerimientos vayan cambiando.

Este último punto queda evidenciado por la mínima diferencia de rendimiento que observamos entre los diferentes algoritmos analizados para un volumen pequeño de datos.

Como futuras mejoras a este trabajo podrían considerarse varios puntos:


- Automatizar la generación de casos de prueba para hacer pruebas con grandes volúmenes de información.
- Incorporar implementaciones de algoritmos de búsqueda para poder comparar con la búsqueda lineal.
- Tener en cuenta el uso de memoria y no sólo los tiempos de ejecución para evaluar el rendimiento de los algoritmos.

7. Bibliografía

- Amberle McKee (8 de Noviembre de 2024). Búsqueda lineal en Python: Guía para principiantes con ejemplos.
<https://www.datacamp.com/es/tutorial/linear-search-python>
- Amberle McKee (3 de Septiembre de 2024). Búsqueda binaria en Python: guía completa para una búsqueda eficiente.
<https://www.datacamp.com/es/tutorial/binary-search-python>
- Ishita Juneja (27 de Junio de 2023). ¿Qué es el algoritmo de clasificación de burbujas para números?.
<https://hackernoon.com/lang/es/cual-es-el-algoritmo-de-clasificacion-de-burbujas-para-numeros>
- Runestone Academy. 5.12. El ordenamiento rápido.
<https://runestone.academy/ns/books/published/pythoned/SortSearch/ElOrdenamientoRapido.html>

8. Anexos

Captura ejecución caso 1 de pruebas:

 Listado original de estudiantes:

Martín - Nota: 5.0


Matías - Nota: 3.0


Lucas - Nota: 8.0


Acciones disponibles:

1. Cargar estudiantes
2. Ordenar estudiantes por nota
3. Buscar estudiante por nombre
4. Salir

Ingrese su opción: 3

 Ingrese el nombre del estudiante que desea buscar: Matías

 Resultado de búsqueda para 'Matías':


 Nombre: Matías - Nota: 3.0


Captura ejecución caso 2 de pruebas:


Acciones disponibles:

1. Cargar estudiantes
2. Ordenar estudiantes por nota
3. Buscar estudiante por nombre
4. Salir

Ingrese su opción: 3


 Ingrese el nombre del estudiante que desea buscar: Roberto


 Resultado de búsqueda para 'Roberto':

 Estudiante no encontrado.

Captura ejecución caso 3 de pruebas:


Tiempo Bubble Sort: 3.7400051951408386e-05 segundos
Tiempo Quick Sort: 3.25001310557127e-05 segundos


 Estudiantes ordenados por nota (de menor a mayor) - Bubble Sort:
Sol - Nota: 1
Martín - Nota: 1.5
Lucas - Nota: 2
Ana - Nota: 3
Matías - Nota: 4
Roberto - Nota: 5
Andrea - Nota: 5.5
Juan - Nota: 6
Carlos - Nota: 7
Pedro - Nota: 8
Paula - Nota: 10

 Estudiantes ordenados por nota (de menor a mayor) - Quick Sort:
Sol - Nota: 1
Martín - Nota: 1.5
Lucas - Nota: 2
Ana - Nota: 3
Matías - Nota: 4

Captura ejecución caso 4 de pruebas:

Tiempo Bubble Sort: 2.229982055723667e-05 segundos
Tiempo Quick Sort: 4.469999112188816e-05 segundos


 Estudiantes ordenados por nota (de menor a mayor) - Bubble Sort:
Paula - Nota: 1
Pedro - Nota: 1.5
Carlos - Nota: 2
Juan - Nota: 3
Andrea - Nota: 4
Roberto - Nota: 5
Matías - Nota: 5.5
Ana - Nota: 6
Lucas - Nota: 7
Martín - Nota: 8
Sol - Nota: 10

 Estudiantes ordenados por nota (de menor a mayor) - Quick Sort:
Paula - Nota: 1
Pedro - Nota: 1.5
Carlos - Nota: 2

Captura ejecución caso 5 de pruebas:

Tiempo Bubble Sort: 2.4200184270739555e-05 segundos

Tiempo Quick Sort: 2.2399937734007835e-05 segundos

 Estudiantes ordenados por nota (de menor a mayor) - Bubble Sort:

Andrea - Nota: 1

Martín - Nota: 1.5

Ana - Nota: 2

Carlos - Nota: 3

Roberto - Nota: 4

Lucas - Nota: 5


Pedro - Nota: 5.5

Matías - Nota: 6

Sol - Nota: 7

Paula - Nota: 8

Juan - Nota: 10

 Estudiantes ordenados por nota (de menor a mayor) - Quick Sort:

Andrea - Nota: 1

Martín - Nota: 1.5

Ana - Nota: 2

Carlos - Nota: 3

Roberto - Nota: 4

Lucas - Nota: 5

Pedro - Nota: 5.5

Matías - Nota: 6