Algoritmos de Búsqueda y Ordenamiento

Alumnos:
Nahuel Ayala – nahuelayala30@gmail.com
Facundo Miguel Archiria – sky_facundo@hotmail.com

**Materia: ** Programación I

**Profesor: ** Bruselario, Sebastián

**Fecha de Entrega: ** 9 de junio de 2025

Índice

Introducción

Los algoritmos de búsqueda y ordenamiento son fundamentales en la programación, no solo por la amplia variedad de implementaciones y aplicaciones que presentan, sino también por su importancia en la optimización del código. Estos algoritmos permiten gestionar eficientemente distintos volúmenes de datos, facilitando la entrega de información relevante al usuario final.

El objetivo de este trabajo es analizar cómo influye el ordenamiento de una lista en la eficiencia de los algoritmos de búsqueda.

Marco Teórico

Los algoritmos de ordenamiento permiten reorganizar una colección de elementos según un criterio especifico.

Es unos de los algoritmos más estudiados al tener múltiples implementaciones para resolver una amplia gama de problemas, como por ejemplo:

- Búsqueda: la búsqueda de un elemento es más eficiente en una lista ordenada.
- Selección: seleccionar elementos según ciertos criterios es más fácil con los datos organizados.
- Duplicados: Identificar elementos repetidos es más rápido cuando la lista está

ordenada.

- Distribución: Evaluar la frecuencia de ciertos elementos es más eficiente con datos ordenados.

Entre los algoritmos de ordenamiento más comunes se encuentran:

- Bubble sort: El ordenamiento por burbuja consiste en realizar pasadas a través de una lista, comparando los elementos adyacentes donde el elemento más grande asciende a la posición correcta.
- Insertion sort: Construye la lista ordenada insertando cada nuevo elemento en su posición correcta, comparándolo con los elementos ya ordenados.
- Quicksort: Selecciona un elemento pivote y divide la lista en dos sublistas: una con elementos menores al pivote y otra con elementos mayores. Luego aplica el mismo proceso de forma recursiva hasta obtener una lista completamente ordenada.

Por otro lado, los algoritmos de búsqueda son esenciales para localizar elementos específicos dentro de una colección de datos.

Entre los algoritmos de búsqueda más comunes se encuentran:

- Búsqueda lineal: Se utiliza para matrices sin ordenar. Realiza principalmente comparaciones individuales del elemento buscado con los elementos de la lista.
- Búsqueda binaria: Se utiliza para una lista ordenada. Compara primero el elemento central de la matriz y si este es igual a la entrada devuelve el resultado. De lo contrario, busca en la mitad izquierda o derecha según el resultado de la comparación.

Caso Práctico

Se desarrolló un programa en Python con el objetivo de analizar el impacto del ordenamiento sobre la eficiencia de los algoritmos de búsqueda. Se generó una lista de números aleatorios y se aplicó el algoritmo de ordenamiento Bubble sort. Luego se evalúo los dos algoritmos de búsqueda: búsqueda lineal sobre la lista desordenada y búsqueda binaria sobre la lista ordenada.
Se midió el tiempo de ejecución de cada uno utilizando la librería time, buscando comparar el rendimiento de cada método para localizar un mismo elemento dentro de la lista.

main.py > ...

```python
import time, random

#creación de lista
lista = [random.randint(0, 1000) for i in range(20000)]
#elección de un valor que se sabe que existe en la lista
elemento = random.choice(lista)

#Algoritmo Bubble Sort
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr

# Búsqueda lineal
def busqueda_lineal(arr, valor):
    for i in range(len(arr)):
        if arr[i] == valor:
            return i, arr[i]
    return -1

# Búsqueda binaria
def busqueda_binaria(arr, valor):
    izquierda = 0
    derecha = len(arr) - 1
    while izquierda <= derecha:
        medio = (izquierda + derecha) // 2
        if arr[medio] == valor:
            return medio, arr[medio]
        elif arr[medio] < valor:
            izquierda = medio + 1
        else:
            derecha = medio - 1
    return -1

# Búsqueda lineal en lista desordenada
# se utiliza el metodo time para medir el tiempo desde el comienzo
# hasta finalizar el proceso.
inicio_lineal = time.time()
elemento_lineal = busqueda_lineal(lista, elemento)
fin_lineal = time.time()
```

```python
# Ordenar la lista
#se utiliza el metodo sort, para identificar la variabilidad de tiempo entre
#diferentes volúmenes de datos
inicio_sort = time.time()
lista_ordenado = bubble_sort(lista)
fin_sort = time.time()

# Búsqueda binaria
# se utiliza el mis
inicio_binaria = ti
elemento_binario =
fin_binaria = time.time()

# Se imprimen los resultados de los tiempos de ejecución
print(f"Volúmen de datos: {len(lista)}, Elemento a buscar: {elemento}")
print(f"Ordenamiento por búrbuja: {fin_sort - inicio_sort:.6f} segundos")
print(f"Búsqueda lineal (lista desordenada): indice: {elemento_lineal[0]}, Valor encontrado: {elemento_lineal[1]}, {fin_lineal - inicio_lineal:.6f} segundos")
print(f"Búsqueda binaria (lista ordenada): indice: {elemento_binario[0]}, Valor encontrado: {elemento_binario[1]}, {fin_binaria - inicio_binaria:.6f} segundos")
```

```
(function) def time() -> float
Return the current time in seconds since the Epoch.
Fractions of a second may be present if the system clock provides them.
```

Metodología Utilizada

La elaboración del trabajo se realizó en las siguientes etapas:

- Recolección de información teórica en documentación confiable.
- Implementación en Python de los algoritmos estudiados.
- Pruebas con diferentes conjuntos de datos.
- Registro de resultados y validación de funcionalidad.
- Elaboración de este informe y preparación de anexos.

Resultados Obtenidos

- Correcta creación de una lista con número aleatorios desordenados.
- Correcto ordenamiento con la utilización del algoritmo Bubble sort, identificando lentitud para listas de gran cantidad de números.
- Correcta búsqueda de un elemento con el algoritmo Búsqueda lineal.
- Correcta búsqueda de un elemento con el algoritmo Búsqueda binaria.
- Correcta recolección de resultados de eficiencia en tiempo de ejecución utilizando la librería time.
- Se comprendieron la diferencia entre los dos algoritmos de búsqueda y cómo influye en la eficiencia una lista ordenada.

Conclusiones

A través del Caso práctico implementado se comprobó la eficiencia del Algoritmo de búsqueda binaria, que requiere una lista ordenada como condición, al aplicarlos sobre listas con distintos volúmenes de datos (1000, 5000, 10000, 15000 y 20000, véase capturas de resultados en el anexo) en comparación con el algoritmo de búsqueda lineal.

Si bien la búsqueda lineal tiene la ventaja de no requerir una lista ordenada, que es eficiente para volúmenes pequeños, menores a 1000 elementos, sus tiempos se vuelven menos eficientes a medida que aumenta el tamaño de la lista. En el caso analizado, fue únicamente con 1000 elementos donde superó levemente a la búsqueda binaria.

Por otro lado, la búsqueda binaria tiene como costo adicional el tiempo que conlleva el ordenamiento, demostró ser claramente más rápida en todos los casos de mayor volumen.

En esta práctica se utilizó el algoritmo ordenamiento por burbuja (Bubble sort) que mediante los resultados obtenidos se evidenció claramente no ser eficiente para grandes volúmenes de datos, aun así la búsqueda binaria compensó este costo realizando búsquedas en aproximadamente la mitad de tiempo que la búsqueda lineal.

Lo que llevó a la conclusión, que en un entorno de grandes cantidades de datos, es más factible pagar el costo de ordenamiento inicial, para luego realizar búsquedas binarias en la mitad de tiempo que llevaría realizarlo con búsqueda lineal.

Bibliografía

- GeeksforGeeks. (n.d.). *Searching algorithms*. GeeksforGeeks. De https://www.geeksforgeeks.org/searching-algorithms/

- Leung, C. *Sorting algorithms in Python*. Real Python. https://realpython.com/sorting-algorithms-python/

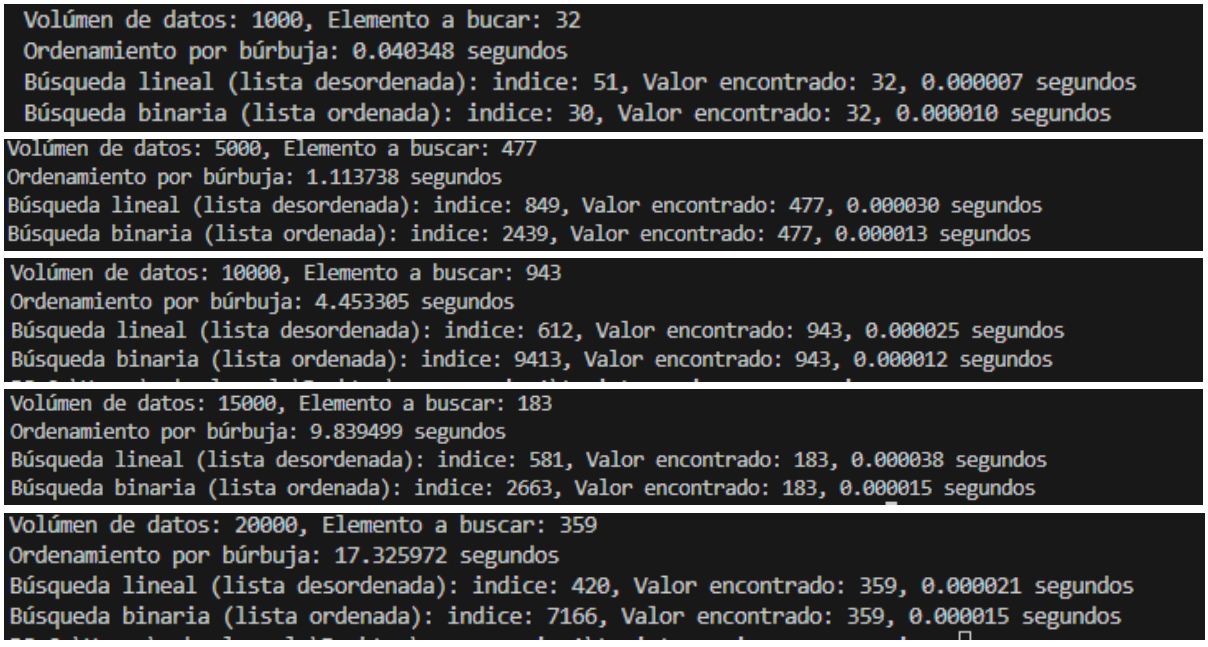
- Leung, C. (2023, enero 17). *How to use binary search in Python*. Real Python. https://realpython.com/binary-search-python/

- Python Textbook. (n.d.). *Sorting and searching algorithms*. *Python 3 Textbook*. de https://python textbok.readthedocs.io/en/latest/Sorting_and_Searching_Algorithms.html

- Python Software Foundation. (n.d.). *time — Time access and conversions*. *The Python Standard Library* de https://docs.python.org/3/library/time.html#module-time

Anexos

- Capturas de pantalla del programa funcionando.



- Repositorio en GitHub: https://github.com/NahuelAyala97/tp-integracion-programacion

- Video explicativo: https://www.youtube.com/watch?v=H7TKF7jj6qs