Trabajo Práctico Integrador
Algoritmos de Búsqueda y Ordenamiento

Alumnos:
Matias Gonzalez – matias92meg@gmail.com
Materia: Programación I
Profesor: Ariel Enferrel
Fecha de Entrega: 9 de junio de 2025

índice:

Introducción

Marco Teórico

Para entender cómo funcionan los algoritmos de búsqueda y ordenamiento, primero tenemos que entender, por un lado, qué es una búsqueda, cuáles los distintos tipos de búsqueda, cuáles son las ventajas y desventajas de cada uno de ellos y en función a esto cómo elegir el tipo de búsqueda correcto para cada caso y, por otro lado, qué es el ordenamiento, cuáles son los distintos tipos de ordenamiento que hay, ventajas y desventajas de cada uno de ellos.

Búsqueda

En programación, podemos definir a una Búsqueda como la técnica y/o el proceso mediante el cual un algoritmo recorre una estructura de datos para localizar uno o varios elementos que cumplan con una condición específica. Es una operación fundamental para muchas aplicaciones, cómo base de datos, sistemas de archivos e incluso inteligencia artificial.

Existen distintos tipos de algoritmos de búsqueda:

- ***Búsqueda lineal:***
  Es el algoritmo de búsqueda más simple, que compara cada elemento de la estructura de datos con el valor objetivo de manera secuencial hasta encontrar una coincidencia o hasta

que se revisen todos los elementos. Es un método fácil de implementar, aunque su eficiencia suele ser limitada cuando la estructura de datos es grande.

Ventajas:

- ***Simplicidad*:** Es un algoritmo muy fácil de entender e implementar
- ***No requiere ordenar la lista:*** Funciona tanto en listas ordenadas como no ordenadas.

Desventajas:

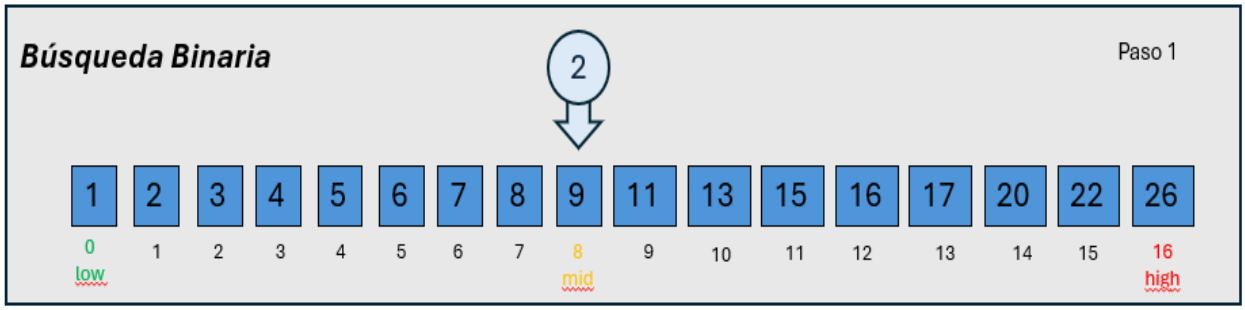
- ***Ineficiencia en grandes volúmenes de datos:*** Tiene una complejidad **O(n)**, lo que significa que el tiempo de ejecución aumenta linealmente con el número de elementos. Si la lista es muy grande, puede volverse lento.

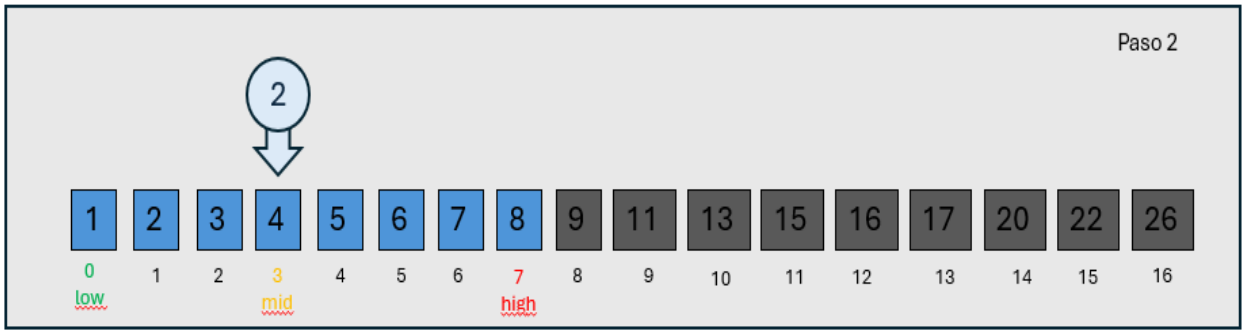
En resumen:

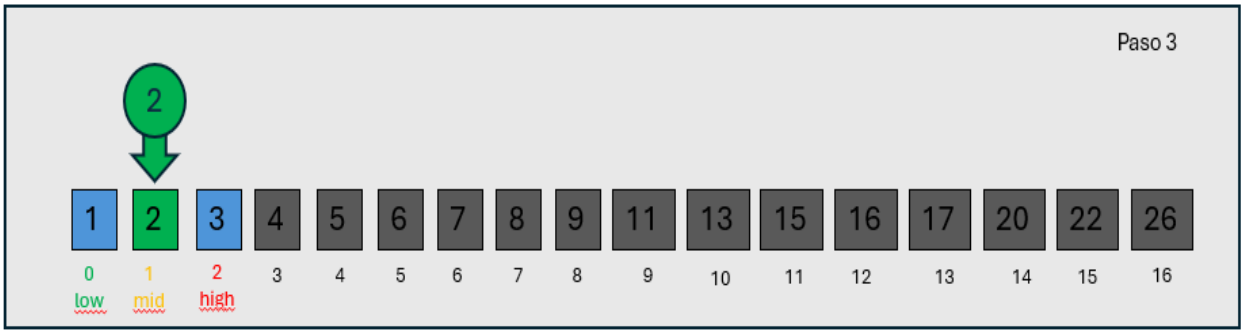
La búsqueda lineal es una técnica de búsqueda sencilla, útil para listas pequeñas o cuando la eficiencia no es crítica. Sin embargo, para listas grandes, se recomienda utilizar algoritmos de búsqueda más eficientes, como la búsqueda binaria, que pueden reducir drásticamente el tiempo de búsqueda.

- **Búsqueda Binaria:**
  Es un algoritmo eficiente para localizar un elemento dentro de una estructura de datos ordenada. Funciona dividiendo repetidamente la estructura en mitades y comparando el valor buscado con el elemento del centro, hasta encontrarlo o determinar que no está en la lista.

  En detalle, teniendo una lista de datos ordenada se divide en dos, compara el valor a buscar con el que se encuentra en la mitad, si el valor coincide termina la búsqueda, en caso de que no coincida comienza un proceso repetitivo donde el rango a buscar depende si el valor a buscar es mayor o menor al que se encuentra en la mitad. Si el valor a buscar es menor al de la mitad, el rango de búsqueda va a ser desde el inicio hasta el de la mitad; de lo contrario el rango de búsqueda es desde la mitad hasta el último dato de la lista, esto se repite hasta que se encuentre el valor o el intervalo de búsqueda este vació.







Ventajas:

- ***Eficiencia*:** En cada paso de búsqueda, reduce el tamaño de la lista a la mitad, lo que hace que la búsqueda sea mucho más rápida que, por ejemplo, la búsqueda lineal.

Desventajas:

- ***Funciona solo en estructuras de datos ordenadas:*** Si la estructura de datos no está ordenada, es necesario ordenarla para aplicar este tipo de algoritmo, lo que implica tiempo de procesamiento extra.

- **Búsqueda de interpolación:**

Este algoritmo de búsqueda mejora la búsqueda binaria al estimar la posición del elemento deseado en función de su valor. Puede ser más eficiente que la búsqueda binaria para conjunto de datos grandes con una distribución uniforme de valores.

Funcionamiento:

1. Calcula la posición estimada del elemento.
2. Compara el valoren la posición estimada con el objetivo
   - Si coincide, termina la búsqueda.
   - Si es menor, ajusta el rango eliminando los elementos de la izquierda.
   - Si es mayor, ajusta el rango eliminando los elementos de la derecha.
3. Repite hasta que el rango sea vacío o se encuentre el objetivo.

Ventajas:

- Puede ser más rápida que la búsqueda binaria en listas uniformemente distribuidas.

Desventajas:

- Si los datos no están distribuidos de manera uniforme, la estimación de posición puede fallar y perder eficiencia

- **Búsqueda de hash:**

Es una técnica que permite acceder a elementos de manera rápida y eficiente mediante una función de hash. Se utiliza en estructuras como tablas hash o diccionarios, donde la clave se convierte en una dirección dentro de la estructura.

Funcionamiento:

1. Calcula el hash de la clave utilizando la función hash
2. Mapea el hash a un índice dentro de la tabla
3. Inserta el valor en ese índice
4. Busca el elemento usando el mismo proceso: aplica la función hash y accede directamente al índice.

Ventajas:

- ***Velocidad de búsqueda:*** La búsqueda de hash permite acceder a los datos en tiempo constante, lo que la hace ideal para aplicaciones donde se necesite rapidez.
- No requiere que los datos estén ordenados.

Desventajas:

- ***Colisiones:*** Si la función hash no es adecuada, puede generar colisiones y disminuir el rendimiento.

Ordenamiento

Los **métodos de ordenamiento** son algoritmos fundamentales en programación que permiten organizar datos (ya sean números, cadenas u otros tipos) en un orden específico (ascendente o descendente). Esta organización facilita posteriormente la tarea de búsqueda y/o análisis de información, entre otras.
El utilizar algoritmos de ordenamiento nos representa algunos beneficios, entre los que se pueden destacar:

- ***Búsquedas más eficientes:*** Una vez ordenados los datos dentro de la estructura, es mucho más fácil buscar un elemento específico, ya que nos brinda la posibilidad de utilizar la búsqueda binaria.
- ***Análisis de datos más fácil:*** Los datos ordenados pueden ser analizados más fácilmente para identificar patrones y tendencias.
- ***Operaciones más rápidas:*** Muchas operaciones, como fusionar dos conjuntos de datos o eliminar elementos duplicados, se pueden realizar de manera más rápida y eficiente en datos ordenados.

Existen muchos algoritmos de ordenamiento diferentes, cada uno con sus propias ventajas y desventajas. Algunos de los algoritmos de ordenamiento más comunes son:

- **Ordenamiento por burbuja (Bubble Sort)**

  El ordenamiento por burbujas es un sencillo algoritmo de ordenamiento que funciona revisando cada elemento con el siguiente, intercambiándolos de posición si están en orden incorrecto. Es considerado un algoritmo de comparación dado que solo utiliza comparaciones para operar elementos.

  Ventajas:
  - ***Sencillez:*** Es un algoritmo sencillo de entender y aplicar.

  Desventajas:
  - ***Tiempo de ejecución:*** Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

- **Ordenamiento rápido (Quick Sort)**

  Es un algoritmo de ordenamiento eficiente. Se basa en el paradigma “Divide y vencerás”: el problema se divide en subproblemas más pequeños y se resuelven de manera recursiva y las soluciones se combinan para obtener el resultado final.

  Funcionamiento:
  1. **Elección del pivote:** En este paso se selecciona un elemento de la estructura como pivote.
  2. **Reordenamiento de la estructura de datos:** En este paso se reordena la estructura quedando, a la izquierda del pivote, los elementos menores que este y a la derecha los mayores.
  3. **Recursión:** Se aplica recursivamente el proceso a la dos sublistas. La recursión continua hasta que las sublistas tengan uno o cero elementos, lo que significa

que ya está ordenada la estructura.

Ventajas:

- ***Eficacia:*** Es adecuado para ordenar grandes volúmenes de datos de forma rápida.
- ***Simplicidad de adaptación:*** Es un algoritmo fácil de entender y adaptar a distintos tipos de lenguaje.

Desventajas:

- ***Dependencia de la elección del pivote:*** El desempeño del algoritmo depende fuertemente de la elección del pivote. Una mala elección del pivote puede afectar significativamente el rendimiento del algoritmo
- ***Problemas con recursión:*** En el peor de los casos, la recursión puede ser excesivamente profunda que, en ciertos entornos o lenguajes, puede provocar desbordamientos de pila (Stack overflow)

- **Ordenamiento por Selección (Selection Sort)**

Es un algoritmo muy sencillo y fácil de entender. Su idea principal es ordenar una lista o arreglo seleccionando, en cada caso, el elemento mínimo de la parte no ordenada y colocarlo en la posición correcta de la parte ordenada.

Funcionamiento:

1. Encuentra el elemento más pequeño del arreglo y lo intercambia con el 1er elemento
2. Encuentra el elemento más pequeño del arreglo (de la parte desordenada) y lo intercambia con el 2do elemento (de la parte ordenada)
3. Encuentra el elemento más pequeño del arreglo (de la parte desordenada) y lo intercambia con el 3er elemento (de la parte ordenada)
4. Repite este proceso de encontrar el siguiente elemento más pequeño y cambiarlo a la posición correcta hasta que se ordene todo el arreglo.

Ventajas:

- ***Simplicidad:*** es un algoritmo muy intuitivo y fácil de implementar.
- ***Número reducido de intercambios:*** Realiza un número relativamente pequeño de intercambios, ya que solo intercambia el elemente mínimo en cada iteración.

Desventajas:

- ***Complejidad temporal:*** El algoritmo necesita recorrer toda la lista en cada iteración, haciendo un número de comparaciones cuadráticos. Esto lo hace ineficiente para listas de gran tamaño.

- **Ordenamiento por inserción (Insertion Sort)**

Es un algoritmo de ordenamiento simple, que construye progresivamente la lista ordenada, elemento por elemento. Se parte del supuesto que el primer elemento ya está ordenado y luego se va tomando e insertando cada elemento en la posición adecuada de la sublista ordenada.

Funcionamiento.

En el ordenamiento por inserción, se compara el elemento "key" con los elementos anteriores. Si los elementos anteriores son mayores, mueve el elemento anterior a la siguiente posición (derecha).
Comienza en el índice 1 y hasta el índice final del arreglo.

Ventajas:

- ***Simplicidad:*** es un algoritmo muy intuitivo y fácil de implementar.

Desventajas:

- ***Complejidad cuadrática:*** Su complejidad en el peor de los casos es O(n^2), lo que lo hace un algoritmo ineficiente para listas con un gran número de elementos.

- **Ordenamiento por Mezcla (Merge Sort)**

Es un algoritmo de ordenamiento basado en el paradigma divide y vencerás. Su estrategia consiste en dividir la estructura de datos en dos mitades, ordenar recursivamente cada mitad y luego fusionarlas para obtener el arreglo completamente ordenado.

Funcionamiento.

Básicamente el funcionamiento se reduce a 3 pasos:

1. **Dividir:** Si la estructura tiene más de un elemento, se divide en dos sublistas. Este proceso se repite recursivamente a cada sublista hasta que cada una contenga un elemento o esté vacía.
2. **Ordenar:** Al queda cada sublista con un único elemento, se considera ordenada de forma natural.
3. **Fusionar:** Se fusionan la dos sublistas ordenadas en una nueva lista ordenada. La fusión se realiza comparando los primeros elementos de cada sublista e insertando el menor en la lista resultante. Se repite el proceso hasta vaciar ambas sublistas.

Ventajas:

- ***Complejidad garantizada:*** Tiene una complejidad O(n log n), lo que lo hace muy predecible en términos de rendimiento.
- ***Adecuado para Datos Grandes:*** Funciona bien con listas grandes.

Desventajas:

- ***Uso de memoria Adicional:*** Requiere memoria adicional para almacenar las sublistas durante la fusión, lo cual puede ser un inconveniente en entornos con recursos de memoria limitados.

Complejidad de los algoritmos

La complejidad de los algoritmos de búsqueda y ordenamiento se mide en términos de cuánto tiempo o espacio necesitan para realizar su tarea a medida que crece el tamaño de la entrada. Se utiliza la notación Big O para expresar esta complejidad, que describe el crecimiento de la complejidad en función del tamaño del conjunto de datos.
En cuanto a los algoritmos de búsqueda:

- *Búsqueda Lineal:*
  - Peor caso: O(n), el elemento buscado es el último de la lista.
  - Mejor caso: O(1), el elemento buscado es el primero de la lista.

- *Búsqueda Binaria:*
  - Peor caso: O(log n), ya que el tamaño de la búsqueda se reduce a la mitad en cada paso.
  - Mejor caso: O(1), el elemento buscado se encuentra en la primera comparación, es decir, el elemento central de la primera iteración.

- *Búsqueda por Interpolación:*
  - Promedio: O(log log n), cuando los datos de distribuyen uniformemente
  - Peor caso: O(n), en situaciones en las que la distribución de los datos es muy desigual, lo que la estimación de la posición sea inexacta.

- *Búsqueda de Hash:*
  - Promedio: O(1), ya que la función de hash permite acceder directamente a la posición donde se encuentra el elemento.
  - Peor caso: O(n), escenario poco frecuente en el que ocurren muchas colisiones y todos los elementos caen en la misma lista encadenada o cuando la función hash no distribuye bien los valores.

En cuanto a los algoritmos de Ordenamiento:

- *Bubble Sort:*
  - Peor y promedio: O(n^2)
  - Mejor caso: O(n), cuando la lista ya se encuentra ordenada.

- *Quick Sort*
  - Promedio: O(n log n)
  - Peor: O(n^2), cuando la elección del pivote no es adecuada y genera particiones muy desbalanceadas.

- *Selection Sort*
  - Siempre: O(n^2)

- *Insertion Sort:*
  - Peor y promedio: O(n^2)
  - Mejor caso: O(n), cuando la los elementos están casi ordenados.

- *Merge Sort:*
  - Todos los casos: O(n log n)

Marco Práctico

Para poner en práctica los conceptos antes mencionados desarrollé un programa en Python para ordenar una lista desordenada de números utilizando algoritmo de ordenamiento por inserción y luego buscar en la lista resultante un número específico utilizando algoritmo de búsqueda binaria.

```python
# Algoritmo de ordenación por inserción
def ordenamiento_insercion(lista_desordenada):
    for i in range(1, len(lista_desordenada)):
        actual = lista_desordenada[i]
        index = i
        """
        Este bucle intercambia los dos número de posición
        mientras que el número anterior sea más grande que el número actual
        """
        while index > 0 and lista_desordenada[index - 1] > actual:
            lista_desordenada[index] = lista_desordenada[index - 1]
            index = index - 1
        lista_desordenada[index] = actual
    return lista_desordenada
```

```python
# Algoritmo de búsqueda binaria
def busqueda_binaria(lista_ordenada, objetivo, inicio, fin ):
    if inicio > fin:
        return -1

    centro = (inicio + fin) // 2
    if lista_ordenada[centro] == objetivo:
        return centro
    elif lista_ordenada[centro] < objetivo:
        return busqueda_binaria(lista_ordenada, objetivo, centro + 1, fin)
    else:
        return busqueda_binaria(lista_ordenada, objetivo, inicio, centro - 1)
```

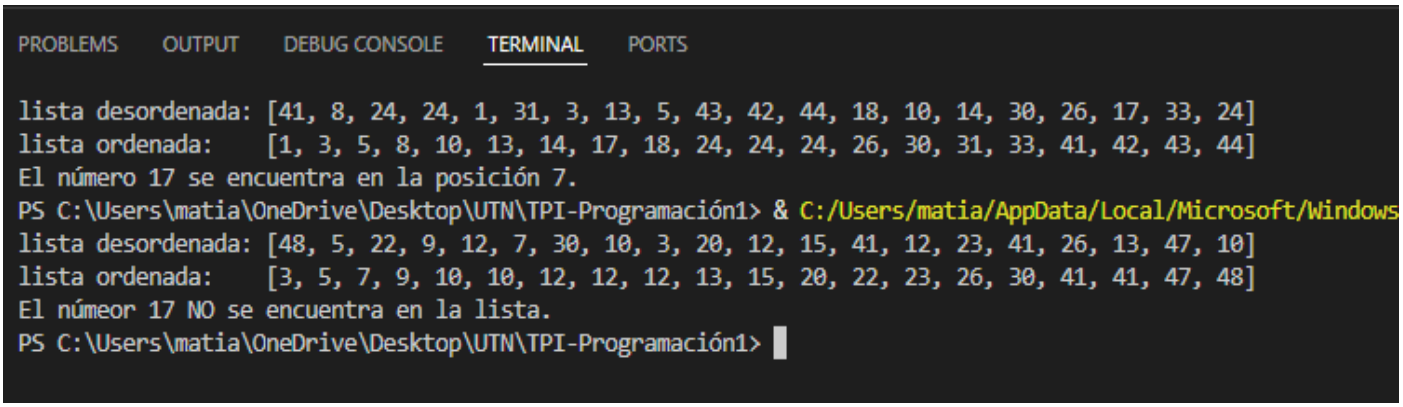
Metodología Utilizada

La elaboración del trabajo práctico se realizó en las siguientes etapas:

- Recolección de información teórica.
- Implementación en Python de los algoritmos aprendidos.
- Pruebas de ambos algoritmos.
- Registro de resultados.
- Preparación de anexos.
- Creación del Repositorio en GitHub

Resultados Obtenidos

- El programa ordenó correctamente la lista de números.
- El algoritmo de búsqueda binaria localizó de manera efectiva el número especificado.
- El algoritmo de búsqueda binaria indicó correctamente cuando el elemento buscado no se encontraba en la lista.
- Se comprendió la complejidad de los distintos algoritmos estudiando.
- Se comprendió la importancia de contar con una estructura de datos correctamente ordenada.



Conclusiones

Los algoritmos de búsqueda y ordenamiento son fundamentales en el desarrollo de software eficiente y en la comprensión profunda de cómo operan las estructuras de datos.
Analizar y comprender algoritmos como búsqueda lineal, binaria, y métodos de ordenamiento como burbuja, selección, inserción, etc., nos permite desarrollar la capacidad de seleccionar la mejor estrategia a implementar según el contexto y la naturaleza de los datos. Esto se traduce en mejor funcionamiento de las aplicaciones.

Bibliografía

- https://4geeks.com/es/lesson/algoritmos-de-ordenamiento-y-busqueda-en-python
- https://visualgo.net/en/sorting
- https://www.freecodecamp.org/espanol/news/algoritmos-de-ordenacion-explicados-con-ejemplos-en-javascript-python-java-y-c/
- https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento
- https://es.khanacademy.org/computing/computer-science/algorithms/binary-search/a/binary-search#:~:text=La%20b%C3%BAsqueda%20binaria%20es%20un,ubicaciones%20posibles%20a%20solo%20una.
- https://www.geeksforgeeks.org/binary-search/
- Documentación del campus de estudio.

Anexos

Repositorio de GitHub: https://github.com/Matias92g/TPIntegrador.P1

Video explicativo: https://youtu.be/FoO__eMbig4?si=q8lPF2HoJil3I8a6