Algoritmos de Ordenamiento, Tuplas y Conjuntos en Python

Algoritmos de Ordenamiento

Los algoritmos de ordenamiento organizan elementos de una lista o arreglo en un orden específico (ascendente o descendente). Son esenciales en informática y se aplican en búsqueda, bases de datos, estructuras de datos y métodos de división y conquista.

Factores a Considerar al Elegir un Algoritmo

- Tamaño de la colección.
- Disponibilidad de memoria.
- Expansión de la colección.
- Estabilidad y eficiencia del algoritmo.

Clasificación de Algoritmos de Ordenamiento

- Tamaño de la lista: Algunos algoritmos funcionan mejor con listas pequeñas.
- Uso de memoria: Algunos necesitan más espacio para trabajar.
- Rapidez: Algunos son más rápidos que otros en ciertos casos.

Tipos de Algoritmos de Ordenamiento

Bubble Sort (Ordenamiento de burbuja): Este método compara elementos adyacentes e intercambia si es necesario, repitiendo el proceso hasta que la lista esté ordenada.

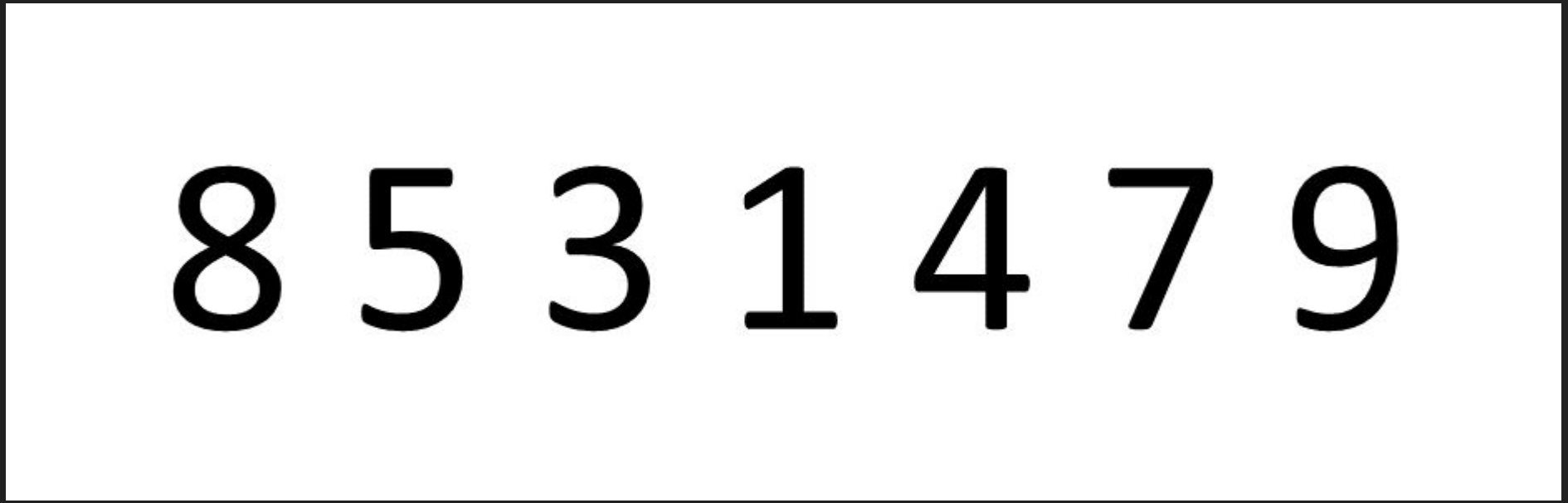
```python
# Algoritmo de ordenamiento burbuja
lista = [64, 34, 25, 12, 22, 11, 90]

def bubble_sort(vector):
    n = len(vector)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if vector[j] > vector[j+1]:
                vector[j], vector[j+1] = vector[j+1], vector[j]
    return vector

print("Lista ordenada:", bubble_sort(lista))
```

Tipos de Algoritmos de Ordenamiento

Bubble Sort (Ordenamiento de burbuja)



Tipos de Algoritmos de Ordenamiento

Selection Sort (Ordenamiento por selección): Encuentra el elemento más pequeño y lo coloca en su posición correcta.

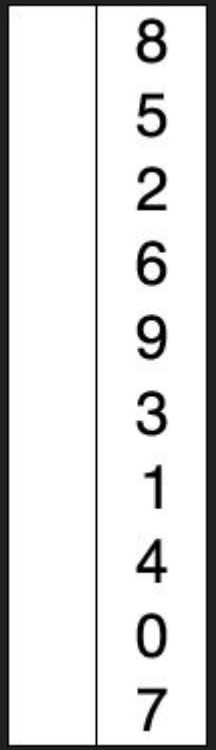
```python
# Algoritmo de ordenamiento por selección
lista = [64, 25, 12, 22, 11]

def selection_sort(lista):
    n = len(lista)
    for i in range(n):
        min_idx = i
        for j in range(i+1, n):
            if lista[j] < lista[min_idx]:
                min_idx = j
        lista[i], lista[min_idx] = lista[min_idx], lista[i]
    return lista

print("Lista ordenada:", selection_sort(lista))
```

Tipos de Algoritmos de Ordenamiento

Selection Sort (Ordenamiento por selección):



Tipos de Algoritmos de Ordenamiento

Insertion Sort (Ordenamiento por inserción): Inserta cada elemento en su posición correcta dentro de una lista ordenada.

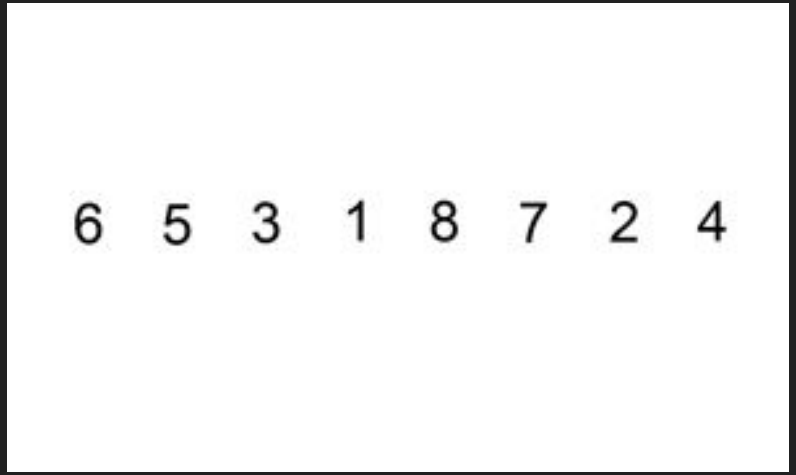
```python
# Algoritmo de ordenamiento por inserción
lista = [5, 2, 9, 1, 5, 6]

def insertion_sort(lista):
    for i in range(1, len(lista)):
        clave = lista[i]
        j = i - 1
        while j >= 0 and clave < lista[j]:
            lista[j + 1] = lista[j]
            j -= 1
        lista[j + 1] = clave
    return lista

print("Lista ordenada:", insertion_sort(lista))
```

Tipos de Algoritmos de Ordenamiento

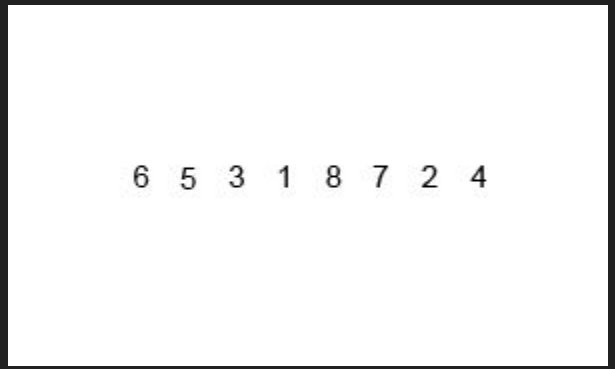
Insertion Sort (Ordenamiento por inserción):



Tipos de Algoritmos de Ordenamiento

Algoritmo de ordenamiento por mezcla (MergeSort)

El Método de ordenamiento por mezcla sigue un proceso específico. Si la lista tiene 0 o 1 elementos, ya se considera ordenada. De lo contrario, el algoritmo divide la lista desordenada en dos sublistas de tamaño aproximadamente igual. Luego, ordena cada sublista de forma recursiva utilizando el mismo método de ordenamiento por mezcla. Finalmente, combina las dos sublistas en una sola lista ordenada.



Tipos de Algoritmos de Ordenamiento

Algoritmo de ordenamiento rápido (QuickSort)

Al igual que el método de ordenamiento por mezcla, el ordenamiento rápido es un algoritmo basado en el principio de dividir y conquistar. Funciona al seleccionar un elemento como pivote y dividir la matriz alrededor de ese pivote elegido. Existen varias versiones del ordenamiento rápido que seleccionan el pivote de diferentes maneras:

Usar siempre el primer elemento como pivote.

Usar siempre el último elemento como pivote.

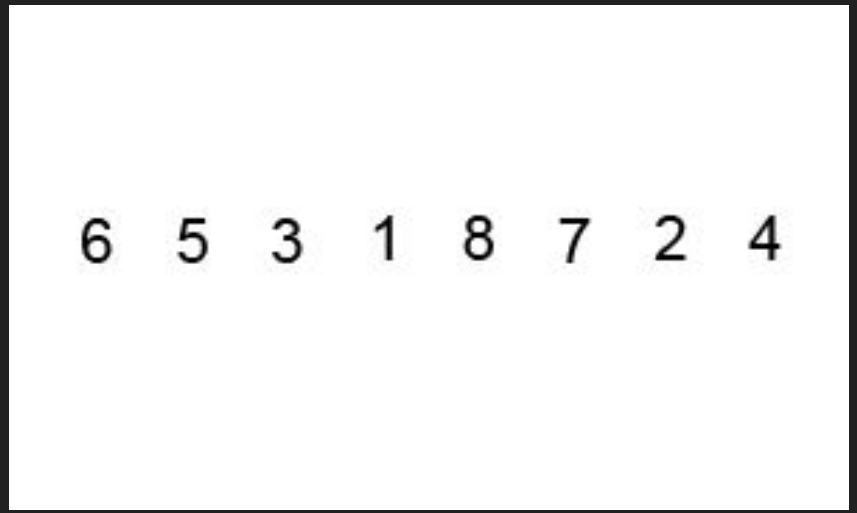
Usar un elemento aleatorio como pivote.

Usar la mitad como pivote.

El proceso central en el ordenamiento rápido es la partición. Su objetivo es, dado un arreglo A y un elemento x como pivote, colocar x en su posición correcta en el arreglo ordenado. Además, se asegura de que todos los elementos menores que x estén antes de x y todos los elementos mayores que x estén después de x en el arreglo. La efectividad del proceso de partición es fundamental para el ordenamiento rápido.

Tipos de Algoritmos de Ordenamiento

Algoritmo de ordenamiento rápido (QuickSort)



Tuplas en Python

¿Qué es una tupla?

Una tupla es una estructura de datos similar a una lista, pero inmutable (no se puede modificar después de su creación). Se usa cuando queremos asegurarnos de que los datos no cambien.

```
tupla = (1, 2, 3, "Hola")
print(tupla)  # (1, 2, 3, 'Hola')
```

Operaciones con Tuplas

Acceso a elementos:

```
print(tupla[0])  # 1
```

Conversión de listas a tuplas:

```
lista = [1, 2, 3]
tupla = tuple(lista)
print(tupla)
```

Desempaquetado:

```
x, y, z, mensaje = tupla
print(mensaje)  # Hola
```

Métodos Tuplas

count(valor) → Cuenta cuántas veces aparece un valor en la tupla.

```
tupla = (1, 2, 3, 1, 1, 4)
print(tupla.count(1))  # 3
```

index(valor, inicio, fin) → Devuelve el índice de la primera aparición de un valor.

```
tupla = (10, 20, 30, 40, 50)
print(tupla.index(30))  # 2
```

Ejemplo

```
libros = [("El Perfume", "Suskind", "Misterio"),
          ("La Metamorfosis", "Kafka", "Novela"),
          ("Misery", "Stephen King", "Misterio")]

clasificacion = input("Ingrese el género: ")
resultado = [libro[0] for libro in libros if libro[2] == clasificacion]
print(resultado if resultado else "No hay libros de ese género")
```

Conjuntos (Set) en Python

¿Qué es un conjunto?

Un conjunto es una colección sin elementos duplicados y sin orden específico.

```
conjunto = {1, 2, 3, 3, 4}
print(conjunto)  # {1, 2, 3, 4}
```

Operaciones con conjuntos

Unión de conjuntos:

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A | B)  # {1, 2, 3, 4, 5}
```

Intersección de conjuntos:

```
print(A & B)  # {3}
```

Diferencia:

```
print(A - B)  # {1, 2}
```

Métodos Conjuntos

add(valor) → Agrega un elemento al conjunto.

```
conjunto = {1, 2, 3}
conjunto.add(4)
print(conjunto)  # {1, 2, 3, 4}
```

remove(valor) → Elimina un elemento (da error si no existe).

```
conjunto = {1, 2, 3}
conjunto.remove(2)
print(conjunto)  # {1, 3}
```

Métodos Conjuntos

discard(valor) → Elimina un elemento sin error si no existe.

```
conjunto = {1, 2, 3}
conjunto.discard(5)  # No da error
```

pop() → Elimina y devuelve un elemento aleatorio.

```
conjunto = {1, 2, 3}
print(conjunto.pop())  # Puede devolver 1, 2 o 3
```

Métodos Conjuntos

clear() → Vacía el conjunto.

```
conjunto = {1, 2, 3}
conjunto.clear()
print(conjunto)  # set()
```

union(set2) → Une dos conjuntos.

```
A = {1, 2, 3}
B = {3, 4, 5}
print(A.union(B))  # {1, 2, 3, 4, 5}
```

Métodos Conjuntos

intersection(set2) → Devuelve la intersección.

```
print(A.intersection(B))  # {3}
```

difference(set2) → Elementos que están en A, pero no en B.

```
print(A.difference(B))  # {1, 2}
```

REVIEW

Descripción:

Se debe desarrollar un programa en Python que permita gestionar una lista de estudiantes, almacenando su información en tuplas y utilizando conjuntos para analizar los cursos inscritos.

Objetivo:

Aplicar listas, tuplas y conjuntos en un mismo ejercicio, permitiendo realizar operaciones básicas como agregar estudiantes, buscar información y analizar cursos.

Instrucciones:

1. Crea un programa en Python que haga lo siguiente:

2. Registrar estudiantes:
   - Cada estudiante se representará con una tupla que almacene: (nombre, edad, curso_inscrito)
   - Los estudiantes se guardarán en una lista.

3. Mostrar la lista de estudiantes registrados.

4. Buscar un estudiante por nombre y mostrar su información.

5. Mostrar los cursos únicos a los que están inscritos los estudiantes (sin repetir) usando un conjunto.

Pistas para desarrollar el ejercicio:

- Usa una lista para almacenar los estudiantes.
- Cada estudiante debe ser una tupla con su información.
- Usa un conjunto para obtener los cursos sin repetir.
- Usa un bucle while para el menú interactivo.
- Usa condicionales (if) para cada opción del menú.